INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

(Approved by AICTE | NAAC Accreditation with 'A++' Grade | Accredited by NBA | Affiliated to JNTUH)
Dundigal, Hyderabad - 500 043, Telangana

OUTCOME BASED EDUCATION
WITH
CHOICE BASED CREDIT SYSTEM

BACHELOR OF TECHNOLOGY
COMPUTER SCIENCE AND ENGINEERING

ACADEMIC REGULATIONS, COURSE CATALOGUE and SYLLABUS
BT25

B.Tech Regular Four Year Degree Program
(for the batches admitted from the academic year 2025 - 2026)

&

B.Tech (Lateral Entry Scheme)
(for the batches admitted from the academic year 2026 - 2027)

These rules and regulations may be altered / changed from time to time by the academic council
FAILURE TO READ AND UNDERSTAND THE RULES IS NOT AN EXCUSE

VISION

To bring forth students, professionally competent and socially progressive, capable of working across cultures meeting the global standards ethically.

MISSION

To provide students with an extensive and exceptional education that prepares them to excel in their profession, guided by dynamic intellectual community and be able to face the technically complex world with creative leadership qualities.

Further, be instrumental in emanating new knowledge through innovative research that emboldens entrepreneurship and economic development for the benefit of wide spread community.

QUALITY POLICY

Our policy is to nurture and build diligent and dedicated community of engineers providing a professional and unprejudiced environment, thus justifying the purpose of teaching and satisfying the stake holders.

A team of well qualified and experienced professionals ensure quality education with its practical application in all areas of the Institute.

PROGRAM OUTCOMES (PO's)	
Engineering Graduates will be able to:	
PO1:	**Engineering Knowledge:** Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization as specified in WK1 to WK4 respectively to develop to the solution of complex engineering problem.
PO2:	**Problem Analysis:** Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development. (WK1 to WK4)
PO3:	**Development of Solutions:** Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for the public health and safety, whole-life cost, net zero carbon, culture, society and environment as required. (WK5).
PO4:	**Conduct Investigations of Complex Problems:** Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions. (WK8).
PO5:	**Engineering Tool Usage:** Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems. (WK2 and WK6).
PO6:	**The Engineer and The World:** Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture and environment. (WK1, WK5, and WK7).
PO7:	**Ethics:** Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws. (WK9)
PO8:	**Individual and Collaborative Team work:** Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams
PO9:	**Communication:** Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences
PO10:	**Project Management and Finance:** Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
PO11:	**Life-Long Learning:** Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change. (WK8).

CONTENTS

"Take up one idea.
Make that one idea your life-think of it, dream of it, live on that idea.
Let the brain muscles, nerves, every part of your body be full of that idea and just leave every other idea alone. This is the way to success"

Swami Vivekananda

PRELIMINARY DEFINITIONS AND NOMENCLATURES

AICTE: Means All India Council for Technical Education, New Delhi.

Autonomous Institute: Means an institute designated as Autonomous by University Grants Commission (UGC), New Delhi in concurrence with affiliating University (Jawaharlal Nehru Technological University, Hyderabad) and State Government.

Academic Autonomy: Means freedom to an institute in all aspects of conducting its academic programs, granted by UGC for Promoting Excellence.

Academic Council: The Academic Council is the highest academic body of the institute and is responsible for the maintenance of standards of instruction, education and examination within the institute. Academic Council is an authority as per UGC regulations and it has the right to take decisions on all academic matters including academic research.

Academic Year: It is the period necessary to complete an actual course of study within a year. It comprises two main semesters i.e., (one odd + one even) and one supplementary semester.

Branch: Means specialization in a program like B.Tech degree program in Aeronautical Engineering, B.Tech degree program in Computer Science and Engineering etc.

Board of Studies (BOS): BOS is an authority as defined in UGC regulations, constituted by Head of the Organization for each of the departments separately. They are responsible for curriculum design and updation in respect of all the programs offered by a department.

Backlog Course: A course is considered to be a backlog course, if the student has obtained a failure grade (F) in that course.

Basic Sciences: The courses offered in the areas of Mathematics, Physics, Chemistry etc., are considered to be foundational in nature.

Betterment: Betterment is a way that contributes towards improvement of the students' grade in any course(s). It can be done by either (a) re-appearing or (b) re-registering for the course.

Commission: Means University Grants Commission (UGC), New Delhi.

Choice Based Credit System: The credit based semester system is one which provides flexibility in designing curriculum and assigning credits based on the course content and hours of teaching along with provision of choice for the student in the course selection.

Certificate Course: It is a course that makes a student to have hands-on expertise and skills required for holistic development in a specific area/field.

Compulsory course: Course required to be undertaken for the award of the degree as per the program.

Continuous Internal Examination: It is an examination conducted towards sessional assessment.

Core: The courses that are essential constituents of each engineering discipline are categorized as professional core courses for that discipline.

Course: A course is offered by a department for learning in a particular semester.

Course Outcomes: The essential skills that need to be acquired by every student through a course.

Credit: A credit is a unit that gives weight to the value, level or time requirements of an academic course. The number of 'Contact Hours' in a week of a particular course determines its credit value. One credit is equivalent to one lecture/tutorial hour per week.

Credit point: It is the product of grade point and number of credits for a course.

Cumulative Grade Point Average (CGPA): It is a measure of cumulative performance of a student over all the completed semesters. The CGPA is the ratio of total credit points secured by a student in various courses in all semesters and the sum of the total credits of all courses in all the semesters. It is expressed up to two decimal places.

Curriculum: Curriculum incorporates the planned interaction of students with instructional content, materials, resources, and processes for evaluating the attainment of Program Educational Objectives.

Department: An academic entity that conducts relevant curricular and co-curricular activities, involving both teaching and non-teaching staff, and other resources in the process of study for a degree.

Detention in a Course: Student who does not obtain minimum prescribed attendance in a course shall be detained in that particular course.

Dropping from Semester: Student who doesn't want to register for any semester can apply in writing in prescribed format before the commencement of that semester.

Elective Course: A course that can be chosen from a set of courses. An elective can be Professional Elective and / or Open Elective.

Evaluation: Evaluation is the process of judging the academic performance of the student in her/his courses. It is done through a combination of continuous internal assessment and semester end examinations.

Experiential Engineering Education (ExEEd): Engineering entrepreneurship requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Our students require sufficient skills to innovate in existing companies or create their own.

Grade: It is an index of the performance of the students in a said course. Grades are indicated by alphabets.

Grade Point: It is a numerical weight allotted to each letter grade on a 10 - point scale.

Honours: An Honours degree typically refers to a higher level of academic achievement at an undergraduate level.

Institute: Means Institute of Aeronautical Engineering, Hyderabad unless indicated otherwise by the context.

Massive Open Online Courses (MOOC): MOOC courses inculcate the habit of self-learning. MOOC courses would be additional choices in all the elective group courses.

Minor: Minor are coherent sequences of courses which may be taken in addition to the courses required for the B.Tech degree.

Pre-requisite: A specific course, the knowledge of which is required to complete before student register another course at the next grade level.

Professional Elective: It indicates a course that is discipline centric. An appropriate choice of minimum number of such electives as specified in the program will lead to a degree with specialization.

Program: Means, UG degree program: Bachelor of Technology (B.Tech); PG degree program: Master of Technology (M.Tech) / Master of Business Administration (MBA).

Program Educational Objectives: The broad career, professional and personal goals that every student will achieve through a strategic and sequential action plan.

Project work: It is a design or research-based work to be taken up by a student during his/her final year to achieve a particular aim. It is a credit based course and is to be planned carefully by the student.

Re-Appearing: A student can reappear only in the semester end examination for theory component of a course to the regulations contained herein.

Registration: Process of enrolling into a set of courses in a semester of a program.

Regulations: The regulations, common to all B.Tech programs offered by Institute, are designated as "BT23" and are binding on all the stakeholders.

Semester: It is a period of study consisting of 16 weeks of academic work equivalent to normally minimum of 90 working days. Odd semester commences usually in July and even semester in December of every year.

Semester End Examinations: It is an examination conducted for all courses offered in a semester at the end of the semester.

S/he: Means "she" and "he" both.

Student Outcomes: The essential skill sets that need to be acquired by every student during her/his program of study. These skill sets are in the areas of employability, entrepreneurial, social and behavioral.

University: Means Jawaharlal Nehru Technological University Hyderabad (JNTUH), Hyderabad, is an affiliating University.

Withdraw from a Course: Withdrawing from a course means that a student can drop from a course within the first two weeks of odd or even semester (deadlines are different for summer sessions). However, s/he can choose a substitute course in place of it, by exercising the option within 5 working days from the date of withdrawal.

PREFACE

Dear Students,

The focus at IARE is to deliver value-based education with academically well qualified faculty and infrastructure. It is a matter of pride that IARE continues to be the preferred destination for students to pursue an engineering degree.

In the year 2015, IARE was granted academic autonomy status by University Grants Commission, New Delhi under Jawaharlal Nehru Technology University Hyderabad. From then onwards, our prime focus is on developing and delivering a curriculum which caters to the needs of various stakeholders. The curriculum has unique features enabling students to develop critical thinking, solve problems, analyze socially relevant issues, etc. The academic cycle designed on the basis of Outcome Based Education (OBE) strongly emphasizes continuous improvement and this has made our curriculum responsive to current requirements.

The curriculum at IARE has been developed by experts from academia and industry and it has unique features to enhance problem solving skills apart from academic enrichment. The curriculum of B.Tech program has been thoroughly revised as per AICTE / UGC / JNTUH guidelines and have incorporated unique features such as competency training / coding, industry driven elective, internship and many more. The curriculum is designed in a way so as to impart engineering education in a holistic approach towards Excellence.

I hope you will have a fruitful stay at IARE.

Dr. L V Narasimha Prasad
Principal

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

ACADEMIC REGULATIONS – BT25

B.Tech. Regular Four-Year Degree Program
(for the batches admitted from the academic year 2025 - 2026)
&
B.Tech. (Lateral Entry Scheme)
(for the batches admitted from the academic year 2026 - 2027)

For pursuing Four-year undergraduate Bachelor of Technology (B.Tech) degree program of study in engineering offered by Institute of Aeronautical Engineering under Autonomous status.

A student after securing admission shall complete the B.Tech. program in a minimum period of four academic years and a maximum period of eight academic years starting from the date of commencement of first semester, failing which student shall forfeit seat in B.Tech. course. Each student has to secure a minimum of 160 credits out of 164 credits for successful completion of the undergraduate program and award of the B.Tech. degree. Additional 20/18 credits can be acquired for the degree of B.Tech with **Honours or Minor in Engineering**. Separate certificate will be issued in addition to major degree program mentioning that the student has cleared Honours / Minor specialization in respective courses.

1. CHOICE BASED CREDIT SYSTEM

The credit-based semester system provides flexibility in designing program curriculum and assigning credits based on the course content and hours of teaching. The Choice Based Credit System (CBCS) provides a 'cafeteria' type approach in which the students can take courses of their choice, learn at their own pace, undergo additional courses and acquire more than the required credits, and adopt an interdisciplinary approach to learning.

A course defines learning objectives and learning outcomes and comprises lectures / tutorials / laboratory work / field based research work / capstone project / seminars / internship / assignments / MOOCs / alternative assessment tools / presentations / self-study etc., or a combination of some of these. Under the CBCS, the requirement for awarding a degree is prescribed in terms of number of credits to be completed by the students.

2. MEDIUM OF INSTRUCTION

The medium of instruction shall be **English** for all courses, examinations, seminar presentations and project work. The program curriculum will comprise courses of study as given in course structure, in accordance with the prescribed syllabi.

3. PROGRAMS OFFERED

Presently, the institute is offering Bachelor of Technology (B.Tech) degree programs in Nine disciplines. The various programs and their two-letter unique codes are given in Table 1.

Table 1: B.Tech programs offered

S.No	Name of the Program	Title	Code
1	Aeronautical Engineering	AE	07
2	Computer Science and Engineering	CS	05
3	Computer Science and Engineering (AI&ML)	CA	34
4	Computer Science and Engineering (Data Science)	CD	35
5	Information Technology	IT	06
6	Electronics and Communication Engineering	EC	04
7	Electrical and Electronics Engineering	EE	02

8	Mechanical Engineering	ME	03
9	Civil Engineering	CE	01

4. SEMESTER STRUCTURE

The undergraduate program is of four academic years and there shall be two semesters in each academic year. There shall be a minimum of 15 weeks of instruction weeks, excluding the mid-term and semester-end exams. Around 15 instruction hours, 30 instruction hours and 45 hours of learning need to be followed per one credit of theory course, practical course and project/field-based learning respectively. All second Saturday's are observed as holiday.

Readmitted students and those admitted on transfer from JNTUH-affiliated institutes, universities, or other institutions are required to pursue the prescribed courses and earn credits on par with regular students, as prescribed by the respective Board of Studies.

5 REGISTRATION / DROPPING / WITHDRAWAL

The academic calendar includes important academic activities to assist the students and the faculty. These includes commencement of class work, continuous internal examinations, preparation holidays and semester end examinations. This enables the students to be well prepared and take full advantage of the flexibility provided by the credit system.

5.1. Each student has to compulsorily register for course work at the beginning of each semester as per the schedule mentioned in the academic calendar. It is compulsory for the student to register for courses in time. The registration will be organized departmentally under the supervision of the Head of the department.

5.2. In ABSENTIA, registration will not be permitted under any circumstances.

5.3. At the time of registration, students should have cleared all the dues of Institute and Hostel for the previous semesters, paid the prescribed fees for the current semester and not been debarred from the institute for a specified period on disciplinary or any other ground.

5.4. In the first two semesters, the prescribed course load per semester is fixed and is mandated to register all the courses. Withdrawal / Dropping of courses in the first and second semester is not allowed.

5.5. In all semesters, the average load is 20 credits / semester, with its minimum and maximum limits being set at 16 and 24 credits. This flexibility enables students (**from IV semester onwards**) to cope with the course work considering the academic strength and capability of student.

5.6. Dropping of Courses:

Within one week after the last date of first continuous internal examination, the student may in consultation with his / her faculty mentor / adviser, drop one or more courses without prejudice to the minimum number of credits as specified in clause 5.5. The dropped courses are not recorded in the memorandum of grades. Student must complete the dropped course(s) by registering in the forthcoming semester in order to earn the required credits.

5.7. Withdrawal from Courses:

A student is permitted to withdraw from a course before commencement first continuous internal examination. Such withdrawals will be permitted without prejudice to the minimum number of credits as specified in clause 5.5. A student cannot withdraw a course more than once and withdrawal of reregistered courses is not permitted.

6. CREDIT SYSTEM

The B.Tech program shall consist of a number of courses and each course shall be assigned with credits. The curriculum shall comprise Foundation Courses (FC), Professional Core (PC), Professional Electives (PE), Open Electives (OE), Laboratory Courses, Skill Development Courses, Other Courses and Mandatory Courses (MC) / Value Added Courses (VAC).

Depending on the complexity and volume of the course, the number of contact periods per week will be assigned. Each theory and laboratory course carries credits based on the number of hours/weeks.

- Contact classes (Theory): 1 credit per lecture hour per week, 1 credit per tutorial hour per week.
- Laboratory hours (Practical): 1 credit for 2 practical hours per week.
- Project work: 1 credit for 4 hours in a semester for Project/Mini-Project session per week.
- Skill Development Courses: 1 credit for 2 hours
- Mandatory courses / Value added courses : 1 credit is awarded.

Category wise distribution of Credits is shown in Table 2.

Table 2: Category Wise Distribution of Credits

S.No	Category	Number of courses	Credits per course(s)	Total Credits	
1	BS - Basic Science (1 - 4 credits, 4 - 3 Credits)	5	4/3	16	18
	BS - Basic Science Laboratories	2	1	2	
2	ES - Engineering Science	3	3	9	20
	ES - Engineering Science Laboratories	2	1	2	
	MEC Courses	2	3/1	4	
	CSC Courses	3	3/1	5	
3	HS - Humanities and Social Sciences	1	3	3	6
	HS - Humanities and Social Sciences Laboratories	3	1	3	
4	Professional Core - Theory	16	3	-	48
	Professional Core - Laboratories	15	1	-	15
5	Professional Electives	6	3		18
6	Open Electives	3	2		6
7	Project Work	1	14		14
8	Skill Development Courses	4	1		4
9	Industry Oriented Mini Project/ Summer Internship	1	2		2
10	Field-Based Project / Internship	1	2		2
11	Mandatory Courses / Value Added Course	3	1		3
12	Innovation and Entrepreneurship	1	2		2
13	Business Economics and Financial Analysis	1	3		3
14	Fundamentals of Management	1	3	-	3
Total			--	-	**164**

Major benefits of adopting the credit system are listed below:

- Quantification and uniformity in the listing of courses for all programs at institute, like program core, electives and project work.
- Ease of allocation of courses under different heads by using their credits to meet national /international practices in technical education.
- Convenience to specify the minimum / maximum limits of course load and its average per semester in the form of credits to be earned by a student.
- Flexibility in program duration for students by enabling them to pace their course load within minimum/maximum limits based on their preparation and capabilities.
- Wider choice of courses available from any department of the same institute or even from other similar institute, either for credit or for audit.
- Improved facility for students to optimize their learning by availing of transfer of credits earned by them from one College to another.

7. CURRICULAR COMPONENTS

Courses in a curriculum may be of three kinds: **Foundation courses, Core courses, Elective courses, Project core, Other core courses, Skill development courses and Value added courses.**

Foundation course:

Foundation courses include the basic science and engineering science course based upon the content leads to enhancement of skill and knowledge as well as value based and are aimed at man making education.

Professional core (PC):

There may be a professional core course in every semester. These courses are to be compulsorily studied by a student as a core requirement to complete the requirement of a program in the said discipline of study.

Professional electives (PE) / Open electives (OE):

Electives provide breadth of experience in respective branch and application areas. The program elective(s) is a course which can be chosen from a pool of courses. An elective may be professional elective, is a discipline centric focusing on those courses which add generic proficiency to the students or may be Open elective course, chosen from unrelated disciplines.

There are six professional elective tracks (PE-1 to PE-VI). Students can choose not more than two courses from each track. Overall, students can opt for six professional electives which suit their capstone project in consultation with the faculty advisor/mentor. Nevertheless, one course from each of the three open electives has to be selected. A student may also opt for more elective courses in his/her area of interest.

Students have the flexibility to choose from the list of professional electives offered by the Institute or opt to register for the equivalent MOOCS courses as listed from time to time by the institute.

It may be:

- Supportive to the discipline of study
- Providing an expanded scope
- Enabling an exposure to some other discipline / domain
- Nurturing student’s proficiency / skill.

Provision for Early Registration of MOOCs:

- For a professional elective in a semester, students are allowed to register for an equivalent MOOCs course listed from time to time.
- For example, a Professional Elective of VI semester shall be allowed to register under MOOCs platform in V semester.
- The credits earned in one semester in advance can be submitted in the subsequent semester for the assessment.
- The student who has registered in advance in an equivalent MOOCs course and fails to secure any pass grade in the MOOCs course, can register for the regular course offered in the following semester of their course catalogue.

Note: MOOCS courses are allowed only for professional electives.

Skill development courses are structured training courses which help students to acquire **practical abilities, competencies, and soft skills** which are essential for academic, professional, and personal growth.

Value Added Course (VAC):

Value-added courses that focus on professional values, traditional knowledge, and the sensitization of societal issues are short, skill-oriented programs designed to help students grow ethically, culturally, and socially beyond technical skills. The evaluation of value added courses shall be similar to theory courses. However, the scheduling of these CIA and SEE may not be combined with regular SEE examinations.

Semester wise course break-up

Following are the **TWO** models of course catalogue out of which any student shall choose or will be allotted with one model based on their academic performance.

i. Full Semester Internship (FSI) Model and
ii. Non Full Semester Internship (NFSI) Model

A student with **no current arrears** upto IV semester shall be eligible to opt for FSI. Students can opt full semester internship (FSI) in VIII semester. In Non-FSI Model, all the selected students shall carry out the course work and capstone project as specified in the course catalogue.

8. EVALUATION METHODOLOGY

Total marks for each course shall be based on Continuous Internal Assessment (CIA) and Semester End Examination (SEE). There shall have a uniform pattern of 40:60 for CIA and SEE of both theory and practical courses. The institute shall conduct multiple CIA for theory courses. All the performances of a student shall be considered for CIA marks distribution is shown in Table 3.

Table 3: Outline for Continuous Internal Assessments (CIA-1 and CIA-2) and SEE:

Activities	CIA-1	CIA-2	SEE	Total Marks
Continuous Internal Examination (CIE)	20 marks	20 marks		20 marks
Objective / Quiz	10 marks	10 marks		10 marks
Assignment	5 marks	5 marks		5 marks
Viva-Voce/PPT/Poster Presentation/ Case Study	5 marks	5 marks		5 marks
Semester End Examination (SEE)			60 marks	60 marks
Total	--	--	100 marks	

8.1 Continuous Internal Assessments (CIA-1 and CIA-2)

Assessment is an ongoing process that begins with establishing clear and measurable expected outcomes of student learning, provides students with sufficient opportunities to achieve those outcomes, and concludes with gathering and interpreting evidence to determine how well students' learning matches expectations.

The first component CIA-1 of assessment is for 20 marks. This assessment and score process should be completed after completing of first 50% of syllabus ($2^{1/2}$) of the course/s and within 45 working days of semester program.

The second component CIA-2 of assessment is for 20 marks, this assessment and score process should be completed after completing of remaining 50% of syllabus ($2^{1/2}$) of the course/s and within 45 working days of semester program.

In case of a student who has failed to attend the CIA1 or CIA2 on a scheduled date, shall be deemed that the student has dropped the examination. However, in case a student could not take the test on scheduled date due to genuine reasons, may appeal to the HOD / Principal. The HOD / Principal in consultation with the class in-charge shall decide about the genuineness of the case and decide to conduct Make-Up examination to such candidate on the date fixed by the Examinations Control Office but before commencement of the concerned semester end examinations.

The performance of a student in every course including value added courses, skill development courses, laboratory courses, capstone project work will be evaluated for 100 marks each, with 40 marks allotted for CIA and 60 marks for SEE, irrespective of the credits allocated.

8.2 Semester End Examination (SEE)

The semester end examinations, for theory courses, will be conducted for 60 marks consisting of two parts viz. i) Part- A for 10 marks and ii) Part - B for 50 marks.

Part-A is compulsory, consists of five short answer questions; each question carries two marks.

Part-B consists of five questions carrying 10 marks each. There shall be two questions asked in the question paper from each module with either-or choice and the student should answer either of the two questions. The student shall answer one question from each of five modules.

The duration of SEE is 3 hours.

8.3 Passing Criteria:

To maintain high standards in all aspects of examinations at the institute, the institute shall follow the standards of passing at CIA and SEE for each course.

a) A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course, if the student secures not less than 35% (21 marks out of 60 marks) in the semester end examinations, and a minimum of 40% (40 marks out of 100 marks) in the sum total of the Continuous Internal Assessment and Semester End Examination taken together; in terms of letter grades, this implies securing 'C' grade or above in that course.

b) A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to Engineering Design Project / Engineering Development Project / Summer Internship, if the student secures not less than 40% marks (i.e. 40 out of 100 allotted marks) in each of them. The student is deemed to have failed, if he/she (i) does not submit a report on Engineering Design Project / Engineering Development Project / Summer Internship or (ii) not make a presentation of the same before the evaluation committee as per schedule, or (iii) secures less than 40% marks in Field-Based Research Project / Industry Oriented Mini Project / Internship evaluations.

c) A student eligible to appear in the SEE for any course, is absent from it or failed (thereby failing to secure 'C' grade or above) may re-appear for that course in the supplementary examination as and when it is conducted. In such cases, internal marks (CIA) assessed earlier for that course will be carried over, and added to the marks obtained in the SEE supplementary examination. If the student secures sufficient marks for passing, 'C' grade or above shall be awarded as specified in clause 11.3.

8.4 Supplementary examinations

Supplementary examinations for the odd semester shall be conducted with the regular examinations of even semester and vice versa. In case of failure in any course, a student may be permitted to register for the same course when offered.

Advanced supplementary examinations in VIII semester courses may be conducted for those who failed in any course offered in VIII semester. It may enable the students to receive their B.Tech provisional certificate at an early date.

There shall be no supplementary examination in the successive semester. The students who could not secure any pass grade in advance supplementary examinations have to wait for regular series examination of next batch to write their backlog examination.

8.5 Laboratory Course

Evaluation methodology of laboratory course (CIA)

Each laboratory courses there shall be a CIA during the semester for 40 marks and 60 marks for SEE. The 40 marks for internal evaluation marks are awarded as follows:

1. A write-up on day-to-day experiment in the laboratory (in terms of aim, components / procedure, expected outcome) which shall be evaluated for **20 marks.**
2. Internal practical examination conducted by the laboratory teacher concerned shall be evaluated for **10 marks.**
3. The remaining **10 marks** will be awarded for the project report submission and evaluation.

Evaluation methodology of laboratory course (SEE)

The SEE shall be conducted by an external examiner along with the laboratory handling faculty. The external examiner shall be appointed from other institutions and will be selected from the panel by the Principal.

The SEE held for 3 hours. Total 60 marks are divided and allocated as shown below:

1. 10 marks for write-up
2. 10 for experiment / program
3. 10 for evaluation of results
4. 10 marks for viva-voce on concerned laboratory course.
5. 20 marks will be awarded for the solving Complex Tasks Descriptions (CTDs) / take-home project identified by department advisory committee which will be announced at the beginning of the semester.

8.6 Evaluation of Engineering Design Project and Engineering Development Project

Engineering Design and Development Projects is the high point of degree studies in engineering. The transition of students to industry is still not optimal, and there is a disparity between the needs of industry and the actual ability of academia to meet these needs. Each project is carried out under the supervision of academic faculty and where appropriate an industry partner.

The **Engineering Design Project and Engineering Development Project** shall be evaluated for 100 marks each. out of which 40 marks for CIA and 60 marks for SEE. The evaluation committee shall consist of a Head of the Department, Supervisor of the Project and a Senior Faculty Member of the department. Student shall have to earn 40% marks, i.e 40 marks out of 100 marks. The student is deemed to have failed, if he (i) does not submit a report on the project, or (ii) does not make a presentation of the same before the internal committee as per schedule, or (iii) secures less than 40% marks in this course.

8.7 Evaluation of Summer Internship

A Summer Internship shall be undertaken in collaboration with an industry relevant to the student's area of specialization. Students must register for the internship immediately after the completion of their V semester examinations and pursue it during the summer vacation. The internship, preferably at a reputed organization, must be documented in the form of a report and presented before a committee during the VII semester, prior to the semester-end examinations.

The internship shall be evaluated for 100 external marks, out of which 40 marks for CIA and 60 marks for SEE. The evaluation committee shall consist of an external examiner, the Head of the Department, supervisor, and a senior faculty member from the department.

8.8 Additional Mandatory Courses for lateral entry B.Tech students

In addition to the non-credit mandatory courses for regular B.Tech students, the lateral entry students shall take up the following three non-credit mandatory bridge courses (one in III semester, one in IV semester and one in V semester) as listed in Table 4. The student shall pass the following non-credit mandatory courses for the award of the degree and must clear these bridge courses before advancing to the VII semester of the program.

Table-4: Additional Mandatory Courses for lateral entry students

S.No	Additional mandatory courses for lateral entry students
1	Dip - Object Oriented Programming
2	Dip - Data Structures
3	Dip - Front-End Web Development

8.9 Innovation and Entrepreneurship

Innovation and entrepreneurship course offered in III semester and its requires strong technical skills in engineering design and computation with key business skills from marketing to business model generation. Students require sufficient skills to innovate in existing companies or create their own.

This course will be evaluated for a total of 100 marks consisting of 40 marks for CIA and 60 marks for SEE. Out of 40 marks of internal assessment, students has to submit Innovative Idea in a team of three / four members in the given format. The SEE for 60 marks shall be conducted internally, students has to present the Innovative Idea and it will be evaluated by internal course handling faculty with at least one faculty member as examiner from the industry, both nominated by the Principal.

8.10 Capstone Project

The capstone project shall be initiated at the beginning of the VIII semester and the duration is one semester. The student must present in consultation with his/her supervisor, the title, objective and plan of action of his/her to the departmental committee for approval within two weeks from the commencement of VIII semester. Only after obtaining the approval of the departmental committee, the student can start his/her capstone project.

Student has to submit report of capstone project at the end of VIII semester. It shall be evaluated for 100 marks. Out of which 40 marks and 60 marks are allocated for CIA and SEE respectively. Evaluation shall be done by a committee comprising the supervisor, Head of the department, and an external examiner nominated by the Principal.

The external examiner shall evaluate the capstone project for 60 marks and the internal committee shall evaluate it for 40 marks. The departmental committee consisting of Head of the department, supervisor and a senior faculty member shall evaluate for 20 marks and supervisor shall evaluate for 20 marks.

The topics for the capstone project shall be different from the topics of Engineering design project / Engineering development project / Summer Internship. The student is deemed to have failed, if he (i) does not submit a capstone project, or (ii) does not make a presentation of the same before the external examiner as per schedule, or (iii) secures less than 40% marks in the sum total of the CIE and SEE taken together.

The external examiner will be selected for conducting viva-voce examination from the list of experts selected by the Principal of the institute.

This gives students a platform to experience a research driven career in engineering, while developing a device / systems and publishing in reputed SCI / SCOPUS indexed journals and/or filing an **Intellectual Property** (IPR-Patent/Copyright) to aid communities around the world. Students should work individually as per the guidelines issued by head of the department concerned. The benefits to students of this mode of learning include increased engagement, fostering of critical thinking and greater independence.

A student who has failed, may re-appear once for the above evaluation, when it is scheduled again; if student fails in such 'one re-appearance' evaluation also, he/she has to appear for the same in the next subsequent year, as and when it is scheduled.

A minimum of 50% of maximum marks shall be obtained to earn the corresponding credits.

8.11 Skill Development Courses

There are four skill development courses included in the curriculum in III, IV, V and VI semesters. Each skill development course carries one credit. The evaluation pattern will be the same as that for a laboratory course including internal and external assessments.

The objective of skill courses is to develop the cognitive skills as well as the psycho-motor skills.

8.12 Full Semester Internship (FSI)

FSI is a full semester internship program carry 14 credits. The FSI shall be opted in VIII semester. During the FSI, student has to spend one full semester in an identified industry / firm / R&D organization or another academic institution/University where sufficient facilities exist to carry out the project work.

The selection procedure is:

- Choice of the students.
- CGPA (> 7.5) upto IV semester having no credit arrears.
- Competency Mapping / Allotment.

It is recommended that the FSI Project work leads to a research publication in a reputed Journal / Conference or the filing of patent / design with the patent office, or, the start-up initiative with a sustainable and viable business model accepted by the incubation center of the institute together with the formal registration of the startup.

9. ATTENDANCE REQUIREMENTS AND DETENTION POLICY

9.1 A student shall be eligible to appear for the semester end examinations, if the student acquires a minimum of 75% of attendance in aggregate of all the courses for that semester.

9.2 Shortage of attendance in aggregate upto 10% (65% and above, and below 75%) in each semester may be condoned by the college academic committee on genuine and valid grounds, based on the student's representation with supporting evidence.

9.3 A stipulated fee shall be payable for condoning of shortage of attendance.

9.4 Shortage of attendance below 65% in aggregate shall in NO case be condoned.

9.5 Students whose shortage of attendance is not condoned in any semester are not eligible to take their semester end examinations of that semester. They get detained and their registration for that semester shall stand cancelled, including all academic credentials (internal marks etc.) of that semester. They will not be promoted to the next semester. They may seek re-registration for that semester in the next academic year.

9.6 A student fulfilling the attendance requirement in the present semester shall not be eligible for readmission into the same semester / class.

9.7 A student detained in a semester due to shortage of attendance may be re-admitted in the same semester in the next academic year for fulfillment of academic requirements. The academic regulations under which a student has been re-admitted shall be applicable. Further, no grade allotments or SGPA/ CGPA calculations will be done for the entire semester in which the student has been detained.

9.8 A student detained due to lack of credits, shall be promoted to the next academic year only after acquiring the required number of academic credits. The academic regulations under which the student has been readmitted shall be applicable to him.

10. CONDUCT OF SEMESTER END EXAMINATIONS AND EVALUATION

10.1 Semester end examination shall be conducted by the Controller of Examinations (COE) by inviting question papers from the external examiners.

10.2 The Controller of Examinations (COE) shall invite external examiners to evaluate all semester end examination answer scripts on the scheduled dates. Similarly, practical laboratory examinations shall be conducted in the presence of external examiners to ensure transparency and fair evaluation.

10.3 Examinations control office shall consolidate the marks awarded by examiner/s and award the grades.

11. LETTER GRADES AND GRADE POINTS

11.1 ABSOLUTE Grading system is followed for awarding the grade to each course.

11.2 Performance of students in each course, Theory, Laboratory, Industry-Oriented Mini Project/ Internship/ Skill development course and Project Work are expressed in terms of marks as well as in Letter Grades based on absolute grading system.

11.3 To measure the performance of a student, a 10-point grading system is followed. The mapping between the percentage of marks secured and the corresponding letter grade is as shown in the below Table 5.

Table-5: Grade Points Scale (Absolute Grading)

Range of % of Marks Secured in a Course	Letter Grade	Grade Point
Greater than or equal to 90	O (Outstanding)	10
80 and less than 90	A+ (Excellent)	9
70 and less than 80	A (Very Good)	8
60 and less than 70	B+ (Good)	7
50 and less than 60	B (Average)	6
40 and less than 50	C (Pass)	5
Below 40	F (Fail)	0
Absent	AB (Absent)	0

11.3 A student is deemed to have passed and acquired to correspondent credits in particular course if s/he obtains any one of the following grades: “O”, “A+”, “A”, “B+”, “B”, “C”.

11.4 A student who has obtained an ‘F’ grade in any course shall be deemed to have **‘failed’** and is required to reappear for a supplementary exam as and when conducted. In such cases, internal marks in those courses will remain the same as those obtained earlier.

11.5 A student who has not appeared for an examination in any course, ‘Ab’ grade will be allocated in that course, and he/she is deemed to have ‘Failed’. Such student will be required to reappear for supplementary/make-up exam as and when conducted. The internal marks in those courses will remain the same as those obtained earlier.

11.6 The students earn a Grade Point (GP) in each course, on the basis of letter grade secured in that course. Every student who passes a course will receive grade point GP ≥ 5 (‘C’ grade or above).

11.7 At the end of each semester, the institute issues grade sheet indicating the SGPA and CGPA of the student. However, grade sheet will not be issued to the student if s/he has any outstanding dues.

11.8 If a student earns more than 160 credits, only the courses corresponding to the best 160 credits shall be considered for the computation of CGPA of B.Tech. degree.

12. COMPUTATION OF SGPA AND CGPA

The UGC recommends to compute the Semester Grade Point Average (SGPA) and Cumulative Grade Point Average (CGPA). The credit points earned by a student are used for calculating the Semester Grade Point Average (SGPA) and the Cumulative Grade Point Average (CGPA), both of which are important performance indices of the student.

The Semester Grade Point Average (SGPA) is calculated only when all the courses offered in a semester are cleared by a student. It is calculated by dividing the sum of credit points secured from all courses registered in a semester, by the total number of credits registered during that semester. SGPA is rounded off to two decimal places. SGPA for each semester is thus computed as

$$\textbf{SGPA} = \{ \sum_{i=1}^{N} C_i\ G_i \} / \{ \sum_{i=1}^{N} C_i \}$$

where ‘i’ is the course indicator index (considering all courses in a semester), ‘N’ is the no. of courses registered for the semester (as listed under the course structure of the branch), C_i is the no. of credits allotted to the ith course, and G୧ represents the grade points corresponding to the letter grade awarded for that ith course.

The Cumulative Grade Point Average (CGPA) is a measure of the overall cumulative performance of a student in all semesters considered for registration. The CGPA is the ratio of the total credit points secured by a student for the courses correspond to best 160 credits out of all registered courses in all semesters, and the total number of credits correspond to those selected courses. CGPA is rounded off to two decimal places. CGPA is thus computed at the end of each semester, from the I year II semester onwards, as per the formula.

$$\mathbf{CGPA = \{ \textstyle\sum_{j=1}^{M} C_j\ G_j \} / \{ \sum_{j=1}^{M} C_j \}}$$

where 'M' is the total no. of courses corresponding to the best 160 credits from the courses registered in all eight semesters, 'j' is the course indicator index (takes into account all courses from 1 to 8 semesters), C_j is the no. of credits allotted to the j^{th} course, and G_j represents the grade points (GP) corresponding to the letter grade awarded for that j^{th} course.

The SGPA and CGPA shall be rounded off to 2 decimal points and reported in the transcripts.

13.0 ILLUSTRATION OF COMPUTATION OF SGPA AND CGPA

13.1 Illustration for SGPA

Course Name	Course Credits	Grade letter	Grade point	Credit Point (Credit x Grade)
Course 1	4	A	8	4 x 8 = 32
Course 2	3	O	10	3 x 10 = 30
Course 3	3	C	5	3 x 5 = 15
Course 4	3	B	6	3 x 6 = 18
Course 5	3	A	8	3 x 8 = 24
Course 6	2	A+	9	2 x 9 = 18
Course 7	1	C	5	1 x 5 = 15
Course 8	1	O	10	1 x 10 = 10
	20			**152**

Thus, SGPA = 152 / 20= 7.6

13.2. The CGPA of the entire B.Tech. program shall be calculated considering the best 160 credits earned by the student.

13.3 For merit ranking or comparison purposes or any other listing, only the 'rounded off' values of the CGPAs will be used.

13.4 SGPA of a semester will be mentioned in the semester Memorandum of Grades if all courses of that semester are passed in first attempt. Otherwise, the SGPA shall be mentioned only on the Memorandum of Grades in which sitting he passed his last exam in that semester.

14. REVALUATION

If the examinee is not satisfied with the marks awarded, s/he may apply for revaluation of answer booklets in prescribed format online within three (3) working days from the date of declaration of result of the examination or issue of the statement of marks, whichever is earlier. The revaluation facility shall be for theory papers only. The revaluation of answer booklets shall not be permitted in respect of the marks awarded to the scripts of practical examination / project work (including theory part) and in viva voce / oral / comprehensive examinations.

15. PROMOTION POLICIES

The following academic requirements have to be satisfied in addition to the attendance requirements mentioned in item no. 9.

S.No	Promotion	Conditions to be Fulfilled
1	First year first semester to first year second semester	Regular course of study of first year first semester and fulfilment of attendance requirement.
2	First year second semester to Second year first semester	(i) Regular course of study of first year second semester and fulfilment of attendance requirement (ii) Must have secured at least 25% of the total credits up to first year second semester from all the relevant regular and supplementary examinations, whether the student takes those examinations or not.
3	Second year first semester to Second year second semester	Regular course of study of second year first semester and fulfilment of attendance requirement.
4	Second year second semester to Third year first semester	(i) Regular course of study of second year second semester and fulfilment of attendance requirement. (ii) Must have secured at least 25% of the total credits up to second year second semester from all the relevant regular and supplementary examinations, whether the student takes those examinations or not.
5	Third year first semester to Third year second semester	Regular course of study of third year first semester and fulfilment of attendance requirement.
6	Third year second semester to Fourth year first semester	Regular course of study of third year second semester and fulfilment of attendance requirement
7	Fourth year first semester to Fourth year second semester	Regular course of study of fourth year first semester and fulfilment of attendance requirement

16. CREDIT EXEMPTION

A student (i) shall register for all courses covering 164 credits as specified and listed in the course catalogue and (ii) earn 160 or more credits to successfully complete the undergraduate Program.

- Best 160 credits shall be considered for CGPA computation. The student can avail exemption of courses ***totalling upto 4 credits*** other than professional core courses, laboratory courses, seminars, capstone project, engineering design project / engineering development project / summer internship, for optional drop out from these 164 credits registered.
- The Semester Grade Point Average (SGPA) of each semester shall be mentioned at the bottom of the grade card, when all the subjects in that semester have been passed by the student.
- Credits earned by the student in either a Minor or Honors program cannot be counted towards the required 160 credits for the award of the B.Tech. degree.

17. AWARD OF DEGREE

17.1 A student who registers for all the specified courses as listed in the course catalogue and secures the required number of 160 credits within 8 academic years from the date of commencement of the first academic year, shall be declared to have 'qualified' for the award of B.Tech. degree in the branch of Engineering selected at the time of admission.

17.2 A student who qualifies for the award of the degree as listed in item 17.1 shall be placed in the following classes.

17.3 A student with final CGPA (at the end of the undergraduate program) ≥ 7.5, and fulfilling the following conditions - shall be placed in '**first class with distinction**'. However,

a. Should have passed all the courses in '**first appearance'** within the first 4 academic years (or 8 sequential semesters) from the date of commencement of first semester.

b. Should not have been detained or prevented from writing the semester end examinations in any semester due to shortage of attendance or any other reason.

c. A student not fulfilling any of the above conditions with final CGPA ≥ 7.5 shall be placed in **‘first class’.**

17.4 Students with final CGPA (at the end of the undergraduate program) ≥ 6.5 but < 7.5 shall be placed in **‘first class’.**

17.5 Students with final CGPA (at the end of the undergraduate program) ≥ 5.5 but < 6.5, shall be placed in ‘**second class**’.

17.5 All other students who qualify for the award of the degree (as per item 17.1), with final CGPA (at the end of the B.Tech program) ≥ 5.00 but < 5.5, shall be placed in **‘pass class’**.

17.6 A student with final CGPA (at the end of the B.Tech program) <5.0 will not be eligible for the award of the degree.

17.7 Students fulfilling the conditions listed under item 17.3 alone will be eligible for award of ‘**Gold Medal**’.

17.8 If more than one student secures the same highest CGPA, then the following tie resolution criteria, in the same order of preference shall be followed for selecting the Gold Medal winner, until the tie is resolved: 1) more number of times secured highest SGPAs, ii) more number of O and A+ grades in that order and iii) highest SGPA in the order of first semester to eight semester.

17.9 Grace Marks

Grace marks shall be given to those students who complete the course work of four year B.Tech. degree, not secured pass grade in not more than three courses and adding a specified grace marks enables the student to pass the course(s) as well as gets eligibility to receive the provisional degree certificate.

Grace marks for students admitted under the R25 Academic Regulations should not exceed 0.15% of the total maximum marks in all eight semesters (excluding the marks allocated for value added courses and skill development courses). The grace marks shall only be added if a student fails in a maximum of three courses and adding the above grace marks make the student eligible to receive the provisional degree certificate.

All the candidates who register for the SEE will be issued a memorandum of grades sheet by the institute. Apart from the semester wise memorandum of grades sheet, the institute will issue the provisional certificate and consolidated grades memorandum certificate to the fulfilment of all the academic requirements.

18. CONVERSION OF CGPA INTO EQUIVALENT PERCENTAGE OF MARKS

The following formula shall be used for the conversion of CGPA into equivalent marks, whenever it is necessary.

Percentage (%) of Marks = (Final CGPA – 0.5) x 10

The Table 06 shows the Percentage Equivalence of Grade Points (for a 10 – Point Scale).

Table 06: Percentage Equivalence of Grade Points (for a 10 – Point Scale)

Grade Point	Percentage of Marks / Class
5.5	50
6.0	55
6.5	60
7.0	65
7.5	70
8.0	75

19. B.TECH WITH HONOURS OR MINOR IN ENGINEERING

Students acquiring 160 credits or more are eligible to get B.Tech degree in Engineering. A student will be eligible to get B.Tech degree with Honours or Minor in Engineering, if s/he completes an additional 20/18 credits (3/4 credits per course). These could be acquired through MOOCs from SWAYAM / NPTEL only. The list for MOOCs will be a dynamic one, as new courses are added from time to time. Few essential skill sets required for employability are also identified year wise. Students interested in doing MOOC courses shall register the course title at their department office at the start of the semester against the courses that are announced by the department. Any expense incurred for the MOOC course / summer program should be met by the students.

Students having no credit arrears and a CGPA of 7.5 or above at the end of the fourth semester are eligible to register for B.Tech (Honours / Minor). After registering for the B.Tech (Honours / Minor) program, if a student fails in any course, s/he will not be eligible for B.Tech (Honours / Minor).

Honours Certificate for Vertical in his/her OWN Branch for Research orientation; Minor in any other branch for Improving Employability.

Honours will be reflected in the degree certificate as “B.Tech (Honours) in XYZ Engineering”. Similarly, Minor as “B.Tech in XYZ Engineering with Minor in ABC”.

19.1. B.Tech with Honours

The key objectives of offering B.Tech with Honors program are:

- To expand the domain knowledge of the students laterally and vertically.
- To increase the employability of undergraduate students with expanded knowledge in one of the core engineering disciplines.
- To provide an opportunity to students to pursue their higher studies in wider range of specializations.

Academic Regulations for B. Tech. Honours degree

1. The weekly instruction hours, internal and external evaluation and award of grades are on par with regular 4-Years B. Tech. program.
2. For B. Tech with Honors program, a student needs to earn additional 20 credits (over and above the required 160 credits for B.Tech degree). All these 20 credits required to be attained for B.Tech Honors degree credits are distributed from V semester to VII semester.
3. After registering for the Honors program, if a student is unable to pass all courses in first attempt and earn the required 20 credits, he/she shall not be awarded Honours degree. However, if the student earns all the required 160 credits or more of B.Tech., he/she will be awarded only B.Tech degree in the concerned branch.
4. There is no transfer of credits from courses of Honors program to regular B.Tech. degree course & vice versa.
5. These **20 credits** are to be earned from the additional courses offered by the host department in the institute or from closely related departments in the institute as well as from the MOOCS platform (NPTEL only).
6. The choice to opt/take the Honors program is purely on the choice of the students.
7. The student shall be given a choice of withdrawing all the courses registered and/or the credits earned for Honours program at any time; and in that case the student will be awarded only B.Tech. degree in the concerned branch on earning the required credits of 160.
8. The students of every branch can choose Honours program in their respective branches if they are eligible for the Honors program. A student who chooses an **Honors program is not eligible to choose a Minor program and vice-versa.**
9. A student can graduate with Honors if he/she fulfils the requirements for his/her regular B.Tech. program as well as fulfills the requirements for Honours program.
10. The institute shall maintain a record of students registered and pursuing their Honors programs branch-wise.

Eligibility conditions of the students for the B.Tech Honors degree

a) A student can opt for B.Tech. degree with Honors, if she/he passed all courses in first attempt in all the semesters till the results announced and maintaining **7.5 or more CGPA.**

b) If a student fails in any registered course of either B.Tech or Honours in any semester of four years program, he/she will not be eligible for obtaining Honors degree. He will be eligible for only

c) Prior approval of mentor and Head of the Department for the enrolment into Honors program, before commencement of V Semester, is mandatory.

d) If more than **30% of the students** in a branch fulfill the eligibility criteria (as stated above), the number of students given eligibility should be limited to 30%. The criteria to be followed for choosing 30% candidates in a branch may be the CGPA secured by the students till **III** semester.

e) Successful completion of **20 credits** earmarked for Honours program with at least 7.5 CGPA along with successful completion of 160 credits earmarked for regular B. Tech. Program with at least 7.5 CGPA and passing all courses in first attempt gives the eligibility for the award of B.Tech (Honors) degree.

f) For CGPA calculation of B.Tech. course, the **20 credits** of Honours program will not be considered.

Following are the details of such Honors which include some of the most interesting areas in the profession today:

S.No	Department	Honors scheme
1	Aeronautical Engineering	Aerospace Engineering / Space Science etc.
2	Computer Science and Engineering / Information Technology	Big data and Analytics / Cyber Physical Systems, Information Security / Cognitive Science / Artificial Intelligence/ Machine Learning / Data Science / Internet of Things (IoT) / Cyber Security etc.
3	Electronics and Communication Engineering	Digital Communication / Signal Processing / Communication Networks / VLSI Design / Embedded Systems etc.
4	Electrical and Electronics Engineering	Renewable Energy systems / Energy and Sustainability / IoT Applications in Green Energy Systems etc.
5	Mechanical Engineering	Industrial Automation and Robotics / Manufacturing Sciences and Computation Techniques etc.
6	Civil Engineering	Structural Engineering / Environmental Engineering etc.

19.2 B.Tech with Minor in Engineering

The key objectives of offering B.Tech with Minor program are:

- To expand the domain knowledge of the students in one of the other branches of engineering.
- To increase the employability of undergraduate students keeping in view of better opportunity in interdisciplinary areas of engineering & technology.
- To provide an opportunity to students to pursue their higher studies in the inter-disciplinary areas in addition to their own branch of study.
- To offer the knowledge in the areas which are identified as emerging technologies/thrust areas of Engineering.

Academic Regulations for B.Tech Degree with Minor programs

1. The weekly instruction hours, internal & external evaluation and award of grades are on par with regular 4-Years B. Tech. program.
2. For B. Tech. with Minor, a student needs to earn additional 18 credits (over and above the required 160 credits for B. Tech degree). The courses are offered from V semester to VII semester only, to obtain minor degree students required to obtain 18 credits.
3. After registering for the Minor program, if a student is unable to earn all the required 18 credits in a specified duration (twice the duration of the course), he/she shall not be awarded Minor degree. However, if the student earns all the required 160 credits of B.Tech, he/she will be awarded only

B. Tech degree in the concerned branch.

4. There is no transfer of credits from Minor program courses to regular B. Tech. degree course & vice versa.
5. These 18 credits are to be earned from the additional courses offered by the host department in the institute as well as from the MOOCs platform.
6. For the course selected under MOOCs platform (NPTEL) following guidelines may be followed:
   a) Prior to registration of MOOCs courses, formal approval of the courses, by the institute is essential, before the issue of approval considers the parameters like the institute / agency which is offering the course, syllabus, credits, duration of the program and mode of evaluation etc.
   b) Minimum credits for MOOCs course must be equal to or more than the credits specified in the Minor course structure provided by the institute.
   c) Only Pass-grade / marks or above shall be considered for inclusion of grades in minor grade memo.
   d) Any expenses incurred for the MOOCs courses are to be met by the students only.
7. The choice to opt / take a Minor program is purely on the choice of the students.
8. The student shall be given a choice of withdrawing all the courses registered and / or the credits earned for Minor program at any time; and in that case the student will be awarded only B. Tech. degree in the concerned branch on earning the required credits of 160.
9. The student can choose only one Minor program along with his / her basic engineering degree. A student who chooses an **Honors program is not eligible to choose a Minor program and vice-versa.**
10. The institute shall maintain a record of students registered and pursuing their Minor programs, minor program-wise and parent branch-wise.
11. The institute / department shall prepare the time-tables for each Minor course offered at their respective institutes without any overlap/clash with other courses of study in the respective semesters.

Eligibility conditions for the student to register for Minor course

a) A student can opt for B.Tech. degree with Minor program if she/he has no active backlogs till III semester at the time of entering into V semester.

b) Prior approval of mentor and Head of the Department for the enrolment into Minor program, before commencement of V Semester, is mandatory.

c) If more than 50% of the students in a branch fulfill the eligibility criteria (as stated above), the number of students given eligibility should be limited to 50%.

20.0 TEMPORARY BREAK OF STUDY FROM THE PROGRAM

20.1 A candidate is normally not permitted to take a break from the study. However, if a candidate intends to temporarily discontinue the program in the middle for valid reasons (such as accident or hospitalization due to prolonged ill health) and to rejoin the program in a later respective semester, s/he shall seek the approval from the Principal in advance. Such application shall be submitted before the last date for payment of examination fee of the semester and forwarded through the Head of the Department stating the reasons for such withdrawal together with supporting documents and endorsement of his / her parent / guardian.

20.2 The institute shall examine such an application and if it finds the case to be genuine, it may permit the student to temporarily withdraw from the program. Such permission is accorded only to those who do not have any outstanding dues / demand at the College / University level including tuition fees, any other fees, library materials etc.

20.3 The candidate has to rejoin the program after the break from the commencement of the respective semester as and when it is offered.

20.4 The total period for completion of the program reckoned from the commencement of the semester to which the candidate was first admitted shall not exceed the maximum period specified in clause 17. The maximum period includes the break period.

20.5 If any candidate is detained for any reason, the period of detention shall not be considered as 'Break of Study'.

21. TERMINATION FROM THE PROGRAM

The admission of a student to the program may be terminated and the student is asked to leave the institute in the following circumstances:

a. The student fails to satisfy the requirements of the program within the maximum period stipulated for that program.
b. A student shall not be permitted to study any semester more than three times during the entire program of study.
c. The student fails to satisfy the norms of discipline specified by the institute from time to time.

22. WITH-HOLDING OF RESULTS

If the student has not paid the fees to the institute at any stage, or has dues pending due to any reason whatsoever, or if any case of indiscipline is pending, the result of the student may be withheld, and the student will not be allowed to go into the next higher semester. The award or issue of the degree may also be withheld in such cases.

.

23. GRADUATION DAY

The institute shall have its own annual Graduation Day for the award of degrees to the students completing the prescribed academic requirements in each case, in consultation with the University and by following the provisions in the Statute. The college shall award prizes and medals to meritorious students and award them annually at the Graduation Day. This will greatly encourage the students to strive for excellence in their academic work.

24. DISCIPLINE

Every student is required to observe discipline and decorum both inside and outside the institute and are expected not to indulge in any activity which will tend to bring down the honour of the institute. If a student indulges in malpractice in any of the theory / practical examination, continuous assessment examinations, he/she shall be liable for punitive action as prescribed by the institute from time to time.

25. GRIEVANCE REDRESSAL COMMITTEE

The institute shall form a Grievance Redressal Committee for each course in each department with the Course Teacher and the HOD as the members. The committee shall solve all grievances related to the course under consideration.

26. MULTIPLE ENTRY MULTIPLE EXIT SCHEME (MEME)

26.1 Exit option after Second Year:

Students enrolled in the 4-Year B.Tech. program are permitted to exit the program after successful completion of the second year (B.Tech. IV Semester). The students who desire to exit after the IV year shall formally inform the exit plan one semester in advance i.e. at the commencement of II Semester itself. Such students need to fulfil the additional requirements as specified in Clause 27 described below.

Upon fulfilling the requirements like earning all the credits up to IV semester and successfully completing the additional requirements, the students will be awarded a 2-year undergraduate (UG) diploma in the concerned engineering branch.

27. ADDITIONAL REQUIREMENTS FOR DIPLOMA AWARD

To qualify for the diploma under the exit option, students must also complete 2 additional credits through one of the following institute-prescribed pathways.

Work-based Vocational Course:

Participation in a practical, hands-on vocational training program relevant to the engineering field, typically conducted during the summer term.

Internship / Apprenticeship:

Completion of a minimum 8-week internship or apprenticeship in their related field to gain practical industry exposure.

In addition, students must clear any associated course(s) or submit the internship/apprenticeship report as per the institute schedule and guidelines.

28. RE-ENTRY INTO THE B.TECH PROGRAM

Students who have exited the B.Tech. program with a 2 Year UG Diploma may apply for re-entry into the Fifth Semester of the B.Tech. program. Re-entry is subject to the following conditions.

- The student must surrender the awarded UG Diploma Certificate.
- Students who wish to rejoin in V semester must join the same B.Tech. program and same college from which the student exited. Before rejoining, students should check for continuation of the same branch at the college. If the specific branch is closed in that particular college, then student should consult the University for the possible alternative solutions.
- Re-registered students will be governed by the academic regulations in effect at the time of re-entry, regardless of the original regulations under which they were admitted.
- If a student opts to continue their studies without a gap after being awarded the diploma, they must register for the third-year courses before the commencement of classwork.

29. BREAK IN STUDY AND MAXIMUM DURATION

Students are allowed to take a break of up to four years after completion of IV Semester with prior University permission through the Principal of the college.

Re-entry after such a break is subject to the condition that the student completes all academic requirements within twice the duration of the program (i.e., within 8 years for a 4-year B.Tech. program).

30. TRANSITORY REGULATIONS TO THE STUDENTS RE-ADMITTED IN BT25 REGULATIONS:

30.1 Transitory regulations are applicable to the students detained due to shortage of attendance as well as detained due to the shortage of credits and seeks permission to re-join the B.Tech program, where BT25 regulations are in force.

30.2 A student detained due to shortage of attendance and re-admitted in BT25 regulations: Such students shall be permitted to join the same semester, but in BT25 Regulations.

30.3 A student detained due to shortage of credits and re-admitted in BT25 regulations: Such students shall be promoted to the next semester in BT25 regulations, only after acquiring the required number of credits as per the corresponding regulations of his/her previous semester.

30.4 A student who has failed in any course in a specific regulation has to pass those courses in the same regulations.

30.5 If a student is readmitted to BT25 Regulations and has any course with 80% of syllabus common with his/her previous regulations, that particular course in BT25 Regulations will be substituted by an equivalent course of BT23 regulations by the institute. All these details are summarized in a set of look-up Table; one set for each B. Tech. branch.

30.6 Look Up Table of equivalence courses

A lookup table will be provided for the benefit of students and Principals. This lookup table will include all the courses to be registered by students who have been re-admitted under the BT25 academic regulations from the BT23 academic regulations. Separate lookup tables will be provided for the following categories of students:

a. Students re-admitted into the II Semester of the R25 Regulations

b. Students re-admitted into the III Semester of the R25 Regulations
c. Students re-admitted into the IV Semester of the R25 Regulations,
d. Students re-admitted into the V Semester of the R25 Regulations
e. Students re-admitted into the VI Semester of the R25 Regulations
f. Students re-admitted into the VII Semester of the R25 Regulations
g. Students re-admitted into the VIII Semester of the R25 Regulations

For every B.Tech. branch there shall be separate set of seven lookup tables

Applicability of Look-up Table: The above look-up table shall be applicable for i) students who seek readmission from BT23 regulations to BT25 regulation and are going to be re-admitted in the same college and ii) detained students of one JNTUH affiliated non-autonomous college who seek admission into another JNTUH affiliated non-autonomous college.

For these two categories of students, the Principals of the affiliated colleges need not consult the University for the equivalence courses. However the Principals need to inform in the specified format, the list of such students and equivalences derived from the transitory regulations.

30.7 These look-Up tables are not applicable for i) the students who seek transfer from other Universities to JNTUH affiliated colleges, autonomous to non-autonomous and non-autonomous to autonomous colleges under JNTUH. Such students should consult the University regarding equivalent courses, as was in previous practice.

30.8 The BT25 Academic Regulations are applicable to a student from the year of re-admission. However, the student is required to complete the study of B.Tech. degree within the stipulated period of eight academic years from the year of first admission.

31. STUDENT TRANSFERS

31.1 There shall be no branch transfers after the completion of admission process.

32.2 There shall be no transfers from one college to another within the constituent colleges and units of Jawaharlal Nehru Technological University Hyderabad.

33.3 The students seeking transfer to colleges affiliated to JNTUH from various other universities / institutions have to pass the failed courses which are equivalent to the courses of JNTUH in JNTUH system, and also pass the additional courses of JNTUH which the students have not studied at the earlier institution.

34.4 The transferred students from other Universities/Institutions to JNTUH affiliated colleges, shall be given a chance to write CBTs for getting CIE component in the equivalent course(s) as per the clearance letter issued by the University.

32. ACADEMIC REGULATIONS FOR B.TECH. (LATERAL ENTRY SCHEME) FROM THE ACADEMIC YEAR: 2026-27

Eligibility for the award of B.Tech. Degree (LES):

1. The LES students after securing admission shall pursue a course of study for not less than three academic years and not more than six academic years.
2. The student shall register for 123/124 credits and secure 120 credits with CGPA ≥ 5 from III semester to VIII semester, B.Tech. program (LES) for the award of B.Tech. degree.
3. The student can avail exemption of courses totaling up to 3/4 credits other than Professional core courses, Laboratory Courses, Seminars, Project Work and Field Based Research Project Industry Oriented Mini Project / Internship, for optional drop out.
4. The students, who fail to fulfil the requirement for the award of the degree in six academic years from the year of admission, shall forfeit their seat in B.Tech.
5. The attendance requirements of B.Tech. (Regular) shall be applicable to B.Tech. (LES).

6. **Promotion Rules**

The following academic requirements have to be satisfied in addition to the attendance requirements mentioned in item no. 9.

S.No	Promotion	Conditions to be Fulfilled
1	Second year first semester to Second year second semester	Regular course of study of second year first semester and fulfilment of attendance requirement.
2	Second year second semester to Third year first semester	(i) Regular course of study of second year second semester and fulfilment of attendance requirement. (ii) Must have secured at least 25% of the total credits up to second year second semester from all the relevant regular and supplementary examinations, whether the student takes those examinations or not.
3	Third year first semester to Third year second semester	Regular course of study of third year first semester and fulfilment of attendance requirement.
4	Third year second semester to Fourth year first semester	Regular course of study of third year second semester and fulfilment of attendance requirement
5	Fourth year first semester to Fourth year second semester	Regular course of study of fourth year first semester and fulfilment of attendance requirement

7. All the other regulations as applicable to B.Tech. 4-year degree course (Regular) will hold good for B.Tech. (Lateral Entry Scheme).
8. LES students are not permitted to exit the B.Tech. program after completion of second year (B.Tech. IV Semester).

33. REVISION OF REGULATIONS AND CURRICULUM

The Institute from time to time may revise, amend or change the regulations, scheme of examinations and syllabi if found necessary and on approval by the Academic Council and the Governing Body shall be binding on the students, faculty, staff, all authorities of the Institute and others concerned.

FAILURE TO READ AND UNDERSTAND THE REGULATIONS IS NOT AN EXCUSE

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

COURSE CATALOGUE - REGULATIONS: BT25

COMPUTER SCIENCE AND ENGINEERING

I SEMESTER

Course Code	Course Name	Subject Area	Category	Periods Per Week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
INDUCTION PROGRAM										
THEORY										
AHSE01	Matrices and Calculus	BSC	Foundation	3	1	0	4	40	60	100
AHSE03	Engineering Chemistry	BSC	Foundation	3	0	0	3	40	60	100
AEEE01	Basic Electrical and Electronics Engineering	ESC	Foundation	3	0	0	3	40	60	100
ACSE01	Object Oriented Programming	CSC	Foundation	3	0	0	3	40	60	100
ACSE02	Essentials of Problem Solving	CSC	Foundation	3	0	0	3	40	60	100
PRACTICAL										
AHSE06	Engineering Chemistry Laboratory	BSC	Foundation	0	0	2	1	40	60	100
ACSE03	Object Oriented Programming Laboratory	CSC	Foundation	0	0	2	1	40	60	100
ACSE04	Front-End Web Development Laboratory	CSC	Foundation	0	0	2	1	40	60	100
AMEE03	Computer Aided Engineering Graphics	MEC	Foundation	1	0	2	2	40	60	100
TOTAL				**16**	**1**	**8**	**21**	**360**	**540**	**900**

II SEMESTER

Course Code	Course Name	Subject Area	Category	Periods Per Week			Credits	Scheme of Examination Max. Marks		
				L	T	P		CIA	SEE	Total
THEORY										
AHSE02	Engineering Physics	BSC	Foundation	3	0	0	3	40	60	100
AHSE08	Ordinary Differential Equations and Vector Calculus	BSC	Foundation	3	0	0	3	40	60	100
ACSE05	Data Structures	CSC	Foundation	3	0	0	3	40	60	100
AHSE04	English for Skill Enhancement	HSC	Foundation	3	0	0	3	40	60	100
ACSE06	AI Foundations	CSC	Foundation	3	0	0	3	40	60	100
PRACTICAL										
AHSE05	Engineering Physics Laboratory	BSC	Foundation	0	0	2	1	40	60	100
ACSE08	Data Structures Laboratory	CSC	Foundation	0	0	2	1	40	60	100
AHSE07	English Language and Communication Skills Laboratory	HSC	Foundation	0	0	2	1	40	60	100
ACSE07	Programming for Problem Solving Laboratory	CSC	Foundation	0	0	2	1	40	60	100
AMEE02	Engineering Workshop	MEC	Foundation	0	0	2	1	40	60	100
TOTAL				**15**	**00**	**10**	**20**	**400**	**600**	**1000**

COURSE CONTENT

MATRICES AND CALCULUS								
I Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / IT								
Course Code	Category	Hours/Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSE01	Foundation	3	1	-	4	40	60	100
Contact Classes: 48	Tutorial Classes: 16	Practical Classes: Nil				Total Classes: 64		
Prerequisite: Basic Principles of Algebra and Calculus								

I. COURSE OVERVIEW:

This course Matrices and Calculus is a foundation course of mathematics for all engineering branches. The concepts of Matrices, Eigen Values, Eigen Vectors, Functions of Single and Several Variables, Fourier Series and Multiple Integrals. This course is applicable for simulations, colour imaging process, finding optimal solutions in all fields of industries.

II. COURSE OBJECTIVES:

The students will try to learn:

I The concept of the rank of a matrix, solve the system of linear equations, eigen values, eigen vectors.

II The geometrical approach to the mean value theorems and their application to the mathematical problems.

III The Fourier series expansion in standard intervals as well as arbitrary intervals.

IV The evaluation of multiple integrals and their applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Determine the rank and solutions of linear equations with elementary operations.
CO 2	Utilize the Eigen values, Eigen vectors for developing spectral matrices.
CO 3	Make use of Cayley-Hamilton theorem for finding powers of the matrix.
CO 4	Apply the mean value theorems for finding analytical problems involving derivatives.
CO 5	Interpret the maxima and minima of given functions by finding the partial derivates.
CO 6	Determine the area of solid bounded regions by using the integral calculus.

IV. COURSE CONTENT:

MODULE - I: MATRICES (09)

Rank of a matrix by Echelon form and Normal form, Inverse of non-singular matrices by Gauss-Jordan method, system of linear equations: Solving system of homogeneous and non-homogeneous equations. Gauss Seidel iteration method.

MODULE - II: EIGEN VALUES AND EIGEN VECTORS (10)

Linear transformation and orthogonal transformation: Eigen values, Eigen vectors and their properties, diagonalization of a matrix, Cayley-Hamilton theorem (without proof), finding inverse and power of a matrix by Cayley-Hamilton theorem, Quadratic forms and nature of the Quadratic forms, reduction of Quadratic form to canonical form by orthogonal transformation

MODULE - III: SINGLE VARIABLE CALCULUS (10)

Limit and continuous of functions and its properties. mean value theorems: Rolle's theorem, Lagrange's mean value theorem with their geometrical interpretation and applications.

Cauchy's mean value theorem, Taylor's series (all the theorems without proof).

Curve Tracing: Curve tracing in cartesian coordinates.

MODULE - IV: MULTIVARIABLE CALCULUS (9)

Definitions of limit and continuity, partial differentiation: Euler's theorem, total derivative, Jacobian, functional dependence & independence. Applications: maxima and minima of functions of two variables and three variables using method of Lagrange multipliers

MODULE - V: MULTIPLE INTEGRALS (10)

Evaluation of double integrals (cartesian and polar coordinates), change of order of integration (only cartesian form), change of variables for double integrals (cartesian to polar). evaluation of triple integrals, change of variables for triple integrals (cartesian to spherical and cylindrical polar coordinates). Applications: areas by double integrals and volumes by triple integrals.

V.TEXT BOOKS:

I. B. S. Grewal, *Higher Engineering Mathematics*, 44/e, Khanna Publishers, 2017.
II. Erwin Kreyszig, *Advanced Engineering Mathematics*, 10/e, John Wiley & Sons, 2011.

VI.REFERENCE BOOKS:

I. R. K. Jain and S. R. K. Iyengar, *Advanced Engineering Mathematics*, 3/ed, Narosa Publications, 5th Edition, 2016.
II. George B. Thomas, Maurice D. Weir and Joel Hass, Thomas, *Calculus*, 13/e, Pearson Publishers, 2013.
III. N.P. Bali and Manish Goyal, *A text book of Engineering Mathematics*, Laxmi Publications, Reprint, 2008.
IV. Dean G. Duffy, *Advanced Engineering Mathematics with MATLAB*, CRC Press.
V. Peter O'Neil, *Advanced Engineering Mathematics*, Cengage Learning.
VI. B.V. Ramana, *Higher Engineering Mathematics*, McGraw Hill Education.

VII. ELECTRONIC RESOURCES:

I. Engineering Mathematics - I, By Prof. Jitendra Kumar | IIT Kharagpur
https://onlinecourses.nptel.ac.in/noc23_ma88/preview
II. Advanced Calculus for Engineers, By Prof. Jitendra Kumar, Prof. Somesh Kumar | IIT Kharagpur
https://onlinecourses.nptel.ac.in/noc23_ma86/preview
III. http://www.efunda.com/math/math_home/math.cfm
IV. http://www.ocw.mit.edu/resourcs/#Mathematics
V. http://www.sosmath.com
VI. http://www.mathworld.wolfram.com

VIII. MATERIAL ONLINE:

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open end experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper - II
9. Lecture notes
10. E-learning readiness videos (ELRV)
11. Power point presentation

COURSE CONTENT

ENGINEERING CHEMISTRY

I Semester: CSE/ IT
II Semester: AE / ME / CE / ECE / EEE/ CSE (AI & ML) /CSE(DS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AHSE03	Foundation	3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		

Prerequisite: Basic principles of chemistry

I. COURSE OVERVIEW:

The course focuses on the fundamental concepts of chemistry and then builds an interface with their industrial applications. It deals with the water purification processes, electrochemical principles in batteries, corrosion of metallic structures and preventive methods to control corrosion in metals, engineering materials such as plastics, fibers and elastomers, biodegradable polymers, renewable and non-renewable energy resources, nanomaterials, lubricants, biosensors and spectroscopic techniques leading to diverse applications across various fields, It cultivates the students to identify chemistry in each piece of finely engineered products used i n industries.

II. COURSES OBJECTIVES:

The students will try to learn

I. The different parameters to remove causes of hardness of water and their reactions towards complexometric method.
II. The concepts of electrochemical principles and causes of corrosion in the new developments and breakthroughs efficiently in engineering and technology.
III. The fundamental knowledge of conventional and non conventional energy sources and their applications in engineering.
IV. The different types of materials with respect to mechanisms and its significance in industrial applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Interpret the water quality characteristics for its usage in domestic and industrial purposes.
CO2	Use complexometry for calculation of hardness of water to avoid industrial problems.
CO3	Implement the principles of electrochemical systems to control the corrosion in metals.
CO4	Extend the applications of polymers based on their degradability and properties.
CO5	Choose the appropriate fuel based on their calorific value for energy efficient processes.
CO6	Predict the knowledge on viability of advanced materials for technological improvements in various sectors.

IV. COURSE CONTENT:

MODULE-I: WATER AND ITS TREATMENT (10)

Introduction: Hardness, types, degree of hardness and units ; estimation of temporary and permanent hardness of water by complexometric method, numerical problems; Potable water and its specifications (WHO), steps involved in treatment of potable water, disinfection of potable water by chlorination and breakpoint chlorination; Internal treatment of boiler feed water: Calgon conditioning, phosphate conditioning and colloidal conditioning; external treatment methods: Softening of water by ion-exchange processes; desalination of brackish water, reverse osmosis.

MODULE-II: ELECTROCHEMISTRY AND CORROSION (10)

Introduction: Electrode potential, standard electrode potential, Nernst equation (no derivation); Electrochemical cells: Galvanic cell, cell representation, EMF of cell, numerical problems; Batteries: classification of batteries, construction, working and applications of Zinc-air and Li-ion battery; Corrosion: Definition, Causes and effects of corrosion; Theories of corrosion: Chemical and electrochemical theories of corrosion; Corrosion control methods: Cathodic protection methods, sacrificial anode and impressed current methods.

MODULE-III: POLYMERS (9)

Polymers: Classification of polymers; types of polymerization-addition and condensation polymerization; Plastics, elastomers and fibers: Preparation, properties and applications of PVC, Buna-S and Nylon 6,6; Differences between thermoplastics and thermosetting plastics;

Conducting polymers: Definition, classification with examples, mechanism of conduction in trans poly acetylene and applications of conducting polymers; Biodegradable polymers: poly lactic acid and their applications.

MODULE–IV: ENERGY SOURCES (10)

Introduction and characteristics of good fuel; Fossil fuels: Introduction, classification, petroleum, refining of crude oil; Cracking: Definition, types of cracking, moving bed catalytic cracking. LPG and CNG composition and uses; Synthetic fuel: Fischer-Tropsch process; Alternative and non-conventional sources of energy: solar, wind and hydropower advantages and disadvantages; Calorific value: units, HCV and LCV and Dulongs formula, numerical problems.

MODULE-V: ADVANCED FUNCTIONAL MATERIALS (9)

Nanomaterials: Introduction, preparation of nanomaterials by sol-gel method, chemical reduction method and applications of nanomaterials. Biosensors: Definition, Amperometric glucose monitor sensor; IR spectroscopy in night vision-security; Pollution Under Control, CO sensor, Passive Infrared detection; Raman spectroscopy application, Tumour detection in medical applications; Lubricants: characteristics of a good lubricant; properties of lubricants: viscosity, flash and fire point, cloud and pour point.

V. TEXT BOOKS:

1. JAIN &JAIN, P.C. Jain, Monika Jain, *Engineering Chemistry* , Dhanpat Rai publishing Company (P) limited, 17th edition, 2022.
2. Shashi Chawla, *Text Book of Engineering Chemistry*, Dhanat Rai and Company (P) Limited, 1st Edition, 2017.

VI. REFERENCE BOOKS:

1. Ramadevi, Dr, P Aparna and Rath, Cengage learning, 13th edition, 2025.
2. Donald. J Leo, Wiley, Engineering analysis of smart material systems, 1st Edition, 2007.
3. **Nitin K Puri, Nanomaterials Synthesis Properties and Applications, I K international publishing house pvt Ltd,**1st edition **2021.**

VII. ELECTRONICS RESOURCES:

1. Engineering chemistry (NPTEL Web-book), by B. L. Tembe, Kamaluddin and M. S.Krishnan.http://www.cdeep.iitb.ac.in/webpage_data/nptel/Core%20Science/Engineering%20Chemistry%201/About- Faculty.html
2. https://books.google.co.in/books?id=R1JtyILNIsAC&pg=PR3&source=gbs_selected_pages&cad=3#v=onepage&q&f=false
3. https://books.google.co.in/books?id=eQTLCgAAQBAJ&pg=SA1PA53&source=gbs_selected_pages&cad=3#v=onepage&q&f=false

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open end experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper - II
9. Lecture notes
10. E-learning readiness videos (ELRV)
11. Power point presentation

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

BASIC ELECTRICAL AND ELECTRONICS ENGINEERING

I Semester: CSE / IT
II Semester: CSE (AIML) / CSE(DS) / AERO / MECH / CE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AEEE01	Foundation	3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

The course introduces the basic concepts of circuit analysis which is the foundation for all subjects of the electrical and electronics engineering. It includes the basic fundamental laws of electricity and magnetism with an emphasis on resistors, inductors and capacitors (RLC) circuits applied to alternating current (AC) or direct current (DC) of electrical networks. This course provides the hands-on experience on designing circuits using Diodes, Bipolar Junction Transistors, and Field Effect Transistors. Provides the capability to extract the characteristics of semiconductor devices and circuits with simulation tools.

II. COURSE OBJECTIVES:

The students will try to learn

I The fundamental principles of electrical circuits including DC and AC systems, and their analysis using laws like KVL and KCL.

II The electrical installations, components of LT switchgear, battery characteristics, and methods for calculating power and energy consumption.

III The construction, working principles, and performance analysis of electrical machines such as transformers, DC motors/generators, and induction motors.

IV The basics of semiconductor devices including diodes, rectifiers, BJTs, and FETs, along with their applications in electronics.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1 Analyze and solve simple electrical circuits using Ohm's Law, Kirchhoff's laws, and phasor techniques for both DC and single-phase/three-phase AC circuits.

CO2 Identify various components of LT switchgear, types of batteries, and perform basic calculations related to energy consumption and battery backup

CO3 Explain the construction, working principles, and characteristics of electrical machines including transformers, DC motors/generators, and three-phase induction motors

CO4 Demonstrate an understanding of the operation and characteristics of P-N junction and Zener diodes, and their role in rectifier and filter circuits.

CO5 Analyze BJT and FET configurations to understand their working, amplification modes, and performance comparisons.

CO6 Apply the knowledge of electrical and electronic components to real-world applications such as power systems, electronics circuits, and energy management.

IV. COURSE CONTENT:

MODULE – I: INTRODUCTION TO ELECTRICAL CIRCUITS (10)

D.C. Circuits: Electrical circuit elements (R, L and C), voltage and current sources, KVL and KCL, analysis of simple circuits with dc excitation.

A.C. Circuits: Representation of sinusoidal waveforms, peak and rms values, phasor representation, real power, reactive power, apparent power, power factor, Analysis of single-phase ac circuits, Three phase balanced circuits, voltage and current relations instar and delta connections.

MODULE – II: ELECTRICAL INSTALLATIONS (08)

Electrical Installations: Components of LT Switchgear: Switch Fuse Unit (SFU), MCB, ELCB, MCCB, Types of Wires and Cables, Earthing. Types of Batteries, Important Characteristics for Batteries. Elementary calculations for energy consumption, power factor improvement and battery backup.

MODULE – III: ELECTRICAL MACHINES (10)

Electrical Machines: Working principle of Single-phase transformer, equivalent circuit, losses in transformers, efficiency, three phase transformer connections. Construction and working principle of DC generators, EMF equation, working principle of DC motors.

Torque equations and Speed control of DC motors, Construction and working principle of Three phase Induction motor, Torques equations and Speed control of Three phase induction motor. Construction and working principle of synchronous generators.

MODULE – IV: DIODES AND RECTIFIERS (10)

P-N Junction and Zener Diode: Principle of Operation Diode equation, Volt, Ampere characteristics, Temperature dependence, Ideal versus practical, Static and dynamic resistances, Equivalent circuit, Zener diode characteristics and applications.

Rectifiers and Filters: P-N junction as a rectifier, Half Wave Rectifier, Ripple Factor, Full Wave Rectifier, Bridge Rectifier, Harmonic components in Rectifier Circuits, Filters – Inductor Filters, Capacitor Filters, L-section Filters, π- section Filters.

MODULE – V: BIPOLAR JUNCTION TRANSISTORS AND FIELD EFFECT TRANSISTOR (10)

Bipolar Junction Transistor (BJT): Construction, Principle of Operation, Amplifying Action, Common Emitter, Common Base and Common Collector configurations, Comparison of CE, CB and CC configurations.

Field Effect Transistor (FET): Construction, Principle of Operation, Comparison of BJT and FET, Biasing FET.

V. TEXT BOOKS:

1. Basic Electrical and electronics Engineering, M S Sukija and TK Nagasarkar, Oxford University, 1st Edition, 2012.
2. Basic Electrical and electronics Engineering, D P Kothari and I J Nagarath, McGraw Hill Education, 2nd Edition, 2020.

VI.REFERENCE BOOKS:

1. Electronic Devices and Circuits, R. L. Boylestad and Louis Nashelsky, PEI and PHI,9th Edition, 2006.
2. Millman's Electronic Devices and Circuits,J. Millman, C. C. Halkias and Satyabrata Jit, TMH, 2nd Edition, 1998.
3. Fundamentals of Electrical Engineering, L. S. Bobrow, Oxford University Press, 12th edition, 2003.
4. Electrical and Electronic Technology, E. Hughes, Pearson Education, 10th Edition, 2010.
5. Electrical Engineering Fundamentals, V. D. Toro, Prentice Hall India, 2nd Edition, 1989.

VII. WEB REFERENCES:

1. https://www.igniteengineers.com
2. https://www.ocw.nthu.edu.tw
3. https://www.uotechnology.edu.iq
4. https://www.iare.ac.in

II. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Assignments
5. Definitions and terminology
6. Open ended experiments
7. Model question paper-I
8. Model question paper-II
9. Lecture notes
10. Power point presentations
11. ELRV videos

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

OBJECT ORIENTED PROGRAMMING

I Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI & ML) / CSE (DS) / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACSE01	Foundation	L	T	P	C	CIA	SEE	Total
		3	0	0	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

This course introduces the principles of Object-Oriented Programming (OOP) and its role in solving complex problems effectively. It provides a solid foundation in object-oriented concepts such as abstraction, encapsulation, inheritance, polymorphism, and collaboration. The course also extends into file handling, exception management, and concurrent execution, preparing students to design, develop, and manage robust real-world applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts and principles of object-oriented programming in high-level programming languages.

II. The advanced concepts for developing well-structured and efficient programs that involve complex data structures, numerical computations, or domain-specific operations.

III. The design and implementation features such as inheritance, polymorphism, and encapsulation for tackling complex problems and creating well-organized, modular, and maintainable code.

IV. The usage of input/output interfaces to transmit and receive data to solve real-time computing problems.

III. COURSE OUTCOMES:

At the end of the course, students should be able to:

CO1	Identify appropriate programming approaches to manage complexity.
CO2	Design modular, reusable, and adaptable software systems.
CO3	Apply structured problem-solving techniques to build reliable and maintainable applications.
CO4	Demonstrate the ability to handle data, manage errors, and ensure smooth program execution.
CO5	Develop applications that are efficient, scalable, and suitable for real-world scenarios
CO6	Develop contemporary solutions to software design problems using object-oriented principles.

IV. COURSE CONTENT:

MODULE - I: Object-oriented concepts (10)
Complex systems: definition, characteristics, and five attributes (hierarchy, abstraction, emergence, encapsulation, modularity).
Evolution of problem-solving: procedural vs. object-oriented thinking.
Objects as fundamental building blocks: state, behavior, and identity.
Benefits of OOP in managing complexity, Applications of OOP in real-world systems.

MODULE II: Abstraction, Encapsulation and Object Collaboration (09)
Abstraction: forms of abstraction (procedural, data, control), abstraction layers, mechanisms.
Encapsulation: information hiding, boundary definition, modularity.
Objects and message passing: collaboration through responsibilities.
Relationships: association, aggregation, composition, dependency.

MODULE III: Inheritance and Generalization (10)
Classification and taxonomy in object-oriented programming, Concepts of generalization and specialization.
Types of inheritance: single, multiple, and hierarchical (conceptual).

Challenges in multiple inheritance: ambiguity and the diamond problem (conceptual).
Importance of generalization for adaptability and method reuse.

MODULE IV: Polymorphism and Interfaces (09)
Polymorphism: static vs dynamic polymorphism, Abstract classes, abstract operations, late binding, and dynamic dispatch.
Interfaces as behavioral contracts, difference between interfaces and abstract classes (conceptual), Multiple realizations of interfaces (role-based modeling).

MODULE V: File structures, Exception handling, Concurrent execution (09)
Working with Files: Files, need for file handling, types, modes, operations and error handling**.**
Exception handling: Detecting problems during execution and responding gracefully, preventing failures from crashing the system and ensuring smooth execution.
Concurrent execution: Allowing multiple tasks to run simultaneously within a system, co-ordinating tasks to avoid conflicts when sharing resources.

V. TEXTBOOKS:

1. Matt Weisfeld, *The Object-Oriented Thought Process*, Addison Wesley Object Technology Series, 4th Edition, 2013.
2. Grady Booch, *Object-Oriented Analysis and Design with Applications*, Addison-Wesley Professional, 3rd Edition, 2007.
3. Craig Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Addison-Wesley Professional, 3rd Edition, 2004.

VI. REFERENCE BOOKS:

1. Timothy Budd, *Introduction to object-oriented programming*, Addison Wesley Object Technology Series, 3rd Edition, 2002.
2. Gaston C. Hillar, *Learning Object-Oriented Programming*, Packt Publishing, 2015.
3. Kingsley Sage, *Concise Guide to Object-Oriented Programming*, Springer International Publishing, 1st Edition, 2019.
4. Rudolf Pecinovsky**,** *OOP - Learn Object Oriented Thinking and Programming*, Tomas Bruckner, 2013.

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/javase/tutorial/java/concepts/
2. https://www.w3schools.com/cpp/
3. https://www.edx.org/learn/object-oriented-programming/
4. https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation
11. E-Learning Readiness Videos (ELRV)

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

ESSENTIALS OF PROBLEM SOLVING

I Semester: CSE / IT / CSE (AI&ML) / CSE (DS)
II Semester: ECE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSE02	Foundation	3	0	0	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		

Prerequisite: There is no prerequisite to take this course

I. COURSE OVERVIEW:

This course aims to provide exposure to problem solving through programming. Useful graph theory concepts, numerical techniques, and their applications to real world problems are discussed. Graph theoretical notions and the use of algorithms, both in the mathematical theory of graphs and its applications are discussed. Student will also learn how to implement and interpret numerical solutions by writing a well-designed computer programs in regard to their efficiency and suitability for real-life applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The fundamental concepts of graph theory and its properties.
II. The basics related to paths and cycles using Eulerian and Hamiltonian cycles.
III. The applications of graph colouring and traversal algorithms for solving real-time problems.
IV. The numerical methods to solve algebraic equations.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Outline the graph terminologies, graph representation, and relate them to practical examples.
CO2	Build efficient graph routing algorithms for various optimization problems on graphs.
CO3	Use effective techniques from graph theory to solve problems in networking and telecommunication.
CO4	Interpret the fundamental concepts of polynomials, roots of equations and solve corresponding problems using computer programs.
CO5	Apply the knowledge of numerical methods to solve algebraic and transcendental equations arising in real-life situations.
CO6	Solve numerical integrals and ordinary differential equations to simulate discrete time algorithms.

IV. COURSE CONTENT:

MODULE - I GRAPH THEORY (08)

Graph terminology, digraphs, weighted graphs, complete graphs, graph complements, bipartite graphs, graph combinations, isomorphisms, matrix representations of graphs, incidence and adjacency matrices, degree sequence.

MODULE - II GRAPH ROUTES (10)

Eulerian circuit: Konigsberg bridge problem, touring a graph; Eulerian graphs, Hamiltonian cycles, the traveling salesman problem; Shortest paths: Dijkstra's algorithm, walks using matrices.

MODULE - III GRAPH COLORING AND GRAPH ALGORITHMS (10)

Four color theorem, vertex coloring, edge coloring, coloring variations, first-fit coloring algorithm.

Graph traversal: depth-first search, bread-first search and its applications; Minimum spanning trees: Kruskal's and Prim's algorithm, union-find structure.

MODULE - IV: ALGEBRAIC AND TRANSCENDENTAL EQUATIONS (10)

Algebraic equations, method of false position, bisection method, iteration method, Newton-Raphson method, Secant method, Ramanujan's Method, Muller's method (Approximation up to 2 decimals only).

MODULE - V: NUMERICAL INTEGRATION AND ORDINARY DIFFERENTIATIAL EQUATIONS (10)

Trapezoidal rule, Simpson's 1/3 rule, Simpson's 3/8 rule, Solution by Taylor's series, Euler's method of solving an ordinary differential equation numerically, Runge-Kutta's second order method of solving ordinary differential equations (Approximation up to 2 decimals only).

V. TEXT BOOKS:

4. Karin R Saoub, *Graph Theory: An Introduction to Proofs, Algorithms, and Applications*, 1st edition, Chapman and Hall, 2021.
5. S S Sastry, *Introductory Methods of Numerical Analysis*, 5th edition, 2012.

VI. REFERENCE BOOKS:

1. Mahinder Kumar Jain, *Numerical Methods: For Scientific and Scientific Computation*, New Age International Pvt. Ltd., 7th edition, 2019.
2. P Kandasamy, K Thilagavathy, K Gunavathi, *Numerical Methods*, S Chand and Company, 2006.
3. R Balakrishnan, K Ranganathan, *A Textbook of Graph Theory*, Springer Exclusive, 2nd edition, 2019.
4. Jann Kiusalaas, *Numerical Methods in Engineering with Python*, Cambridge University Press, 2nd edition 2010.
5. Gary Chartrand, Ping Zhang, *A First Course in Graph Theory*, Dover Publications Inc., 2012.
6. James F. Epperson, *An Introduction to Numerical Methods and Analysis*, Wiley, 2nd edition, 2013.

VII. ELECTRONICS RESOURCES:

1. https://www.geeksforgeeks.org/numerical-methods-and-calculus-gq/
2. https://www.geeksforgeeks.org/program-for-bisection-method/
3. https://ocw.mit.edu/courses/2-993j-introduction-to-numerical-analysis-for-engineering-13-002j-spring-2005/pages/lecture-notes/
4. https://www.tutorialspoint.com/graphs-and-its-traversal-algorithms
5. https://web.mit.edu/urban_or_book/www/book/chapter6/6.4.4.html
6. https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/tutorial/
7. https://www.codingninjas.com/studio/library/euler-and-hamilton-paths

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech-talk topics
4. Open-ended experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper – II
9. Lecture notes
10. PowerPoint presentation

I A R E

EDUCATION FOR LIBERATION

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

ENGINEERING CHEMISTRY LABORATORY								
I Semester: CSE / IT **II Semester:** AE / ME / CE / ECE / EEE / CSE (AI&ML) / CSE (DS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AHSE06	Foundation	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		-	-	2	1	40	60	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 36**				**Total Classes: 36**		
Prerequisite: Basic Principles of Chemistry								

I. COURSE OVERVIEW:

The course encourages introducing analytical tools in an Engineering perspective. The course efforts to provide the basic knowledge of analytical methodology, outlines the importance of volumetric analysis, comprehensive instrumental analysis for properties of fuels, colorimetric analysis and spectroscopic analysis. This practical approach gives the essence of analytical chemistry for skill development in determinations of materials properties and its viability in the industry.

II. COURSES OBJECTIVES:

The students will try to learn:

I The quantitative analysis to know the strength of unknown solutions by instrumental methods.
II The troubles of hard water and its estimation by analytical techniques.
III The applications of appropriate lubricant for finely tuned machinery.
IV The basic knowledge on quantity of light absorbed by the materials.

III. COURSE OUTCOMES:

After successful completion of the course students should be able to:

CO1	Use analytical techniques like conductometry and pH metry to recognize the electrical properties of solutions
CO2	Utilize the potentiometer to characterize and measure the electrical potential of an analyte
CO3	Implement the principles of water analysis for domestic and industrial applications.
CO4	Synthesize the polymeric materials from monomers with polymerization process
CO5	Select different types of lubricants to know its properties for the proper lubrication of machinery in industries.
CO6	Identify the absorption tendency of solids or liquids by using colorimetry

IV. COURSE CONTENT:

1. GETTING STARTED EXERCISES

1.1 Introduction to Chemistry Laboratory

The fundamental concepts and theories required for carrying out qualitative and quantitative analysis. Detailed explanation on the analytical techniques used for qualitative analysis. Emphasis on instrumental method of analysis and its advantages over conventional methods.

i. Types of analysis
ii. Difference between qualitative and quantitative analysis
iii. Common techniques of qualitative and quantitative analysis
iv. Introduction to instrumental method of analysis
v. Introduction to basic techniques and handling of common apparatus
vi. Discussion of Material Safety Data Sheet (MSDS) of chemicals
vii. Identification of toxic signs and safety procedures of chemical laboratory

1.2 Safety Guidelines to Chemistry Laboratory

The chemistry laboratory must be a safe place in which to work and learn about chemistry.

i. Wear a chemical-resistant apron.
ii. Be familiar with your lab work sheet before you come to lab. Follow all written and verbal instructions carefully. Observe the safety alerts in the laboratory directions. If you do not understand a direction or part of a procedure, ask the teacher before proceeding.
iii. When entering the laboratory room, do not touch any equipment, chemicals, or other materials without being instructed to do so. Perform only those experiments authorized by the instructor.
iv. If you take more of a chemical substance from a container than you need, you should not return the excess to the container. This might cause contamination of the substance remaining. Dispose of the excess as your instructor directs.
v. Never smell anything in the laboratory unless your teacher tells you it is safe. Do not smell a substance by putting your nose directly over the container and inhaling. Instead, waft the vapors toward your nose by gently fanning the vapors toward yourself.
vi. Do not directly touch any chemical with your hands. Never taste materials in the laboratory.
vii. Work areas should be kept clean and tidy at all times. Always replace lids or caps on bottles and jars.

1.3 Data recording and reports

Students must record their experimental values in the provided tables in this laboratory manual and reproduce them in the laboratory worksheets. Worksheets are integral to recording the methodology and results of an experiment. In engineering practice, the laboratory worksheets serve as a valuable reference to the technique used in the laboratory. Note that the data collected will be an accurate and permanent record of the data obtained during the experiment and the analysis of the results.

2. CONDUCTOMETRY

2.1 Estimation of the concentration of strong acid using conductometer

i. The basic principle of conductometric titrations
ii. Titration of unknown solution of acid with base
iii. Graphical plots on volume of titrant vs. conductance

3. CONDUCTOMETRY

3.1 Estimation of concentration of strong and weak acid in an acid mixture using conductometer

i. The basic principle of conductometric titrations
ii. Titration of unknown solution of mixture of acid with base
ii. Graphical plots on volume of titrant vs. conductance

4. POTENTIOMETRY

4.1 Estimation of iron content of the given solution by $K_2Cr_2O_7$ using potentiometer

i. The basic principle of potentiometric titrations
ii. Titration of Mohr's salt with potassium dichromate
iii. Graphical plots on volume of titrant vs. potential

5. POTENTIOMETRY

5.1 Estimation of concentration of hydrochloric acid using potentiometer

i. The basic principle of conductometric titrations
ii. Titration of unknown solution of acid with base
iii. Graphical plots on volume of titrant vs. conductance

6. pH METRY

6.1 Determination of strength of given hydrochloric acid using pH meter

i. The basic principle of pH metry
ii. Titration of unknown solution with standard acid
iii. Graphical plots on volume of titrant vs. pH to obtain equivalence point

7. MEASUREMENT OF TOTAL DISSOLVED SOLIDS IN WATER

7.1 Measurement of total dissolved solids (TDS) in different water samples

i. Specifications of potable water
i. Measure the total dissolved solids in different water samples by TDS meter

8. COMPLEXOMETRY METHOD

8.1 Estimate the total hardness of water by EDTA

i. Principle of complexometric titration
ii Titration of water samples by using EDTA to find the total hardness in water.

9. PRECIPITATION METHOD

9.1 Determination of chloride content in water by Argentometry

i. Principle of Argentometric titration
ii. Titration of water samples by using $AgNO_3$ to find the chloride content in water.

10 PREPARATION OF POLYMER

10.1 Preparation of Thiokol rubber by using sodium polysulphide.

i. Significance of artificial rubbers in industries
ii. Synthesize the Thiokol rubeer by using sodium polysulphide and ethylene dichloride

11. VISCOSITY OF LUBRICANT

11.1 Determine the viscosity of the lubricants using Ostwald's viscometer

i. The principle of viscosity of lubricant
ii. Significance of viscosity index of lubricant
iii Viscosity of given lubricant by using Ostwald's viscometer

12. PROPERTIES OF LUBRICANTS

12.1 Determine the flash and fire points of lubricants

i. Significance of flash and fire point of lubricant in industries
ii. Flash and Fire points of a given lubricant by using Pensky Martens flash point apparatus

13. CLOUD AND POUR POINT OF LUBRICANTS

13.1 Determination of cloud and pout point of lubricants

i. Significance of cloud and pout point of lubricant in industries
ii. Cloud and pour point of given lubricants by using cloud and pour point tester.

14. COLORIMETRY

14.1 Estimate the metal ion concentration using colorimeter

i. Complexation of metal ion with ligands
ii. Detection of absorbance of the colored metal -ligand complex solution
iii Graphical determination of concentration of the metal ions in the solution

V. TEXTBOOKS:

1.K. Mukkanti et. al,Practical *Engineering Chemistry*, B.S. Publications, Hyderabad.
2. Vogel's, *Quantitative chemical analysis*, prentice Hall, 6th Edition, 2009.

VI. REFERENCE BOOKS:

1. Solanki, M. K. Engineering Chemistry Laboratory Manual. (Edu creation Publishing, 2019).
2. Jeffery, G. H. in TEXTBOOK OF QUANTITATIVE CHEMICAL ANALYSIS (ed John Wiley and Sons) (1989).
3. Gary-D-Christian, P. K. S. D., Kevin A. Schug. Analytical-Chemistry-by-Gary-D-Christian. 7 edn, Vol. 7 826 (Wiley, 2014).
4. Budinski, Kenneth G., Engineering materials: properties and selection, 5th edition, Prentice-Hall, 1996, pg.423.
5. B. Ramadevi and P. Aparna, S Chand publications, *lab manual for engineering chemistry*, S Chand publications, NewDelhi,1st Edition 2022.

VII. ELECTRONICS RESOURCES:

1. https://nptel.ac.in/translation
2. https://nptel.ac.in/courses/115105120
3. https://archive.nptel.ac.in/courses/122/101/122101001/#

VIII. MATERIALS ONLINE

1. Chemistry Lab Course Template
2. Chemistry Lab Manual

COURSE CONTENT

OBJECT ORIENTED PROGRAMMING LABORATORY								
I Semester: AE / ME / CE / ECE / EEE / CSE / IT / CSE (AI&ML) / CSE (DS)								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACSE03	Foundation	L	T	P	C	CIA	SEE	Total
		0	0	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: NIL	Practical Classes: 30				Total Classes: 30		
Prerequisite: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

This course provides a solid foundation in object-oriented programming concepts and hands-on experience in using them. It introduces the concepts of abstraction and reusable code design via the object-oriented paradigm. Through a series of examples and exercises students gain coding skills and develop an understanding of professional programming practices. Mastering Java facilitate the learning of other technologies.

II. COURSES OBJECTIVES:

The students will try to learn

I. The strong foundation with the Java Virtual Machine, its concepts and features.
II. The systematic understanding of key aspects of the Java Class Library
III. The usage of a modern IDE with an object oriented programming language to develop programs.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Develop non-trivial programs in a modern programming language.
CO 2	Apply the principles of selection and iteration.
CO 3	Appreciate uses of modular programming concepts for handling complex problems.
CO 4	Recognize and apply principle features of object-oriented design such as abstraction and encapsulation.
CO 5	Design classes with a view of flexibility and reusability.
CO 6	Code, test and evaluate small use cases to conform to a specification.

IV. COURE CONTENT:

EXERCISES FOR OBJECT ORIENTED PROGRAMMING LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 HelloWorld

1. Install JDK on your machine.
2. Write a Hello-world program using JDK and a source-code editor, such as:
   - For All Platforms: Sublime Text, Atom
   - For Windows: TextPad, NotePad++
   - For macOS: jEdit, gedit
   - For Ubuntu: gedit
3. Do ALL the exercises.

1.2 Writing Good Programs

The only way to learn programming is program, program and program. Learning programming is like learning cycling, swimming or any other sports. You can't learn by watching or reading books. Start to program immediately. On the other hands, to improve your programming, you need to read many books and study how the masters program.

It is easy to write programs that work. It is much harder to write programs that not only work but also easy to maintain and understood by others – I call these *good programs*. In the real world, writing program is not meaningful. You have to write good programs, so that others can understand and maintain your programs.

Pay particular attention to:

1. Coding Style:
   - Read "Java Code Convention" (@ https://www.oracle.com/technetwork/java/codeconventions-150003.pdf or google "Java Code Convention").
   - Follow the *Java Naming Conventions* for variables, methods, and classes STRICTLY. Use *CamelCase* for names. Variable and method names begin with lowercase, while class names begin with uppercase. Use nouns for variables (e.g., `radius`) and class names (e.g., `Circle`). Use verbs for methods (e.g., `getArea()`, `isEmpty()`).
   - **Use Meaningful Names**: Do not use names like a, b, c, d, x, x1, x2, and x1688 - they are meaningless. Avoid single-alphabet names like `i`, `j`, k. They are easy to type, but usually meaningless. Use single-alphabet names only when their meaning is clear, e.g., x, y, z for co-ordinates and `i` for array index. Use meaningful names like `row` and `col` (instead of `x` and `y`, `i` and `j`, `x1` and `x2`), `numStudents` (not n), `maxGrade`, `size` (not n), and upperbound (not n again). Differentiate between singular and plural nouns (e.g., use `books` for an array of books, and `book` for each item).
   - Use consistent indentation and coding style. Many IDEs (such as Eclipse/NetBeans) can re-format your source codes with a single click.

2. Program Documentation: Comment! Comment! and more Comment to explain your code to other people and to yourself three days later.

1.3 CheckPassFail (if-else)

Write a program called **CheckPassFail** which prints "PASS" if the int variable "mark" is more than or equal to 50; or prints "FAIL" otherwise. The program shall always print “DONE” before exiting.

Hints

Use >= for *greater than or equal to* comparison.

```
 /* Trying if-else statement.
 */
public class CheckPassFail {  // Save as "CheckPassFail.java"
   public static void main(String[] args) {  // Program entry point
      int mark = 49;   // Set the value of "mark" here!
      System.out.println("The mark is " + mark);

      // if-else statement
      if ( ...... ) {
         System.out.println( ...... );
      } else {
         System.out.println( ...... );
      }
      System.out.println( ...... );
   }
}
```

Try

mark = 0, 49, 50, 51, 100 and verify your results.

Take note of the source-code **indentation**!!! Whenever you open a block with '{', indent all the statements inside the block by 3 (or 4 spaces). When the block ends, un-indent the closing '}' to align with the opening statement.

1.4 CheckOddEven (if-else)

Write a program called **CheckOddEven** which prints "Odd Number" if the int variable “number” is odd, or “Even Number” otherwise. The program shall always print “bye!” before exiting.

Hints

n is an even number if (*n* % 2) is 0; otherwise, it is an odd number. Use == for comparison, e.g., (n % 2) == 0.

```
/**
 * Trying if-else statement and modulus (%) operator.
 */
public class CheckOddEven {   // Save as "CheckOddEven.java"
   public static void main(String[] args) {  // Program entry point
      int number = 49;       // Set the value of "number" here!
      System.out.println("The number is " + number);
      if ( ...... ) {
         System.out.println( ...... );   // even number
      } else {
         System.out.println( ...... );   // odd number
      }
      System.out.println( ...... );
   }
}
```

Try

`number = 0`, `1`, `88`, `99`, `-1`, `-2` and verify your results.

Again, take note of the source-code indentation! Make it a good habit to ident your code properly, for ease of reading your program.

1.5 PrintNumberInWord (nested-if, switch-case)

Write a program called **`PrintNumberInWord`** which prints `"ONE"`, `"TWO"`,... , `"NINE"`, `"OTHER"` if the `int` variable `"number"` is `1`, `2`,... , `9`, or other, respectively. Use (a) a `"nested-if"` statement; (b) a `"switch-case-default"` statement.

Hints

```
/**
 * Trying nested-if and switch-case statements.
 */
public class PrintNumberInWord {   // Save as "PrintNumberInWord.java"
   public static void main(String[] args) {
      int number = 5;  // Set the value of "number" here!

      // Using nested-if
      if (number == 1) {   // Use == for comparison
         System.out.println( ...... );
      } else if ( ...... ) {
         ......
      } else if ( ...... ) {
         ......
      ......
      ......
      } else {
         ......
      }

      // Using switch-case-default
      switch(number) {
         case 1:
            System.out.println( ...... ); break;  // Don't forget the "break" after each case!
         case 2:
            System.out.println( ...... ); break;
         ......
         ......
         default: System.out.println( ...... );
      }
   }
}
```

Try

`number = 0`, `1`, `2`, `3`, ..., `9`, `10` and verify your results.

1.6 PrintDayInWord (nested-if, switch-case)

Write a program called **PrintDayInWord** which prints "Sunday", "Monday", ... "Saturday" if the int variable "dayNumber" is 0, 1, ..., 6, respectively. Otherwise, it shall print "Not a valid day". Use (a) a "nested-if" statement; (b) a "switch-case-default" statement.

Try

dayNumber = 0, 1, 2, 3, 4, 5, 6, 7 and verify your results.

2. Exercises on Number Systems (for Science/Engineering Students)

To be proficient in programming, you need to be able to operate on these number systems:

1. Decimal (used by human beings for input and output)
2. Binary (used by computer for storage and processing)
3. Hexadecimal (shorthand or compact form for binary)

2.1 Exercises (Number Systems Conversion)

1. Convert the following *decimal* numbers into *binary* and *hexadecimal* numbers:
   a. 108
   b. 4848
   c. 9000

   Convert the following binary numbers into hexadecimal and decimal numbers:

   a. 10000000
   b. 101010101010
   c. 1000011000

   Convert the following hexadecimal numbers into binary and decimal numbers:

   a. 1234
   b. 80F
   c. ABCDE

   Convert the following decimal numbers into binary equivalent:

   a. 123.456D
   b. 19.25D

2.2 Exercise (Integer Representation)

1. What are the ranges of 8-bit, 16-bit, 32-bit and 64-bit integer, in "unsigned" and "signed" representation?
2. Give the value of 88, 0, 1, 127, and 255 in 8-bit unsigned representation.
3. Give the value of +88, -88 , -1, 0, +1, -128, and +127 in 8-bit 2's complement signed representation.
4. Give the value of +88, -88 , -1, 0, +1, -127, and +127 in 8-bit sign-magnitude representation.
5. Give the value of +88, -88 , -1, 0, +1, -127 and +127 in 8-bit 1's complement representation.

2.3 Exercises (Floating-point Numbers)

1. Compute the largest and smallest positive numbers that can be represented in the 32-bit normalized form.

2. Compute the largest and smallest negative numbers can be represented in the 32-bit normalized form.

3. Repeat (1) for the 32-bit denormalized form.

4. Repeat (2) for the 32-bit denormalized form.

Hints:

1. Largest positive number: `S=0`, `E=1111 1110 (254)`, `F=111 1111 1111 1111 1111 1111`. Smallest positive number: `S=0`, `E=0000 00001 (1)`, `F=000 0000 0000 0000 0000 0000`.
2. Same as above, but `S=1`.
3. Largest positive number: `S=0`, `E=0`, `F=111 1111 1111 1111 1111 1111`. Smallest positive number: `S=0`, `E=0`, `F=000 0000 0000 0000 0000 0001`.
4. Same as above, but `S=1`.

2.4 Exercises (Data Representation)

For the following 16-bit codes:

```
0000 0000 0010 1010;
1000 0000 0010 1010;
```

Give their values, if they are representing:

1. a 16-bit unsigned integer;
2. a 16-bit signed integer;
3. two 8-bit unsigned integers;
4. two 8-bit signed integers;
5. a 16-bit Unicode characters;
6. two 8-bit ISO-8859-1 characters.

3. Exercises on Decision and Loop

3.1 SumAverageRunningInt (Decision & Loop)

Write a program called **`SumAverageRunningInt`** to produce the sum of `1`, `2`, `3`, ..., to `100`. Store `1` and `100` in variables `lowerbound` and `upperbound`, so that we can change their values easily. Also compute and display the average.

The output shall look like:

```
The sum of 1 to 100 is 5050
The average is 50.5
```

Hints

```
/**
 * Compute the sum and average of running integers from a lowerbound to an upperbound using loop.
 */
public class SumAverageRunningInt {   // Save as "SumAverageRunningInt.java"
   public static void main (String[] args) {
      // Define variables
      int sum = 0;          // The accumulated sum, init to 0
      double average;       // average in double
      final int LOWERBOUND = 1;
      final int UPPERBOUND = 100;
      // Use a for-loop to sum from lowerbound to upperbound
      for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
            // The loop index variable number = 1, 2, 3, ..., 99, 100
         sum += number;     // same as "sum = sum + number"
      }
      // Compute average in double. Beware that int / int produces int!
      ......
      // Print sum and average
      ......
   }
}
```

Try

1. Modify the program to use a "while-do" loop instead of "for" loop.

```
int sum = 0;
      int number = LOWERBOUND;         // declare and init loop index variable
      while (number <= UPPERBOUND) {   // test
         sum += number;
         ++number;                     // update
      }
```

2. Modify the program using do-while loop

```
int sum = 0;
      int number = LOWERBOUND;          // declare and init loop index variable
      do {
         sum += number;
         ++number;                      // update
      } while (number <= UPPERBOUND);   // test
```

3. What is the difference between "for" and "while-do" loops? What is the difference between "while-do" and "do-while" loops?
4. Modify the program to sum from `111` to `8899`, and compute the average. Introduce an `int` variable called `count` to count the numbers in the specified range (to be used in computing the average).
5. Modify the program to find the "sum of the squares" of all the numbers from `1` to `100`, i.e. `1*1 + 2*2 + 3*3 + ... + 100*100`.
6. Modify the program to produce two sums: sum of odd numbers and sum of even numbers from `1` to `100`. Also computer their absolute difference.

3.2 Product1ToN (or Factorial) (Decision & Loop)

Write a program called Product1ToN to compute the product of integers from 1 to 10 (i.e., 1×2×3×...×10), as an int. Take note that It is the same as factorial of N.

Hints

Declare an `int` variable called `product`, initialize to 1, to accumulate the product.

```
      // Define variables
      int product = 1;      // The accumulated product, init to 1
      final int LOWERBOUND = 1;
      final int UPPERBOUND = 10;
```

Try

1. Compute the product from 1 to 11, 1 to 12, 1 to 13 and 1 to 14. Write down the product obtained and decide if the results are correct.
   HINTS: Factorial of 13 (=6227020800) is outside the range of int [-2147483648, 2147483647]. Take note that computer programs may not produce the correct result even though the code seems correct!
2. Repeat the above, but use long to store the product. Compare the products obtained with int for N=13 and N=14.

   HINTS: With long, you can store factorial of up to 20.

3.3 HarmonicSum (Decision & Loop)

Write a program called **HarmonicSum** to compute the sum of a harmonic series, as shown below, where *n*=50000. The program shall compute the sum from *left-to-right* as well as from the *right-to-left*. Are the two sums the same? Obtain the absolute difference between these two sums and explain the difference. Which sum is more accurate?

$$Harmonic(n) = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

Hints

```
/**
 * Compute the sum of harmonics series from left-to-right and right-to-left.
 */
public class HarmonicSum {   // Save as "HarmonicSum.java"
   public static void main (String[] args) {
      // Define variables
      final int MAX_DENOMINATOR = 50000;  // Use a more meaningful name instead of n
      double sumL2R = 0.0;          // Sum from left-to-right
      double sumR2L = 0.0;          // Sum from right-to-left
      double absDiff;               // Absolute difference between the two sums

      // for-loop for summing from left-to-right
      for (int denominator = 1; denominator <= MAX_DENOMINATOR; ++denominator) {
          // denominator = 1, 2, 3, 4, 5, ..., MAX_DENOMINATOR
          ......
          // Beware that int/int gives int, e.g., 1/2 gives 0.
      }
      System.out.println("The sum from left-to-right is: " + sumL2R);

      // for-loop for summing from right-to-left
      ......

      // Find the absolute difference and display
      if (sumL2R > sumR2L) ......
      else ......
   }
}
```

3.4 ComputePI (Decision & Loop)

Write a program called **ComputePI** to compute the value of π, using the following series expansion. Use the maximum denominator (MAX_DENOMINATOR) as the terminating condition. Try MAX_DENOMINATOR of 1000, 10000, 100000, 1000000 and compare the PI obtained. Is this series suitable for computing PI? Why?

$$\pi = 4 \times \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \frac{1}{13} - \frac{1}{15} + \cdots \right)$$

Hints

Add to sum if the denominator `%` 4 is 1, and subtract from sum if it is 3.

```
      double sum = 0.0;
      int MAX_DENOMINATOR = 1000;   // Try 10000, 100000, 1000000
      for (int denominator = 1; denominator <= MAX_DENOMINATOR; denominator += 2) {
            // denominator = 1, 3, 5, 7, ..., MAX_DENOMINATOR
         if (denominator % 4 == 1) {
            sum += ......;
         } else if (denominator % 4 == 3) {
            sum -= ......;
         } else {   // remainder of 0 or 2
            System.out.println("Impossible!!!");
         }
      }
      ......
```

Try

1. Instead of using maximum denominator as the terminating condition, rewrite your program to use the maximum number of terms (`MAX_TERM`) as the terminating condition.

```
int MAX_TERM = 10000;  // number of terms used in computation
      int sum = 0.0;
      for (int term = 1; term <= MAX_TERM; term++) {
            // term = 1, 2, 3, 4, ..., MAX_TERM
         if (term % 2 == 1) {  // odd term number: add
            sum += 1.0 / (term * 2 - 1);
         } else {              // even term number: subtract
            ......
         }
      }
```

2. JDK maintains the value of π in a built-in double constant called `Math.PI` (=`3.141592653589793`). Add a statement to compare the values obtained and the `Math.PI`, in percents of `Math.PI`, i.e., `(piComputed / Math.PI) * 100`.

3.5 CozaLozaWoza (Decision & Loop)

Write a program called **CozaLozaWoza** which prints the numbers `1` to `110`, `11` numbers per line. The program shall print "Coza" in place of the numbers which are multiples of 3, "Loza" for multiples of 5, "Woza" for multiples of 7, "CozaLoza" for multiples of 3 and 5, and so on. The output shall look like:

```
1 2 Coza 4 Loza Coza Woza 8 Coza Loza 11
Coza 13 Woza CozaLoza 16 17 Coza 19 Loza CozaWoza 22
23 Coza Loza 26 Coza Woza 29 CozaLoza 31 32 Coza
......
```

Hints

```
public class CozaLozaWoza {    // Save as "CozaLozaWoza.java"
   public static void main(String[] args) {
      final int LOWERBOUND = 1, UPPERBOUND = 110;
      for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
            // number = LOWERBOUND+1, LOWERBOUND+2, ..., UPPERBOUND
         // Print "Coza" if number is divisible by 3
         if ( ...... ) {
            System.out.print("Coza");
         }
         // Print "Loza" if number is divisible by 5
         if ( ...... ) {
            System.out.print(.....);
         }
         // Print "Woza" if number is divisible by 7
         ......
         // Print the number if it is not divisible by 3, 5 and 7 (i.e., it has not
been processed above)
         if ( ...... ) {
            ......
         }
         // After processing the number, print a newline if number is divisible by
11;
         // else print a space
         if ( ...... ) {
            System.out.println();  // print newline
         } else {
            System.out.print( ...... );  // print a space
         }
      }
   }
}
```

Notes

1. You cannot use nested-if (if ... else if ... else if ... else) for this problem. It is because the tests are not mutually exclusive. For example, 15 is divisible by both 3 and 5. Nested-if is only applicable if the tests are mutually exclusive.

2. The tests above look messy. A better solution is to use a boolean flag to keep track of whether the number has been processed, as follows:

```
    final int LOWERBOUND = 1, UPPERBOUND = 110;
      boolean printed;
      for (int number = LOWERBOUND; number <= UPPERBOUND; ++number) {
         printed = false;  // init before processing each number
         // Print "Coza" if number is divisible by 3
         if ( ...... ) {
            System.out.print( ...... );
            printed = true;   // processed!
         }
         // Print "Loza" if number is divisible by 5
         if ( ...... ) {
            System.out.print( ..... );
            printed = true;   // processed!
         }
         // Print "Woza" if number is divisible by 7
         ......
         // Print the number if it has not been processed
         if (!printed) {
            ......
         }
         // After processing the number, print a newline if it is divisible by 11;
         // else, print a space
         ......
      }
```

3.6 Fibonacci (Decision & Loop)

Write a program called **`Fibonacci`** to print the first `20` Fibonacci numbers `F(n)`, where `F(n)=F(n–1)+F(n–2)` and `F(1)=F(2)=1`. Also compute their average. The output shall look like:

```
The first 20 Fibonacci numbers are:
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765
The average is 885.5
```

Hints

```
/**
 * Print first 20 Fibonacci numbers and their average
 */
public class Fibonacci {
   public static void main (String[] args) {
      int n = 3;          // The index n for F(n), starting from n=3, as n=1 and n=2
are pre-defined
      int fn;             // F(n) to be computed
      int fnMinus1 = 1;   // F(n-1), init to F(2)
      int fnMinus2 = 1;   // F(n-2), init to F(1)
      int nMax = 20;      // maximum n, inclusive
      int sum = fnMinus1 + fnMinus2;  // Need sum to compute average
      double average;

      System.out.println("The first " + nMax + " Fibonacci numbers are:");
      ......

      while (n <= nMax) {  // n starts from 3
            // n = 3, 4, 5, ..., nMax
         // Compute F(n), print it and add to sum
         ......
         // Increment the index n and shift the numbers for the next iteration
         ++n;
         fnMinus2 = fnMinus1;
         fnMinus1 = fn;
      }

      // Compute and display the average (=sum/nMax).
      // Beware that int/int gives int.
      ......
   }
}
```

Try

1. *Tribonacci numbers* are a sequence of numbers `T(n)` similar to *Fibonacci numbers*, except that a number is formed by adding the three previous numbers, i.e., `T(n)=T(n-1)+T(n-2)+T(n-3)`, `T(1)=T(2)=1`, and `T(3)=2`. Write a program called **`Tribonacci`** to produce the first twenty Tribonacci numbers.

3.7 ExtractDigits (Decision & Loop)

Write a program called **ExtractDigits** to extract each digit from an int, in the reverse order. For example, if the int is 15423, the output shall be "3 2 4 5 1", with a space separating the digits.

Hints

The *coding pattern* for extracting individual digits from an integer *n* is:

1. Use (n % 10) to extract the last (least-significant) digit.
2. Use n = n / 10 to drop the last (least-significant) digit.
3. Repeat if (n > 0), i.e., more digits to extract.

Take note that *n* is destroyed in the process. You may need to clone a copy.

```
    int n = ...;
      while (n > 0) {
         int digit = n % 10;  // Extract the least-significant digit
         // Print this digit
         ......
         n = n / 10;  // Drop the least-significant digit and repeat the loop
      }
```

4. Exercises on Input, Decision and Loop

4.1 Add2Integer (Input)

Write a program called Add2Integers that prompts user to enter two integers. The program shall read the two integers as int; compute their sum; and print the result. For example,

```
Enter first integer: 8
Enter second integer: 9
The sum is: 17
```

Hints

```
import java.util.Scanner;   // For keyboard input
/**
 * 1. Prompt user for 2 integers
 * 2. Read inputs as "int"
 * 3. Compute their sum in "int"
 * 4. Print the result
 */
public class Add2Integers {  // Save as "Add2Integers.java"
   public static void main (String[] args) {
      // Declare variables
      int number1, number2, sum;

      // Put up prompting messages and read inputs as "int"
      Scanner in = new Scanner(System.in);  // Scan the keyboard for input
      System.out.print("Enter first integer: ");    // No newline for prompting
message
      number1 = in.nextInt();                     // Read next input as "int"
      ......
      in.close();  // Close Scanner

      // Compute sum
      sum = ......

      // Display result
      System.out.println("The sum is: " + sum);   // Print with newline
   }
}
```

4.2 SumProductMinMax3 (Arithmetic & Min/Max)

Write a program called `SumProductMinMax3` that prompts user for three integers. The program shall read the inputs as `int`; compute the sum, product, minimum and maximum of the three integers; and print the results. For example,

```
Enter 1st integer: 8
Enter 2nd integer: 2
Enter 3rd integer: 9
The sum is: 19
The product is: 144
The min is: 2
The max is: 9
```

Hints

```
    // Declare variables
      int number1, number2, number3;  // The 3 input integers
      int sum, product, min, max;     // To compute these

      // Prompt and read inputs as "int"
      Scanner in = new Scanner(System.in);  // Scan the keyboard
      ......
      ......
      in.close();

      // Compute sum and product
      sum = ......
      product = ......

      // Compute min
      // The "coding pattern" for computing min is:
      // 1. Set min to the first item
      // 2. Compare current min with the second item and update min if second item
is smaller
      // 3. Repeat for the next item
      min = number1;        // Assume min is the 1st item
      if (number2 < min) {  // Check if the 2nd item is smaller than current min
         min = number2;     // Update min if so
      }
      if (number3 < min) {  // Continue for the next item
         min = number3;
      }

      // Compute max - similar to min
      ......

      // Print results
      ......
```

Try

1. Write a program called SumProductMinMax5 that prompts user for five integers. The program shall read the inputs as int; compute the sum, product, minimum and maximum of the five integers; and print the results. Use five int variables: number1, number2, ..., number5 to store the inputs.

4.3 CircleComputation (double & printf())

Write a program called **CircleComputation** that prompts user for the radius of a circle in floating point number. The program shall read the input as double; compute the diameter, circumference, and area of the circle in double; and print the values rounded to 2 decimal places. Use System-provided constant Math.PI for pi. The formulas are:

```
diameter = 2.0 * radius;
area = Math.PI * radius * radius;
circumference = 2.0 * Math.PI * radius;
```

Hints

```
   // Declare variables
    double radius, diameter, circumference, area;  // inputs and results - all in
double
    ......

    // Prompt and read inputs as "double"
    System.out.print("Enter the radius: ");
    radius = in.nextDouble();  // read input as double

    // Compute in "double"
    ......

    // Print results using printf() with the following format specifiers:
    //   %.2f for a double with 2 decimal digits
    //   %n for a newline
    System.out.printf("Diameter is: %.2f%n", diameter);
    ......
```

Try

1. Write a program called **SphereComputation** that prompts user for the radius of a sphere in floating point number. The program shall read the input as double; compute the volume and surface area of the sphere in double; and print the values rounded to 2 decimal places. The formulas are:

```
surfaceArea = 4 * Math.PI * radius * radius;
volume = 4 /3 * Math.PI * radius * radius * radius;   // But this does not work in
programming?! Why?
```

Take note that you cannot name the variable surface area with a space or surface-area with a dash. Java's naming convention is surfaceArea. Other languages recommend surface_area with an underscore.

2. Write a program called **CylinderComputation** that prompts user for the base radius and height of a cylinder in floating point number. The program shall read the inputs as double; compute the base area, surface area, and volume of the cylinder; and print the values rounded to 2 decimal places. The formulas are:

```
baseArea = Math.PI * radius * radius;
surfaceArea = 2.0 * Math.PI * radius + 2.0 * baseArea;
volume = baseArea * height;
```

4.4 Swap2Integers

Write a program called `Swap2Integers` that prompts user for two integers. The program shall read the inputs as `int`, save in two variables called `number1` and `number2`; swap the contents of the two variables; and print the results. For examples,

```
Enter first integer: 9
Enter second integer: -9
After the swap, first integer is: -9, second integer is: 9
```

Hints

To swap the contents of two variables x and y, you need to introduce a temporary storage, say `temp`, and do: `temp` ⇐ x; x ⇐ y; y ⇐ `temp`.

4.5 IncomeTaxCalculator (Decision)

The progressive income tax rate is mandated as follows:

Taxable Income	Rate (%)
First $20,000	0
Next $20,000	10
Next $20,000	20
The remaining	30

For example, suppose that the taxable income is `$85000`, the income tax payable is `$20000*0% + $20000*10% + $20000*20% + $25000*30%`.

Write a program called **`IncomeTaxCalculator`** that reads the taxable income (in `int`). The program shall calculate the income tax payable (in `double`); and print the result rounded to 2 decimal places. For examples,

```
Enter the taxable income: $41234
The income tax payable is: $2246.80

Enter the taxable income: $67891
The income tax payable is: $8367.30

Enter the taxable income: $85432
The income tax payable is: $13629.60

Enter the taxable income: $12345
The income tax payable is: $0.00
```

Hints

```
      // Declare constants first (variables may use these constants)
      // The keyword "final" marked these as constant (i.e., cannot be changed).
      // Use uppercase words joined with underscore to name constants
      final double TAX_RATE_ABOVE_20K = 0.1;
      final double TAX_RATE_ABOVE_40K = 0.2;
      final double TAX_RATE_ABOVE_60K = 0.3;

      // Declare variables
      int taxableIncome;
      double taxPayable;
      ......

      // Compute tax payable in "double" using a nested-if to handle 4 cases
      if (taxableIncome <= 20000) {         // [0, 20000]
         taxPayable = ......;
      } else if (taxableIncome <= 40000) {  // [20001, 40000]
         taxPayable = ......;
      } else if (taxableIncome <= 60000) {  // [40001, 60000]
         taxPayable = ......;
      } else {                              // [60001, ]
         taxPayable = ......;
      }
      // Alternatively, you could use the following nested-if conditions
      // but the above follows the table data
      //if (taxableIncome > 60000) {          // [60001, ]
      //   ......
      //} else if (taxableIncome > 40000) {   // [40001, 60000]
      //   ......
      //} else if (taxableIncome > 20000) {   // [20001, 40000]
      //   ......
      //} else {                              // [0, 20000]
      //   ......
      //}

      // Print results rounded to 2 decimal places
      System.out.printf("The income tax payable is: $%.2f%n", ...);
```

Try

Suppose that a 10% tax rebate is announced for the income tax payable, capped at $1,000, modify your program to handle the tax rebate. For example, suppose that the tax payable is $12,000, the rebate is $1,000, as 10% of $12,000 exceed the cap.

4.6 IncomeTaxCalculatorWithSentinel (Decision & Loop)

Based on the previous exercise, write a program called IncomeTaxCalculatorWithSentinel which shall repeat the calculation until user enter -1. For example,

```
Enter the taxable income (or -1 to end): $41000
The income tax payable is: $2200.00

Enter the taxable income (or -1 to end): $62000
The income tax payable is: $6600.00

Enter the taxable income (or -1 to end): $73123
The income tax payable is: $9936.90

Enter the taxable income (or -1 to end): $84328
The income tax payable is: $13298.40

Enter the taxable income: $-1
bye!
```

The `-1` is known as the *sentinel value*. (Wiki: In programming, a *sentinel value*, also referred to as a flag value, trip value, rogue value, signal value, or dummy data, is a special value which uses its presence as a condition of termination.)

Hints

The *coding pattern* for handling input with sentinel value is as follows:

```
      // Declare constants first
      final int SENTINEL = -1;    // Terminating value for input
      ......
      // Declare variables
      int taxableIncome;
      double taxPayable;
      ......

      // Read the first input to "seed" the while loop
      System.out.print("Enter the taxable income (or -1 to end): $");
      taxableIncome = in.nextInt();

      while (taxableIncome != SENTINEL) {
         // Compute tax payable
         ......
         // Print result
         ......

         // Read the next input
         System.out.print("Enter the taxable income (or -1 to end): $");
         taxableIncome = in.nextInt();
            // Repeat the loop body, only if the input is not the SENTINEL value.
            // Take note that you need to repeat these two statements inside/outside
the loop!
      }
      System.out.println("bye!");
```

Take note that we repeat the input statements inside and outside the loop. Repeating statements is NOT a good programming practice. This is because it is easy to repeat (Cntl-C/Cntl-V), but hard to maintain and synchronize the repeated statements. In this case, we have no better choices!

4.7 PensionContributionCalculatorWithSentinel (Decision & Loop)

Based on the previous `PensionContributionCalculator`,
write a program called **`PensionContributionCalculatorWithSentinel`** which shall repeat the calculations until user enter -1 for the salary. For examples,

```
Enter the monthly salary (or -1 to end): $5123
Enter the age: 21
The employee's contribution is: $1024.60
The employer's contribution is: $870.91
The total contribution is: $1895.51

Enter the monthly salary (or -1 to end): $5123
Enter the age: 64
The employee's contribution is: $384.22
The employer's contribution is: $461.07
The total contribution is: $845.30

Enter the monthly salary (or -1 to end): $-1
bye!
```

Hints

```
      // Read the first input to "seed" the while loop
      System.out.print("Enter the monthly salary (or -1 to end): $");
      salary = in.nextInt();

      while (salary != SENTINEL) {
         // Read the remaining
         System.out.print("Enter the age: ");
         age = in.nextInt();

         ......
         ......

         // Read the next input and repeat
         System.out.print("Enter the monthly salary (or -1 to end): $");
         salary = in.nextInt();
      }
```

4.8 SalesTaxCalculator (Decision & Loop)

A sales tax of 7% is levied on all goods and services consumed. It is also mandatory that all the price tags should include the sales tax. For example, if an item has a price tag of $107, the actual price is $100 and $7 goes to the sales tax.

Write a program using a loop to continuously input the tax-inclusive price (in double); compute the actual price and the sales tax (in double); and print the results rounded to 2 decimal places. The program shall terminate in response to input of -1; and print the total price, total actual price, and total sales tax. For examples,

```
Enter the tax-inclusive price in dollars (or -1 to end): 107
Actual Price is: $100.00, Sales Tax is: $7.00

Enter the tax-inclusive price in dollars (or -1 to end): 214
Actual Price is: $200.00, Sales Tax is: $14.00

Enter the tax-inclusive price in dollars (or -1 to end): 321
Actual Price is: $300.00, Sales Tax is: $21.00

Enter the tax-inclusive price in dollars (or -1 to end): -1
Total Price is: $642.00
Total Actual Price is: $600.00
Total Sales Tax is: $42.00
```

Hints

```
      // Declare constants
      final double SALES_TAX_RATE = 0.07;
      final int SENTINEL = -1;        // Terminating value for input

      // Declare variables
      double price, actualPrice, salesTax;  // inputs and results
      double totalPrice = 0.0, totalActualPrice = 0.0, totalSalesTax = 0.0;   // to
accumulate
      ......

      // Read the first input to "seed" the while loop
      System.out.print("Enter the tax-inclusive price in dollars (or -1 to end): ");
      price =  in.nextDouble();

      while (price != SENTINEL) {
         // Compute the tax
         ......
         // Accumulate into the totals
         ......
         // Print results
         ......

         // Read the next input and repeat
         System.out.print("Enter the tax-inclusive price in dollars (or -1 to end):
");
         price =  in.nextDouble();
      }
      // print totals
      ......
```

4.9 `ReverseInt` (Loop with Modulus/Divide)

Write a program that prompts user for a positive integer. The program shall read the input as `int`; and print the "reverse" of the input integer. For examples,

```
Enter a positive integer: 12345
The reverse is: 54321
```

Hints

Use the following *coding pattern* which uses a while-loop with repeated modulus/divide operations to extract and drop the last digit of a positive integer.

```
      // Declare variables
      int inNumber;   // to be input
      int inDigit;    // each digit
      ......

      // Extract and drop the "last" digit repeatably using a while-loop with
modulus/divide operations
      while (inNumber > 0) {
         inDigit = inNumber % 10; // extract the "last" digit
         // Print this digit (which is extracted in reverse order)
         ......
         inNumber /= 10;          // drop "last" digit and repeat
      }
      ......
```

4.10 `SumOfDigitsInt` (Loop with Modulus/Divide)

Write a program that prompts user for a positive integer. The program shall read the input as `int`; compute and print the sum of all its digits. For examples,

```
Enter a positive integer: 12345
The sum of all digits is: 15
```

Hints

See "`ReverseInt`".

4.11 InputValidation (Loop with boolean flag)

Your program often needs to validate the user's inputs, e.g., marks shall be between 0 and 100.

Write a program that prompts user for an integer between `0-10` or `90-100`. The program shall read the input as `int`; and repeat until the user enters a valid input. For examples,

```
Enter a number between 0-10 or 90-100: -1
Invalid input, try again...
Enter a number between 0-10 or 90-100: 50
Invalid input, try again...
Enter a number between 0-10 or 90-100: 101
Invalid input, try again...
Enter a number between 0-10 or 90-100: 95
You have entered: 95
```

Hints

Use the following *coding pattern* which uses a do-while loop controlled by a boolean flag to do input validation. We use a do-while instead of while-do loop as we need to execute the body to prompt and process the input at least once.

```
      // Declare variables
      int numberIn;      // to be input
      boolean isValid;   // boolean flag to control the loop
      ......

      // Use a do-while loop controlled by a boolean flag
      // to repeatably read the input until a valid input is entered
      isValid = false;   // default assuming input is not valid
      do {
         // Prompt and read input
         ......

         // Validate input by setting the boolean flag accordingly
         if (numberIn ......) {
            isValid = true;   // exit the loop
         } else {
            System.out.println(......);  // Print error message and repeat
         }
      } while (!isValid);
      ......
```

4.12 AverageWithInputValidation (Loop with boolean flag)

Write a program that prompts user for the mark (between `0-100` in `int`) of 3 students; computes the average (in double); and prints the result rounded to 2 decimal places. Your program needs to perform input validation. For examples,

```
Enter the mark (0-100) for student 1: 56
Enter the mark (0-100) for student 2: 101
Invalid input, try again...
Enter the mark (0-100) for student 2: -1
Invalid input, try again...
Enter the mark (0-100) for student 2: 99
Enter the mark (0-100) for student 3: 45
The average is: 66.67
```

Hints

```
      // Declare constant
      final int NUM_STUDENTS = 3;

      // Declare variables
      int numberIn;
      boolean isValid;   // boolean flag to control the input validation loop
      int sum = 0;
      double average;
      ......

      for (int studentNo = 1; studentNo <= NUM_STUDENTS; ++studentNo) {
          // Prompt user for mark with input validation
          ......
          isValid = false;   // reset assuming input is not valid
          do {
             ......
          } while (!isValid);

          sum += ......;
      }
      ......
```

5. Exercises on Nested-Loops

5.1 SquarePattern (nested-loop)

Write a program called **SquarePattern** that prompts user for the size (a non-negative integer in `int`); and prints the following square pattern using two nested for-loops.

```
Enter the size: 5
# # # # #
# # # # #
# # # # #
# # # # #
# # # # #
```

Hints

The *code pattern* for printing 2D patterns using nested loops is:

```
      // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            System.out.print( ...... );    // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

Notes

1. You should name the loop indexes `row` and `col`, NOT `i` and `j`, or `x` and `y`, or `a` and `b`, which are meaningless.

2. The `row` and `col` could start at 1 (and upto `size`), or start at `0` (and upto `size-1`). As computer counts from `0`, it is probably more efficient to start from `0`. However, since humans counts from `1`, it is easier to read if you start from `1`.

Try

Rewrite the above program using nested while-do loops.

5.2 CheckerPattern (nested-loop)

Write a program called **`CheckerPattern`** that prompts user for the size (a non-negative integer in `int`); and prints the following checkerboard pattern.

```
Enter the size: 7
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
 # # # # # # #
# # # # # # #
```

Hints

```
 // Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if ((row % 2) == 0) {   // row 2, 4, 6, ...
               ......
            }
            System.out.print( ...... );    // Use print() without newline inside the
inner loop
            ......
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

5.3 TimeTable (nested-loop)

Write a program called **`TimeTable`** that prompts user for the size (a positive integer in `int`); and prints the multiplication table as shown:

```
Enter the size: 10
 * |   1   2   3   4   5   6   7   8   9  10
--------------------------------------------
 1 |   1   2   3   4   5   6   7   8   9  10
 2 |   2   4   6   8  10  12  14  16  18  20
 3 |   3   6   9  12  15  18  21  24  27  30
 4 |   4   8  12  16  20  24  28  32  36  40
 5 |   5  10  15  20  25  30  35  40  45  50
 6 |   6  12  18  24  30  36  42  48  54  60
 7 |   7  14  21  28  35  42  49  56  63  70
 8 |   8  16  24  32  40  48  56  64  72  80
 9 |   9  18  27  36  45  54  63  72  81  90
10 |  10  20  30  40  50  60  70  80  90 100
```

Hints

1. Use `printf()` to format the output, e.g., each cell is `%4d`.
2. See "Java Basics" article.

5.4 TriangularPattern (nested-loop)

Write 4 programs called **`TriangularPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints each of the patterns as shown:

```
Enter the size: 8

#                    # # # # # # # #      # # # # # # # #                    #
# #                  # # # # # # #          # # # # # # #                  # #
# # #                # # # # # #              # # # # # #                # # #
# # # #              # # # # #                  # # # # #              # # # #
# # # # #            # # # #                      # # # #            # # # # #
# # # # # #          # # #                          # # #          # # # # # #
# # # # # # #        # #                              # #        # # # # # # #
# # # # # # # #      #                                  #      # # # # # # # #
     (a)                   (b)                  (c)                  (d)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. You need to print the *leading* blanks, in order to push the # to the right. The *trailing* blanks are optional, which does not affect the pattern.
3. For pattern (a), `if (row >= col) print #`. Trailing blanks are optional.
4. For pattern (b), `if (row + col <= size + 1) print #`. Trailing blanks are optional.
5. For pattern (c), `if (row >= col) print #; else print blank`. Need to print the *leading* blanks.
6. For pattern (d), `if (row + col >= size + 1) print #; else print blank`. Need to print the *leading* blanks.
7. The *coding pattern* is:

```
// Outer loop to print each of the rows
      for (int row = 1; row <= size; row++) {  // row = 1, 2, 3, ..., size
         // Inner loop to print each of the columns of a particular row
         for (int col = 1; col <= size; col++) {  // col = 1, 2, 3, ..., size
            if (......) {
                System.out.print("# ");
            } else {
                System.out.print("  ");  // Need to print the "leading" blanks
            }
         }
         // Print a newline after printing all the columns
         System.out.println();
      }
```

5.5 BoxPattern (nested-loop)

Write 4 programs called **`BoxPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8
```

```
# # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #
#           #     #                       #       #       #     # #       # #
#           #       #                   #           #   #       #   #   #   #
#           #         #               #               #         #     #     #
#           #           #           #               #   #       #   #   #   #
#           #             #       #               #       #     # #       # #
# # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #   # # # # # # #
     (a)             (b)             (c)             (d)             (e)
```

Hints

1. On the main diagonal, `row = col`. On the opposite diagonal, `row + col = size + 1`, where `row` and `col` begin from 1.
2. For pattern (a), `if (row == 1 || row == size || col == 1 || col == size) print #; else print blank`. Need to print the intermediate blanks.
3. For pattern (b), `if (row == 1 || row == size || row == col) print #; else print blank`.

5.6 HillPattern (nested-loop)

Write 3 programs called **`HillPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the rows: 6

          #                # # # # # # # # # # #                #                # # # # # # # # # # #
        # # #                # # # # # # # # #                # # #              # # # # #   # # # # #
      # # # # #                # # # # # # #                # # # # #            # # # #       # # # #
    # # # # # # #                # # # # #                # # # # # # #          # # #           # # #
  # # # # # # # # #                # # #                # # # # # # # # #        # #               # #
# # # # # # # # # # #                #                # # # # # # # # # # #      #                   #
         (a)                        (b)                 # # # # # # # # #        # #               # #
                                                          # # # # # # #          # # #           # # #
                                                            # # # # #            # # # #       # # # #
                                                              # # #              # # # # #   # # # # #
                                                                #                # # # # # # # # # # #
                                                               (c)                        (d)
```

Hints

For pattern (a):

```
for (int row = 1; ......) {
         // numCol = 2*numRows - 1
         for (int col = 1; ......) {
            if ((row + col >= numRows + 1) && (row >= col - numRows + 1)) {
               ......;
            } else {
               ......;
            }
         }
         ......;
      }
```

or, use 2 sequential inner loops to print the columns:

```
      for (int row = 1; row <= rows; row++) {
         for (int col = 1; col <= rows; col++) {
            if ((row + col >= rows + 1)) {
               ......
            } else {
               ......
            }
         }
         for (int col = 2; col <= rows; col++) {  // skip col = 1
            if (row >= col) {
               ......
            } else {
               ......
            }
         }
         ......
      }
```

5.7 NumberPattern (nested-loop)

Write 4 programs called **`NumberPatternX`** (`X = A, B, C, D`) that prompts user for the size (a non-negative integer in `int`); and prints the pattern as shown:

```
Enter the size: 8

1                    1 2 3 4 5 6 7 8                    1      8 7 6 5 4 3 2 1
1 2                    1 2 3 4 5 6 7                  2 1      7 6 5 4 3 2 1
1 2 3                    1 2 3 4 5 6                3 2 1      6 5 4 3 2 1
1 2 3 4                    1 2 3 4 5              4 3 2 1      5 4 3 2 1
1 2 3 4 5                    1 2 3 4            5 4 3 2 1      4 3 2 1
1 2 3 4 5 6                    1 2 3          6 5 4 3 2 1      3 2 1
1 2 3 4 5 6 7                    1 2        7 6 5 4 3 2 1      2 1
1 2 3 4 5 6 7 8                    1      8 7 6 5 4 3 2 1      1
     (a)                  (b)                    (c)                (d)
```

6. Magic(Special) Numbers

6.1. Amicable umbers

Two different numbers are said to be so Amicable numbers if each sum of divisors is equal to the other number. Amicalble Numbers are: (220, 284), (1184, 1210), (2620, 2924), (5020, 5564), (6232, 6368). For example,

```
Enter 1st number: 228
Enter 2nd number: 220
The numbers are Amicable Numbers.
```

Hints

220 and 284 are Amicable Numbers.

Divisors of 220 = 1, 2, 4, 5, 10, 11, 20, 22, 44, 55, 110

1+2+4+5+10+11+20+22+44+55+110 = 284

Divisors of 284 = 1, 2, 4, 71, 142

```
1+2+4+71+142 = 220
```

6.2. Armstrong Number

Armstrong number is a positive number if it is equal to the sum of cubes of its digits is called Armstrong number and if its sum is not equal to the number then it's not a Armstrong number. For example,

```
Enter number=145
145 is not an Armstrong Number

Enter number = 153
153 is an Armstrong Number
```

Hints

```
Examples: 153 is Armstrong

(1*1*1)+(5*5*5)+(3*3*3) = 153
```

6.3. Capricorn Number

A number is called Capricorn or Kaprekar number whose square is divided into two parts in any conditions and parts are added, the additions of parts is equal to the number, is called Capricorn or Kaprekar number. For example,

```
Enter a number : 45
45 is a Capricorn/Kaprekar number
Enter a number : 297
297 is a Capricorn/Kaprekar number
Enter a number : 44
44 is not a Capricorn/Kaprekar number
```

Hints

```
Number = 45
(45)2  = 2025

All parts for 2025:
202    + 5   =    207 (not 45)
20     + 25  =    45
2+  025    =    27 (not 45)
```

From the above we can see one combination is equal to number so that 45 is Capricorn or Kaprekar number.

Try

Write a Java program to generate and show all Kaprekar numbers less than 1000.

6.4. Circular Prime

A circular prime is a prime number with the property that the number generated at each intermediate step when cyclically permuting its digits will be prime. For example, 1193 is a circular prime, since 1931, 9311 and 3119 all are also prime. For example,

```
Enter a number : 137
137 is a Circular Prime
Enter a number : 44
44 is not a Circular Prime
```

6.5. Happy Number

A happy number is a natural number in a given number base that eventually reaches 1 when iterated over the perfect digital invariant function for. Those numbers that do not end in 1 are -unhappy numbers. For example,

```
Enter a number : 31
31 is a Happy number

Enter a number : 32
32 is not a Happy number
```

6.6. Automorphic Number

An Automorphic number is a number whose square "ends" in the same digits as the number itself. For example,

```
Enter a number : 5
5 is a Automorphic Number

Enter a number : 25
25 is a Automorphic Number

Enter a number : 2
2 is not a Automorphic Number
```

Hints

5*5 = 25, 6*6 = 36, 25*25 = 625

5,6,25 are automorphic numbers

6.7. Disarium Number

A number is called Disarium number if the sum of its power of the positions from left to right is equal to the number. For example,

```
Enter a number : 135
135 is a Disarium Number

Enter a number : 32
32 is not a Disarium Number
```

Hints

$1^1 + 3^2 + 5^3 = 1 + 9 + 125 = 135$

6.8. Magic Number

Magic number is the if the sum of its digits recursively are calculated till a single digit If the single digit is 1 then the number is a magic number. Magic number is very similar with Happy Number. For example,

```
Enter a number : 226
226 is a Magic Number

Enter a number : 32
32 is not a Magic Number
Enter number = 541
153 is a Magic Number
```

Hints

226 is said to be a magic number

2+2+6=10 sum of digits is 10 then again 1+0=1 now we get a single digit number is 1.if we single digit number will now 1 them it would not a magic number.

6.9. Neon Number

A neon number is a number where the sum of digits of square of the number is equal to the number. For example if the input number is 9, its square is 9*9 = 81 and sum of the digits is 9. i.e. 9 is a neon number. For example,

```
Enter a number: 9
9 is a Neon Number

Enter a number: 8
8 is not a Neon Number
```

6.10. Palindromic Number

A palindromic number is a number that remains the same when its digits are reversed. For example,

```
Enter a number : 16461
16461 is a Palendromic Number

Enter a number : 1234
1234 is not a Plaindromic Number
```

6.11. Perfect Number

A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. For instance, 6 has divisors 1, 2 and 3, and 1 + 2 + 3 = 6, so 6 is a perfect number. For example,

```
Enter a number : 6
6 is a Perfect Number

Enter a number : 3
3 is not a Perfect Number
```

6.12. Special Number

A number is said to be special number when the sum of factorial of its digits is equal to the number itself. Example- 145 is a Special Number as 1!+4!+5!=145. For example,

```
Enter a number : 145
145 is a Special Number

Enter a number : 23
23 is not a Special Number
```

6.13. Spy Number

A spy number is a number where the sum of its digits equals the product of its digits. For example, 1124 is a spy number, the sum of its digits is 1+1+2+4=8 and the product of its digits is 1*1*2*4=8. For example,

```
Enter a number : 1124
1124 is a Spy Number

Enter a number : 12
12 is not a Spy Number
```

6.14. Ugly Number

A number is said to be an Ugly number if positive numbers whose prime factors only include 2, 3, 5. For example, 6(2×3), 8(2x2x2), 15(3×5) are ugly numbers while 14(2×7) is not ugly since it includes another prime factor 7. Note that 1 is typically treated as an ugly number. For example,

```
Enter a number : 6
6 is an Ugly Number

Enter a number : 14
14 is not an Ugly Number
```

7. Exercises on `String` and `char` Operations

7.1 `ReverseString` (`String` & `char`)

Write a program called **`ReverseString`**, which prompts user for a `String`, and prints the *reverse* of the `String` by extracting and processing each character. The output shall look like:

```
Enter a String: abcdef
The reverse of the String "abcdef" is "fedcba".
```

Hints

For a `String` called `inStr`, you can use `inStr.length()` to get the *length* of the `String`; and `inStr.charAt(idx)` to retrieve the `char` at the `idx` position, where `idx` begins at 0, up to `instr.length() - 1`.

```
      // Define variables
      String inStr;        // input String
      int inStrLen;        // length of the input String
      ......
      // Prompt and read input as "String"
      System.out.print("Enter a String: ");
      inStr = in.next();   // use next() to read a String
      inStrLen = inStr.length();

      // Use inStr.charAt(index) in a loop to extract each character
      // The String's index begins at 0 from the left.
      // Process the String from the right
      for (int charIdx = inStrLen - 1; charIdx >= 0; --charIdx) {
            // charIdx = inStrLen-1, inStrLen-2, ... ,0
         ......
      }
```

7.2 CountVowelsDigits (String & char)

Write a program called **CountVowelsDigits**, which prompts the user for a String, counts the number of vowels (a, e, i, o, u, A, E, I, O, U) and digits (0-9) contained in the string, and prints the counts and the percentages (rounded to 2 decimal places). For example,

```
Enter a String: testing12345
Number of vowels: 2 (16.67%)
Number of digits: 5 (41.67%)
```

Hints

1. To check if a char c is a digit, you can use boolean expression (c >= '0' && c <= '9'); or use built-in boolean function Character.isDigit(c).
2. You could use in.next().toLowerCase() to convert the input String to lowercase to reduce the number of cases.
3. To print a % using printf(), you need to use %%. This is because % is a prefix for format specifier in printf(), e.g., %d and %f.

7.3 PhoneKeyPad (String & char)

On you phone keypad, the alphabets are mapped to digits as follows:

ABC(2), DEF(3), GHI(4), JKL(5), MNO(6), PQRS(7), TUV(8), WXYZ(9).

Write a program called **PhoneKeyPad**, which prompts user for a String (case insensitive), and converts to a sequence of keypad digits. Use (a) a nested-if, (b) a switch-case-default.

Hints

1. You can use in.next().toLowerCase() to read a String and convert it to lowercase to reduce your cases.
2. In switch-case, you can handle multiple cases by omitting the break statement, e.g.,

```
switch (inChar) {
   case 'a': case 'b': case 'c':  // No break for 'a' and 'b', fall thru 'c'
      System.out.print(2); break;
   case 'd': case 'e': case 'f':
      ......
   default:
      ......
}
```

7.4 Caesar's Code (String & char)

Caesar's Code is one of the simplest encryption techniques. Each letter in the plaintext is replaced by a letter some fixed number of position (n) down the alphabet cyclically. In this exercise, we shall pick `n=3`. That is, `'A'` is replaced by `'D'`, `'B'` by `'E'`, `'C'` by `'F'`, ..., `'X'` by `'A'`, ..., `'Z'` by `'C'`.

Write a program called **`CaesarCode`** to cipher the Caesar's code. The program shall prompt user for a plaintext string consisting of mix-case letters only; compute the ciphertext; and print the ciphertext in uppercase. For example,

```
Enter a plaintext string: Testing
The ciphertext string is: WHVWLQJ
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases (`'A'`-`'Z'`). But it is much better to consider `'A'` to `'W'` as one case; `'X'`, `'Y'` and `'Z'` as 3 separate cases.
3. Take note that `char 'A'` is represented as Unicode number `65` and `char 'D'` as `68`. However, `'A' + 3 gives 68`. This is because `char + int` is implicitly casted to `int + int` which returns an `int` value. To obtain a `char` value, you need to perform explicit type casting using `(char)('A' + 3)`. Try printing `('A' + 3)` with and without type casting.

7.5 Decipher Caesar's Code (String & char)

Write a program called **`DecipherCaesarCode`** to decipher the Caesar's code described in the previous exercise. The program shall prompts user for a ciphertext string consisting of mix-case letters only; compute the plaintext; and print the plaintext in uppercase. For example,

```
Enter a ciphertext string: wHVwLQJ
The plaintext string is: TESTING
```

7.6 Exchange Cipher (String & char)

This simple cipher exchanges `'A'` and `'Z'`, `'B'` and `'Y'`, `'C'` and `'X'`, and so on.

Write a program called **`ExchangeCipher`** that prompts user for a plaintext string consisting of mix-case letters only. You program shall compute the ciphertext; and print the ciphertext in uppercase. For examples,

```
Enter a plaintext string: abcXYZ
The ciphertext string is: ZYXCBA
```

Hints

1. Use `in.next().toUpperCase()` to read an input string and convert it into uppercase to reduce the number of cases.
2. You can use a big nested-if with 26 cases (`'A'`-`'Z'`), or use the following relationship:

```
'A' + 'Z' == 'B' + 'Y' == 'C' + 'X' == ... == plainTextChar + cipherTextChar
Hence, cipherTextChar = 'A' + 'Z' - plainTextChar
```

7.7 TestPalindromicWord and TestPalindromicPhrase (String & char)

A word that reads the same backward as forward is called a *palindrome*, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive).

Write a program called **TestPalindromicWord**, that prompts user for a word and prints ""xxx" is|is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization).

Modify your program (called **TestPalindromicPhrase**) to check for palindromic phrase. Use in.nextLine() to read a line of input.

Hints

1. Maintain two indexes, `forwardIndex` (`fIdx`) and `backwardIndex` (`bIdx`), to scan the phrase forward and backward.

```
int fIdx = 0, bIdx = strLen - 1;
while (fIdx < bIdx) {
   ......
   ++fIdx;
   --bIdx;
}
// or
for (int fIdx = 0, bIdx = strLen - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   ......
}
```

2. You can check if a `char` `c` is a letter either using built-in `boolean` function `Character.isLetter(c)`; or boolean expression `(c >= 'a' && c <= 'z')`. Skip the index if it does not contain a letter.

7.8 CheckBinStr (String & char)

The binary number system uses 2 symbols, 0 and 1. Write a program called **`CheckBinStr`** to verify a binary string. The program shall prompt user for a binary string; and decide if the input string is a valid binary string. For example,

```
Enter a binary string: 10101100
"10101100" is a binary string

Enter a binary string: 10120000
"10120000" is NOT a binary string
```

Hints

Use the following coding pattern which involves a boolean flag to check the input string.

```
// Declare variables
      String inStr;     // The input string
      int inStrLen;     // The length of the input string
      char inChar;      // Each char of the input string
      boolean isValid;  // "is" or "is not" a valid binary string?
      ......

      isValid = true;  // Assume that the input is valid, unless our check fails
      for (......) {
         inChar = ......;
         if (!(inChar == '0' || inChar == '1')) {
            isValid = false;
            break;  // break the loop upon first error, no need to continue for more
errors
                    // If this is not encountered, isValid remains true after the
loop.
         }
      }
      if (isValid) {
         System.out.println(......);
      } else {
         System.out.println(......);
      }
      // or using one liner
      //System.out.println(isValid ? ... : ...);
```

7.9 CheckHexStr (String & char)

The hexadecimal (hex) number system uses 16 symbols, `0-9` and `A-F` (or `a-f`). Write a program to verify a hex string. The program shall prompt user for a hex string; and decide if the input string is a valid hex string. For examples,

```
Enter a hex string: 123aBc
"123aBc" is a hex string

Enter a hex string: 123aBcx
"123aBcx" is NOT a hex string
```

Hints

```
if (!((inChar >= '0' && inChar <= '9')
      || (inChar >= 'A' && inChar <= 'F')
      || (inChar >= 'a' && inChar <= 'f'))) {   // Use positive logic and then
reverse
   ......
}
```

7.10 Bin2Dec (String & char)

Write a program called **`Bin2Dec`** to convert an input binary string into its equivalent decimal number. Your output shall look like:

```
Enter a Binary string: 1011
The equivalent decimal number for binary "1011" is: 11

Enter a Binary string: 1234
error: invalid binary string "1234"
```

Hints

See "Code Example".

7.11 Hex2Dec (String & char)

Write a program called **`Hex2Dec`** to convert an input hexadecimal string into its equivalent decimal number. Your output shall look like:

```
Enter a Hexadecimal string: 1a
The equivalent decimal number for hexadecimal "1a" is: 26
```

```
Enter a Hexadecimal string: 1y3
error: invalid hexadecimal string "1y3"
```

Hints

See "Code Example".

7.12 Oct2Dec (String & char)

Write a program called **Oct2Dec** to convert an input Octal string into its equivalent decimal number. For example,

```
Enter an Octal string: 147
The equivalent decimal number "147" is: 103
```

8. Exercises on Arrays

8.1 PrintArray (Array)

Write a program called PrintArray which prompts user for the number of items in an array (a non-negative integer), and saves it in an int variable called NUM_ITEMS. It then prompts user for the values of all the items and saves them in an int array called items. The program shall then print the contents of the array in the form of [x1, x2, ..., xn]. For example,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 3 2 5 6 9
The values are: [3, 2, 5, 6, 9]
```

Hints

```
      // Declare variables
      tinal int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......

      // Prompt for for the number of items and read the input as "int"
      ......
      NUM_ITEMS = ......

      // Allocate the array
      items = new int[NUM_ITEMS];

      // Prompt and read the items into the "int" array, if array length > 0
      if (items.length > 0) {
         ......
         for (int i = 0; i < items.length; ++i) {  // Read all items
            ......
         }
      }

      // Print array contents, need to handle first item and subsequent items
differently
      ......
      for (int i = 0; i < items.length; ++i) {
         if (i == 0) {
            // Print the first item without a leading commas
            ......
         } else {
            // Print the subsequent items with a leading commas
            ......
         }
         // or, using a one liner
         //System.out.print((i == 0) ? ...... : ......);
      }
```

8.2 PrintArrayInStars (Array)

Write a program called **printArrayInStars** which prompts user for the number of items in an array (a non-negative integer), and saves it in an int variable called NUM_ITEMS. It then prompts user for the values of all the items (non-negative integers) and saves them in an int array called items. The program shall then print the contents of the array in a graphical form, with the array index and values represented by number of stars. For examples,

```
Enter the number of items: 5
Enter the value of all items (separated by space): 7 4 3 0 7
0: *******(7)
1: ****(4)
2: ***(3)
3: (0)
4: *******(7)
```

Hints

```
      // Declare variables
      final int NUM_ITEMS;
      int[] items;  // Declare array name, to be allocated after NUM_ITEMS is known
      ......
      ......
      // Print array in "index: number of stars" using a nested-loop
      // Take note that rows are the array indexes and columns are the value in that
index
      for (int idx = 0; idx < items.length; ++idx) {  // row
         System.out.print(idx + ": ");
         // Print value as the number of stars
         for (int starNo = 1; starNo <= items[idx]; ++starNo) {  // column
            System.out.print("*");
         }
         ......
      }
      ......
```

8.3 GradesStatistics (Array)

Write a program which prompts user for the number of students in a class (a non-negative integer), and saves it in an int variable called numStudents. It then prompts user for the grade of each of the students (integer between 0 to 100) and saves them in an int array called grades. The program shall then compute and print the average (in double rounded to 2 decimal places) and minimum/maximum (in int).

```
Enter the number of students: 5
Enter the grade for student 1: 98
Enter the grade for student 2: 78
Enter the grade for student 3: 78
Enter the grade for student 4: 87
Enter the grade for student 5: 76
The average is: 83.40
The minimum is: 76
The maximum is: 98
```

8.4 Hex2Bin (Array for Table Lookup)

Write a program called **Hex2Bin** that prompts user for a hexadecimal string and print its equivalent binary string. The output shall look like:

```
Enter a Hexadecimal string: 1abc
The equivalent binary for hexadecimal "1abc" is: 0001 1010 1011 1100
```

Hints

1. Use an array of 16 `Strings` containing binary strings corresponding to hexadecimal number `0-9A-F` (or `a-f`), as follows

```
final String[] HEX_BITS = {"0000", "0001", "0010", "0011",
                           "0100", "0101", "0110", "0111",
                           "1000", "1001", "1010", "1011",
                           "1100", "1101", "1110", "1111"};
```

8.5 Dec2Hex (Array for Table Lookup)

Write a program called **Dec2Hex** that prompts user for a positive decimal number, read as `int`, and print its equivalent hexadecimal string. The output shall look like:

```
Enter a decimal number: 1234
The equivalent hexadecimal number is 4D2
```

9. Exercises on Methods

9.1 exponent() (method)

Write a method called `exponent(int base, int exp)` that returns an `int` value of `base` raises to the power of `exp`. The signature of the method is:

```
  public static int exponent(int base, int exp);
```

Assume that exp is a non-negative integer and base is an integer. Do not use any Math library functions. Also write the `main()` method that prompts user for the base and exp; and prints the result. For example,

```
Enter the base: 3
Enter the exponent: 4
3 raises to the power of 4 is: 81
```

Hints

```
......
public class Exponent {
   public static void main(String[] args) {
      // Declare variables
      int exp;    // exponent (non-negative integer)
      int base;   // base (integer)
      ......
      // Prompt and read exponent and base
      ......
      // Print result
      System.out.println(base + " raises to the power of " + exp + " is: " +
exponent(base, exp));
   }

   // Returns "base" raised to the power "exp"
   public static int exponent(int base, int exp) {
      int product = 1;   // resulting product

      // Multiply product and base for exp number of times
      for (......) {
         product *= base;
      }

      return product;
   }
}
```

9.2 isOdd() (method)

Write a boolean method called `isOdd()` in a class called **`OddEvenTest`**, which takes an `int` as input and returns `true` if the it is odd. The signature of the method is as follows:

```
public static boolean isOdd(int number);
```

Also write the `main()` method that prompts user for a `number`, and prints "ODD" or "EVEN". You should test for negative input. For examples,

```
Enter a number: 9
9 is an odd number

Enter a number: 8
8 is an even number

Enter a number: -5
-5 is an odd number
```

Hints

See Notes.

9.3 hasEight() (method)

Write a boolean method called `hasEight()`, which takes an `int` as input and returns `true` if the number contains the digit `8` (e.g., `18`, `168`, `1288`). The signature of the method is as follows:

```
public static boolean hasEight(int number);
```

Write a program called **`MagicSum`**, which prompts user for integers (or `-1` to end), and produce the sum of numbers containing the digit `8`. Your program should use the above methods. A sample output of the program is as follows:

```
Enter a positive integer (or -1 to end): 1
Enter a positive integer (or -1 to end): 2
Enter a positive integer (or -1 to end): 3
Enter a positive integer (or -1 to end): 8
Enter a positive integer (or -1 to end): 88
Enter a positive integer (or -1 to end): -1
```

```
The magic sum is: 96
```

Hints

The *coding pattern* to repeat until input is `-1` (called *sentinel* value) is:

```
final int SENTINEL = -1;  // Terminating input
int number;

// Read first input to "seed" the while loop
System.out.print("Enter a positive integer (or -1 to end): ");
number = in.nextInt();

while (number != SENTINEL) {  // Repeat until input is -1
   ......
   ......

   // Read next input. Repeat if the input is not the SENTINEL
   // Take note that you need to repeat these codes!
   System.out.print("Enter a positive integer (or -1 to end): ");
   number = in.nextInt();
}
```

You can either repeatably use modulus/divide (`n%10` and `n=n/10`) to extract and drop each digit in `int`; or convert the `int` to `String` and use the `String`'s `charAt()` to inspect each `char`.

9.4 `print()` (Array & Method)

Write a method called **`print()`**, which takes an `int` array and print its contents in the form of `[a1, a2, ..., an]`. Take note that there is no comma after the last element. The method's signature is as follows:

```
public static void print(int[] array);
```

Also write a test driver to test this method (you should test on empty array, one-element array, and n-element array).

How to handle `double[]` or `float[]`? You need to write a overloaded version for `double[]` and a overloaded version for `float[]`, with the following signatures:

```
public static void print(double[] array)
public static void print(float[] array)
```

The above is known as *method overloading*, where the same method name can have many versions, differentiated by its parameter list.

Hints

For the first element, print its value; for subsequent elements, print commas followed by the value.

9.5 `arrayToString()` (Array & Method)

Write a method called **`arrayToString()`**, which takes an `int` array and return a `String` in the form of `[a1, a2, ..., an]`. Take note that this method returns a `String`, the previous exercise returns `void` but prints the output. The method's signature is as follows:

```
public static String arrayToString(int[] array);
```

Also write a test driver to test this method (you should test on empty array, one-element array, and n-element array).

Notes: This is similar to the built-in function `Arrays.toString()`. You could study its source code.

9.6 contains() (Array & Method)

Write a boolean method called **contains()**, which takes an array of int and an int; and returns true if the array contains the given int. The method's signature is as follows:

```
public static boolean contains(int[] array, int key);
```

Also write a test driver to test this method.

9.7 search() (Array & Method)

Write a method called **search()**, which takes an array of int and an int; and returns the array *index* if the array contains the given int; or -1 otherwise. The method's signature is as follows:

```
public static int search(int[] array, int key);
```

Also write a test driver to test this method.

9.8 equals() (Array & Method)

Write a boolean method called **equals()**, which takes two arrays of int and returns true if the two arrays are exactly the same (i.e., same length and same contents). The method's signature is as follows:

```
public static boolean equals(int[] array1, int[] array2)
```

Also write a test driver to test this method.

9.9 copyOf() (Array & Method)

Write a boolean method called **copyOf()**, which takes an int Array and returns a copy of the given array. The method's signature is as follows:

```
public static int[] copyOf(int[] array)
```

Also write a test driver to test this method.

Write another version for **copyOf()** which takes a second parameter to specify the length of the new array. You should truncate or pad with zero so that the new array has the required length.

```
public static int[] copyOf(int[] array, int newLength)
```

NOTES: This is similar to the built-in function Arrays.copyOf().

9.10 swap() (Array & Method)

Write a method called **swap()**, which takes two arrays of int and swap their contents if they have the same length. It shall return true if the contents are successfully swapped. The method's signature is as follows:

```
public static boolean swap(int[] array1, int[] array2)
```

Also write a test driver to test this method.

Hints

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

9.11 reverse() (Array & Method)

Write a method called **reverse()**, which takes an array of int and reverse its contents. For example, the reverse of [1,2,3,4] is [4,3,2,1]. The method's signature is as follows:

```
public static void reverse(int[] array)
```

Take note that the array passed into the method can be modified by the method (this is called "*pass by reference*"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "*pass by value*").

Also write a test driver to test this method.

Hints

You might use two indexes in the loop, one moving forward and one moving backward to point to the two elements to be swapped.

```
for (int fIdx = 0, bIdx = array.length - 1; fIdx < bIdx; ++fIdx, --bIdx) {
   // Swap array[fIdx] and array[bIdx]
   // Only need to transverse half of the array elements
}
```

You need to use a temporary location to swap two storage locations.

```
// Swap item1 and item2
int item1, item2, temp;
temp = item1;
item1 = item2;
item2 = item1;
// You CANNOT simply do: item1 = item2; item2 = item2;
```

9.12 GradesStatistics (Array & Method)

Write a program called **GradesStatistics**, which reads in *n* grades (of int between 0 and 100, inclusive) and displays the *average*, *minimum*, *maximum*, *median* and *standard deviation*. Display the floating-point values upto 2 decimal places. Your output shall look like:

```
Enter the number of students: 4
Enter the grade for student 1: 50
Enter the grade for student 2: 51
Enter the grade for student 3: 56
Enter the grade for student 4: 53
The grades are: [50, 51, 56, 53]
The average is: 52.50
The median is: 52.00
The minimum is: 50
The maximum is: 56
The standard deviation is: 2.29
```

The formula for calculating standard deviation is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} {x_i}^2 - \mu^2}, \text{ where } \mu \text{ is the mean}$$

Hints:

```
public class GradesStatistics {
   public static int[] grades;  // Declare an int[], to be allocated later.
                                // This array is accessible by all the methods.

   public static void main(String[] args) {
      readGrades();  // Read and save the inputs in global int[] grades
      System.out.println("The grades are: ");
      print(grades);
      System.out.println("The average is " + average(grades));
      System.out.println("The median is " + median(grades));
      System.out.println("The minimum is " + min(grades));
      System.out.println("The maximum is " + max(grades));
      System.out.println("The standard deviation is " + stdDev(grades));
   }

   // Prompt user for the number of students and allocate the global "grades" array.
   // Then, prompt user for grade, check for valid grade, and store in "grades".
   public static void readGrades() { ....... }

   // Print the given int array in the form of [x1, x2, x3,..., xn].
   public static void print(int[] array) { ....... }

   // Return the average value of the given int[]
   public static double average(int[] array) { ...... }

   // Return the median value of the given int[]
   // Median is the center element for odd-number array,
   // or average of the two center elements for even-number array.
   // Use Arrays.sort(anArray) to sort anArray in place.
   public static double median(int[] array) { ...... }

   // Return the maximum value of the given int[]
   public static int max(int[] array) {
      int max = array[0];   // Assume that max is the first element
      // From second element, if the element is more than max, set the max to this
element.
      ......
   }

   // Return the minimum value of the given int[]
   public static int min(int[] array) { ....... }

   // Return the standard deviation of the given int[]
   public static double stdDev(int[] array) { ....... }
}
```

Take note that besides readGrade() that relies on global variable grades, all the methods are *self-contained general utilities* that operate on any given array.

9.13 GradesHistogram (Array & Method)

Write a program called **GradesHistogram**, which reads in *n* grades (as in the previous exercise), and displays the horizontal and vertical histograms. For example:

```
 0 -  9: ***
10 - 19: ***
20 - 29:
30 - 39:
40 - 49: *
50 - 59: *****
60 - 69:
70 - 79:
80 - 89: *
90 -100: **

                               *
                               *
 *     *                       *
 *     *                       *                       *
 *     *                 *     *                 *     *
0-9  10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100
```

10. Exercises on Command-line Arguments, Recursion

10.1 Arithmetic (Command-Line Arguments)

Write a program called **Arithmetic** that takes three command-line arguments: two integers followed by an arithmetic operator (+, -, * or /). The program shall perform the corresponding operation on the two integers and print the result. For example:

```
java Arithmetic 3 2 +
3+2=5

java Arithmetic 3 2 -
3-2=1

java Arithmetic 3 2 /
3/2=1
```

Hints

The method `main(String[] args)` takes an argument: "an array of `String`", which is often (but not necessary) named `args`. This parameter captures the command-line arguments supplied by the user when the program is invoked. For example, if a user invokes:

```
java Arithmetic 12345 4567 +
```

The three command-line arguments `"12345"`, `"4567"` and `"+"` will be captured in a `String` array `{"12345", "4567", "+"}` and passed into the `main()` method as the argument `args`. That is,

```
args is: {"12345", "4567", "+"}  // args is a String array
args.length is: 3                // length of the array
args[0] is: "12345"              // 1st element of the String array
args[1] is: "4567"               // 2nd element of the String array
args[2] is: "+"                  // 3rd element of the String array
args[0].length() is: 5           // length of 1st String element
args[1].length() is: 4           // length of the 2nd String element
args[2].length() is: 1           // length of the 3rd String element
```

```
public class Arithmetic {
  public static void main (String[] args) {
    int operand1, operand2;
    char theOperator;

    // Check if there are 3 command-line arguments in the
    //  String[] args by using length variable of array.
    if (args.length != 3) {
      System.err.println("Usage: java Arithmetic int1 int2 op");
      return;
    }

    // Convert the 3 Strings args[0], args[1], args[2] to int and char.
    // Use the Integer.parseInt(aStr) to convert a String to an int.
    operand1 = Integer.parseInt(args[0]);
    operand2 = ......

    // Get the operator, assumed to be the first character of
    //  the 3rd string. Use method charAt() of String.
    theOperator = args[2].charAt(0);
    System.out.print(args[0] + args[2] + args[1] + "=");

    switch(theOperator) {
      case ('-'): System.out.println(operand1 - operand2); break;
      case ('+'): ......
      case ('*'): ......
      case ('/'): ......
      default:
        System.err.println("Error: invalid operator!");
    }
  }
}
```

Notes:

- To provide command-line arguments, use the "cmd" or "terminal" to run your program in the form "`java` *`ClassName arg1 arg2`* `....`".
- To provide command-line arguments in Eclipse, right click the source code ⇒ "Run As" ⇒ "Run Configurations..." ⇒ Select "Main" and choose the proper main class ⇒ Select "Arguments" ⇒ Enter the command-line arguments, e.g., "3 2 +" in "Program Arguments".
- To provide command-line arguments in NetBeans, right click the "Project" name ⇒ "Set Configuration" ⇒ "Customize..." ⇒ Select categories "Run" ⇒ Enter the command-line arguments, e.g., "3 2 +" in the "Arguments" box (but make sure you select the proper Main class).

Question: Try "`java Arithmetic 2 4 *`" (in CMD shell and Eclipse/NetBeans) and explain the result obtained. How to resolve this problem?

In Windows' CMD shell, * is known as a wildcard character, that expands to give the list of file in the directory (called Shell Expansion). For example, "`dir *.java`" lists all the file with extension of "`.java`".

You could double-quote the * to prevent shell expansion. Eclipse has a bug in handling this, even * is double-quoted. NetBeans??

`SumDigits` (Command-line Arguments)

Write a program called **`SumDigits`** to sum up the individual digits of a positive integer, given in the command line. The output shall look like:

```
java SumDigits 12345
The sum of digits = 1 + 2 + 3 + 4 + 5 = 15
```

Exercises on Recursion

In programming, a recursive function (or method) calls itself. The classical example is `factorial(n)`, which can be defined recursively as `f(n)=n*f(n-1)`. Nonetheless, it is important to take note that a recursive function should have a terminating condition (or base case), in the case of factorial, `f(0)=1`. Hence, the full definition is:

```
factorial(n) = 1, for n = 0
factorial(n) = n * factorial(n-1), for all n > 1
```

For example, suppose `n = 5`:

```
// Recursive call
factorial(5) = 5 * factorial(4)
factorial(4) = 4 * factorial(3)
factorial(3) = 3 * factorial(2)
factorial(2) = 2 * factorial(1)
factorial(1) = 1 * factorial(0)
factorial(0) = 1  // Base case
// Unwinding
factorial(1) = 1 * 1 = 1
factorial(2) = 2 * 1 = 2
factorial(3) = 3 * 2 = 6
factorial(4) = 4 * 6 = 24
factorial(5) = 5 * 24 = 120 (DONE)
```

10.2 Factorial Recursive

Write a *recursive* method called `factorial()` to compute the factorial of the given integer.

```
public static int factorial(int n)
```

The recursive algorithm is:

```
factorial(n) = 1, if n = 0
factorial(n) = n * factorial(n-1), if n > 0
```

Compare your code with the *iterative* version of the `factorial()`:

```
factorial(n) = 1*2*3*...*n
```

Hints

Writing recursive function is straight forward. You simply translate the recursive definition into code with `return`.

```
// Return the factorial of the given integer, recursively
public static int factorial(int n) {
   if (n == 0) {
      return 1;   // base case
   } else {
      return n * factorial(n-1);  // call itself
   }
   // or one liner
   // return (n == 0) ? 1 : n*factorial(n-1);
}
```

Notes

1. Recursive version is often much shorter.
2. The recursive version uses much more computational and storage resources, and it need to save its current states before each successive recursive call, so as to unwind later.

10.3 Fibonacci (Recursive)

Write a *recursive* method to compute the Fibonacci number of n, defined as follows:

```
F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)  for n >= 2
```

Compare the recursive version with the *iterative* version written earlier.

Hints

```
// Translate the recursive definition into code with return statements
public static int fibonacci(int n) {
   if (n == 0) {
      return 0;
   } else if (n == 1) {
      return 1;
   } else {
      return fibonacci(n-1) + fibonacci(n-2);
   }
}
```

10.4 Length of a Running Number Sequence (Recursive)

A special number sequence is defined as follows:

```
S(1) = 1
S(2) = 12
S(3) = 123
S(4) = 1234
......
S(9) = 123456789         // length is 9
S(10) = 12345678910      // length is 11
S(11) = 1234567891011    // length is 13
S(12) = 123456789101112  // length is 15
......
```

Write a *recursive* method to compute the length of `S(n)`, defined as follows:

```
len(1) = 1
len(n) = len(n-1) + numOfDigits(n)
```

Also write an *iterative* version.

10.5 GCD (Recursive)

Write a recursive method called `gcd()` to compute the greatest common divisor of two given integers.

```
public static void int gcd(int a, int b)

gcd(a,b) = a, if b = 0
gcd(a,b) = gcd(b, remainder(a,b)), if b > 0
```

11. More (Difficult) Exercises

11.1 JDK Source Code

Extract the source code of the class `Math` from the JDK source code (JDK Installed Directory ⇒ `"lib"` ⇒ `"src.zip"` ⇒ `"java.base"` ⇒ `"java"` ⇒ `"lang"` ⇒ `"Math.java"`). Study how constants such as `E` and `PI` are defined. Also study how methods such as `abs()`, `max()`, `min()`, `toDegree()`, etc, are written.

Also study the `"Integer.java"`, `"String.java"`.

11.2 Matrices (2D Arrays)

Similar to `Math` class, write a `Matrix` library that supports matrix operations (such as addition, subtraction, multiplication) via 2D arrays. The operations shall support both `double` and `int`. Also write a test class to exercise all the operations programmed.

Hints

```
public class Matrix {
   // Method signatures
   public static void print(int[][] m);
   public static void print(double[][] m);
   public static boolean haveSameDimension(int[][] m1, int[][] m2);   // Used in
add(), subtract()
   public static boolean haveSameDimension(double[][] m1, double[][] m2);
   public static int[][] add(int[][] m1, int[][] m2);
   public static double[][] add(double[][] m1, double[][] m2);
   public static int[][] subtract(int[][] m1, int[][] m2);
   public static double[][] subtract(double[][] m1, double[][] m2);
   public static int[][] multiply(int[][] m1, int[][] m2);
   public static double[][] multiply(double[][] m1, double[][] m2);
   ......
}
```

11.3 PrintAnimalPattern (Special Characters and Escape Sequences)

Write a program called **`PrintAnimalPattern`**, which uses `println()` to produce this pattern:

```
          '__'
          (©©)
  /========\/
 / || %% ||
*  ||----||
   ¥¥    ¥¥
   ""    ""
```

Hints

Use escape sequence \uhhhh where hhhh are four hex digits to display Unicode characters such as ¥ and ©. ¥ is 165 (00A5H) and © is 169 (00A9H) in both ISO-8859-1 (Latin-1) and Unicode character sets.

Double-quote (") and black-slash (\) require escape sequence inside a String. Single quote (') does not require escape sign.

Try

Print the same pattern using `printf()`. (Hints: Need to use %% to print a % in `printf()` because % is the suffix for format specifier.)

11.4 Print Patterns (nested-loop)

Write a method to print each of the followings patterns using nested loops in a class called **`PrintPatterns`**. The program shall prompt user for the sizde of the pattern. The signatures of the methods are:

```
public static void printPatternX(int size);  // X: A, B, C,...; size is a positive integer.
```

```
# # # # # # # # # # #                #                          #
  # # # # # # # # #                # # #                      # # #
    # # # # # # #                # # # # #                  # # # # #
      # # # # #                # # # # # # #              # # # # # # #
        # # #                # # # # # # # # #          # # # # # # # # #
          #                # # # # # # # # # # #      # # # # # # # # # # #
         (a)                        (b)                 # # # # # # # # #
                                                          # # # # # # #
                                                            # # # # #
                                                              # # #
                                                                #
                                                               (c)
```

```
1                    1 2 3 4 5 6 7 8                    1      8 7 6 5 4 3 2 1
1 2                    1 2 3 4 5 6 7                  2 1      7 6 5 4 3 2 1
1 2 3                    1 2 3 4 5 6                3 2 1      6 5 4 3 2 1
1 2 3 4                    1 2 3 4 5              4 3 2 1      5 4 3 2 1
1 2 3 4 5                    1 2 3 4            5 4 3 2 1      4 3 2 1
1 2 3 4 5 6                    1 2 3          6 5 4 3 2 1      3 2 1
1 2 3 4 5 6 7                    1 2        7 6 5 4 3 2 1      2 1
1 2 3 4 5 6 7 8                    1      8 7 6 5 4 3 2 1      1
     (d)                  (e)                    (f)                (g)
```

```
              1                    1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
            1 2 1                    1 2 3 4 5 6 7 6 5 4 3 2 1
          1 2 3 2 1                    1 2 3 4 5 6 5 4 3 2 1
        1 2 3 4 3 2 1                    1 2 3 4 5 4 3 2 1
      1 2 3 4 5 4 3 2 1                    1 2 3 4 3 2 1
```

```
    1 2 3 4 5 6 5 4 3 2 1                    1 2 3 2 1
  1 2 3 4 5 6 7 6 5 4 3 2 1                    1 2 1
1 2 3 4 5 6 7 8 7 6 5 4 3 2 1                    1
             (h)                                (i)

1                           1      1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
1 2                       2 1      1 2 3 4 5 6 7   7 6 5 4 3 2 1
1 2 3                   3 2 1      1 2 3 4 5 6       6 5 4 3 2 1
1 2 3 4               4 3 2 1      1 2 3 4 5           5 4 3 2 1
1 2 3 4 5           5 4 3 2 1      1 2 3 4               4 3 2 1
1 2 3 4 5 6       6 5 4 3 2 1      1 2 3                   3 2 1
1 2 3 4 5 6 7   7 6 5 4 3 2 1      1 2                       2 1
1 2 3 4 5 6 7 8 7 6 5 4 3 2 1      1                           1
             (j)                                (k)

              1
            2 3 2
          3 4 5 4 3
        4 5 6 7 6 5 4
      5 6 7 8 9 8 7 6 5
    6 7 8 9 0 1 0 9 8 7 6
  7 8 9 0 1 2 3 2 1 0 9 8 7
8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
             (l)
```

11.5 Print Triangles (nested-loop)

Write a method to print each of the following patterns using nested-loops in a class called **PrintTriangles**. The program shall prompt user for the number of rows. The signatures of the methods are:

```
public static void printXxx(int numRows);  // Xxx is the pattern's name
```

```
                            1
                        1   2   1
                    1   2   4   2   1
                1   2   4   8   4   2   1
            1   2   4   8  16   8   4   2   1
        1   2   4   8  16  32  16   8   4   2   1
    1   2   4   8  16  32  64  32  16   8   4   2   1
1   2   4   8  16  32  64 128  64  32  16   8   4   2   1
                  (a) PowerOf2Triangle

1                                      1
1  1                                 1   1
1  2  1                            1   2   1
1  3  3  1                       1   3   3   1
```

```
1  4  6  4  1                  1    4    6    4    1
1  5 10 10  5  1             1    5   10   10    5    1
1  6 15 20 15  6  1        1    6   15   20   15    6    1
(b) PascalTriangle1           (c) PascalTriangle2
```

11.6 Trigonometric Series

Write a method to compute sin(x) and cos(x) using the following series expansion, in a class called **TrigonometricSeries**. The signatures of the methods are:

```
public static double sin(double x, int numTerms);   // x in radians, NOT degrees
public static double cos(double x, int numTerms);
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \cdots$$

Compare the values computed using the series with the JDK methods `Math.sin()`, `Math.cos()` at x=0, π/6, π/4, π/3, π/2 using various numbers of terms.

Hints

Do not use `int` to compute the factorial; as factorial of 13 is outside the `int` range. Avoid generating large numerator and denominator. Use `double` to compute the terms as:

$$\frac{x^n}{n!} = \left(\frac{x}{n}\right)\left(\frac{x}{n-1}\right)\ldots\left(\frac{x}{1}\right)$$

11.7 Exponential Series

Write a method to compute e and exp(x) using the following series expansion, in a class called **ExponentialSeries**. The signatures of the methods are:

```
public static double exp(int numTerms);   // x in radians
public static double exp(double x, int numTerms);
```

$$\mathrm{e} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots$$

$$\mathrm{e}^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots$$

11.8 Special Series

Write a method to compute the sum of the series in a class called `SpecialSeries`. The signature of the method is:

```
public static double specialSeries(double x, int numTerms);
```

$$x + \frac{1}{2} \times \frac{x^3}{3} + \frac{1 \times 3}{2 \times 4} \times \frac{x^5}{5} + \frac{1 \times 3 \times 5}{2 \times 4 \times 6} \times \frac{x^7}{7} + \frac{1 \times 3 \times 5 \times 7}{2 \times 4 \times 6 \times 8} \times \frac{x^9}{9} + \cdots; \; -1 \le x \le 1$$

11.9 FactorialInt (Handling Overflow)

Write a program called **FactorialInt** to list all the factorials that can be expressed as an `int` (i.e., 32-bit signed integer in the range of `[-2147483648, 2147483647]`). Your output shall look like:

```
The factorial of 1 is 1
The factorial of 2 is 2
```

```
...
The factorial of 12 is 479001600
The factorial of 13 is out of range
```

Hints

The maximum and minimum values of a 32-bit `int` are kept in constants `Integer.MAX_VALUE` and `Integer.MIN_VALUE`, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use `F(n) * (n+1) > Integer.MAX_VALUE` to check for overflow. Instead, overflow occurs for `F(n+1)` if `(Integer.MAX_VALUE / Factorial(n)) < (n+1)`, i.e., no more room for the next number.

Try

Modify your program called **`FactorialLong`** to list all the factorial that can be expressed as a `long` (64-bit signed integer). The maximum value for `long` is kept in a constant called `Long.MAX_VALUE`.

11.10 `FibonacciInt` (Handling Overflow)

Write a program called **`FibonacciInt`** to list all the Fibonacci numbers, which can be expressed as an `int` (i.e., 32-bit signed integer in the range of `[-2147483648, 2147483647]`). The output shall look like:

```
F(0) = 1
F(1) = 1
F(2) = 2
...
F(45) = 1836311903
F(46) is out of the range of int
```

Hints

The maximum and minimum values of a 32-bit `int` are kept in constants `Integer.MAX_VALUE` and `Integer.MIN_VALUE`, respectively. Try these statements:

```
System.out.println(Integer.MAX_VALUE);
System.out.println(Integer.MIN_VALUE);
System.out.println(Integer.MAX_VALUE + 1);
```

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use `F(n) = F(n-1) + F(n-2) > Integer.MAX_VALUE` to check for overflow. Instead, overflow occurs for `F(n)` if `Integer.MAX_VALUE – F(n-1) < F(n-2)` (i.e., no more room for the next Fibonacci number).

Try

Write a similar program called **`TribonacciInt`** for Tribonacci numbers.

11.11 Number System Conversion

Write a method call **toRadix()** which converts a positive integer from one radix into another. The method has the following header:

```
public static String toRadix(String in, int inRadix, int outRadix)  // The input
and output are treated as String.
```

Write a program called **NumberConversion**, which prompts the user for an input string, an input radix, and an output radix, and display the converted number. The output shall look like:

```
Enter a number and radix: A1B2
Enter the input radix: 16
Enter the output radix: 2
"A1B2" in radix 16 is "1010000110110010" in radix 2.
```

11.12 NumberGuess

Write a program called **NumberGuess** to play the number guessing game. The program shall generate a random number between 0 and 99. The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly. For example:

```
java NumberGuess
Key in your guess:
50
Try higher
70
Try lower
65
Try lower
61
You got it in 4 trials!
```

Hints

Use `Math.random()` to produce a random number in double between `0.0` (inclusive) and `1.0` (exclusive). To produce an int between `0` and `99`, use:

```
final int SECRET_NUMBER = (int)(Math.random()*100);  // truncate to int
```

11.13 WordGuess

Write a program called **WordGuess** to guess a word by trying to guess the individual characters. The word to be guessed shall be provided using the command-line argument. Your program shall look like:

```
java WordGuess testing
Key in one character or your guess word: t
Trial 1: t__t___
Key in one character or your guess word: g
Trial 2: t__t__g
Key in one character or your guess word: e
Trial 3: te_t__g
Key in one character or your guess word: testing
Congratulation!
You got in 4 trials
```

Hints

Set up a `boolean` array (of the length of the word to be guessed) to indicate the positions of the word that have been guessed correctly.

Check the length of the input `String` to determine whether the player enters a single character or a guessed word. If the player enters a single character, check it against the word to be guessed, and update the `boolean` array that keeping the result so far.

Try

Try retrieving the word to be guessed from a text file (or a dictionary) randomly.

11.14 `DateUtil`

Complete the following methods in a class called **`DateUtil`**:

- `boolean isLeapYear(int year)`: returns `true` if the given `year` is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.
- `boolean isValidDate(int year, int month, int day)`: returns `true` if the given `year`, `month` and `day` constitute a given date. Assume that year is between 1 and 9999, month is between 1 (Jan) to 12 (Dec) and day shall be between 1 and 28|29|30|31 depending on the month and whether it is a leap year.
- `int getDayOfWeek(int year, int month, int day)`: returns the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT, for the given date. Assume that the date is valid.
- `String toString(int year, int month, int day)`: prints the given date in the format "`xxxday d mmm yyyy`", e.g., "Tuesday 14 Feb 2012". Assume that the given date is valid.

Hints

To find the day of the week (Reference: Wiki "Determination of the day of the week"):

1700-	1800-	1900-	2000-	2100-	2200-	2300-	2400-
4	2	0	6	4	2	0	6

1. Based on the first two digit of the year, get the number from the following "century" table.
2. Take note that the entries 4, 2, 0, 6 repeat.
3. Add to the last two digit of the year.
4. Add to "the last two digit of the year divide by 4, truncate the fractional part".
5. Add to the number obtained from the following month table:

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Non-Leap Year	0	3	3	6	1	4	6	2	5	0	3	5
Leap Year	6	2	same as above									

6. Add to the day.
7. The sum modulus 7 gives the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT.

For example: 2012, Feb, 17

```
(6 + 12 + 12/4 + 2 + 17) % 7 = 5 (Fri)
```

The skeleton of the program is as follows:

```
/* Utilities for Date Manipulation */
public class DateUtil {

   // Month's name – for printing
   public static String[] strMonths
      = {"Jan", "Feb", "Mar", "Apr", "May", "Jun",
         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"};

   // Number of days in each month (for non-leap years)
   public static int[] daysInMonths
      = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

   // Returns true if the given year is a leap year
   public static boolean isLeapYear(int year) { ...... }

   // Return true if the given year, month, day is a valid date
   // year: 1-9999
   // month: 1(Jan)-12(Dec)
   // day: 1-28|29|30|31. The last day depends on year and month
   public static boolean isValidDate(int year, int month, int day) { ...... }

   // Return the day of the week, 0:Sun, 1:Mon, ..., 6:Sat
   public static int getDayOfWeek(int year, int month, int day) { ...... }

   // Return String "xxxday d mmm yyyy" (e.g., Wednesday 29 Feb 2012)
   public static String printDate(int year, int month, int day) { ...... }

   // Test Driver
   public static void main(String[] args) {
      System.out.println(isLeapYear(1900));  // false
      System.out.println(isLeapYear(2000));  // true
      System.out.println(isLeapYear(2011));  // false
      System.out.println(isLeapYear(2012));  // true

      System.out.println(isValidDate(2012, 2, 29));  // true
      System.out.println(isValidDate(2011, 2, 29));  // false
      System.out.println(isValidDate(2099, 12, 31)); // true
      System.out.println(isValidDate(2099, 12, 32)); // false

      System.out.println(getDayOfWeek(1982, 4, 24));  // 6:Sat
      System.out.println(getDayOfWeek(2000, 1, 1));   // 6:Sat
      System.out.println(getDayOfWeek(2054, 6, 19));  // 5:Fri
      System.out.println(getDayOfWeek(2012, 2, 17));  // 5:Fri

      System.out.println(toString(2012, 2, 14)); // Tuesday 14 Feb 2012
   }
}
```

Notes

You can compare the day obtained with the Java's `Calendar` class as follows:

```
// Construct a Calendar instance with the given year, month and day
Calendar cal = new GregorianCalendar(year, month - 1, day);  // month is 0-based
// Get the day of the week number: 1 (Sunday) to 7 (Saturday)
int dayNumber = cal.get(Calendar.DAY_OF_WEEK);
String[] calendarDays = { "Sunday", "Monday", "Tuesday", "Wednesday",
                          "Thursday", "Friday", "Saturday" };
// Print result
System.out.println("It is " + calendarDays[dayNumber - 1]);
```

The calendar we used today is known as *Gregorian calendar*, which came into effect in October 15, 1582 in some countries and later in other countries. It replaces the *Julian calendar*. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400. Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.

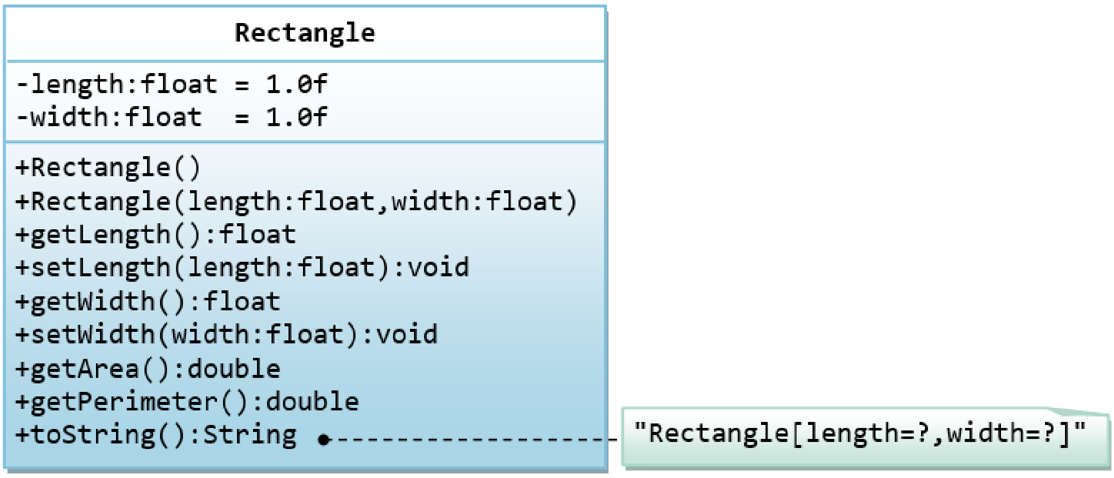
This above algorithm work for Gregorian dates only. It is difficult to modify the above algorithm to handle pre-Gregorian dates. A better algorithm is to find the number of days from a known date.

12. Exercises on Classes and Objects

12.1 The Rectangle Class

A class called `Rectangle`, which models a rectangle with a length and a width (in `float`), is designed as shown in the following class diagram. Write the `Rectangle` class.

Hints:



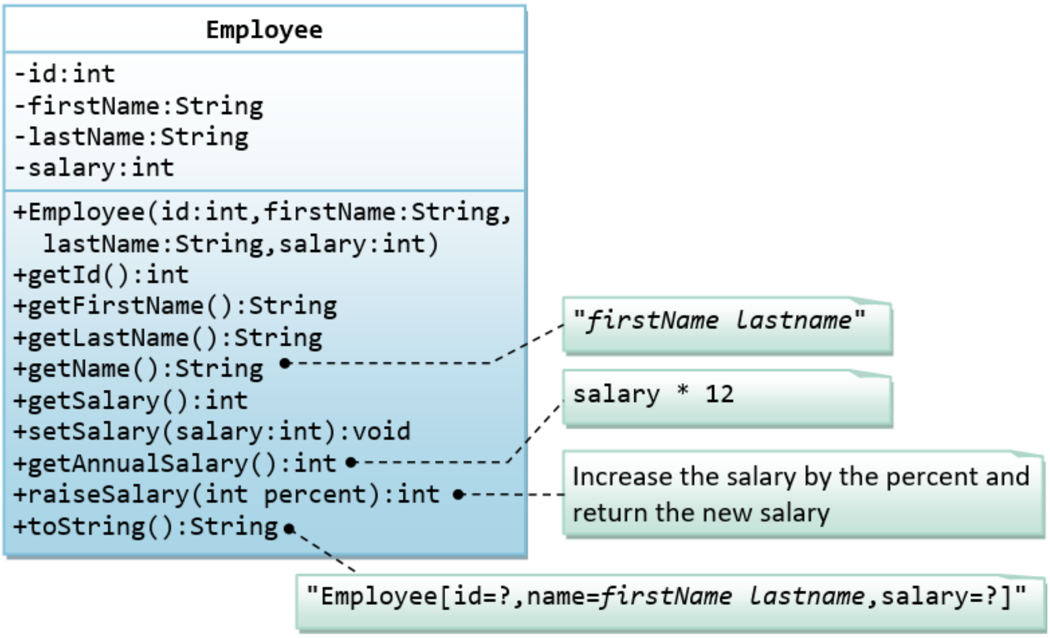
The expected output is:

```
Rectangle[length=1.2,width=3.4]
Rectangle[length=1.0,width=1.0]
Rectangle[length=5.6,width=7.8]
length is: 5.6
width is: 7.8
area is: 43.68
perimeter is: 26.80
```

12.2 The Employee Class

A class called `Employee`, which models an employee with an ID, name and salary, is designed as shown in the following class diagram. The method `raiseSalary(percent)` increases the salary by the given percentage. Write the `Employee` class.

Hints:


Employee
-id:int
-firstName:String
-lastName:String
-salary:int
+Employee(id:int,firstName:String,
lastName:String,salary:int)
+getId():int
+getFirstName():String
+getLastName():String
+getName():String
+getSalary():int
+setSalary(salary:int):void
+getAnnualSalary():int
+raiseSalary(int percent):int
+toString():String
"firstName lastname"
salary * 12
Increase the salary by the percent and return the new salary
"Employee[id=?,name=firstName lastname,salary=?]"

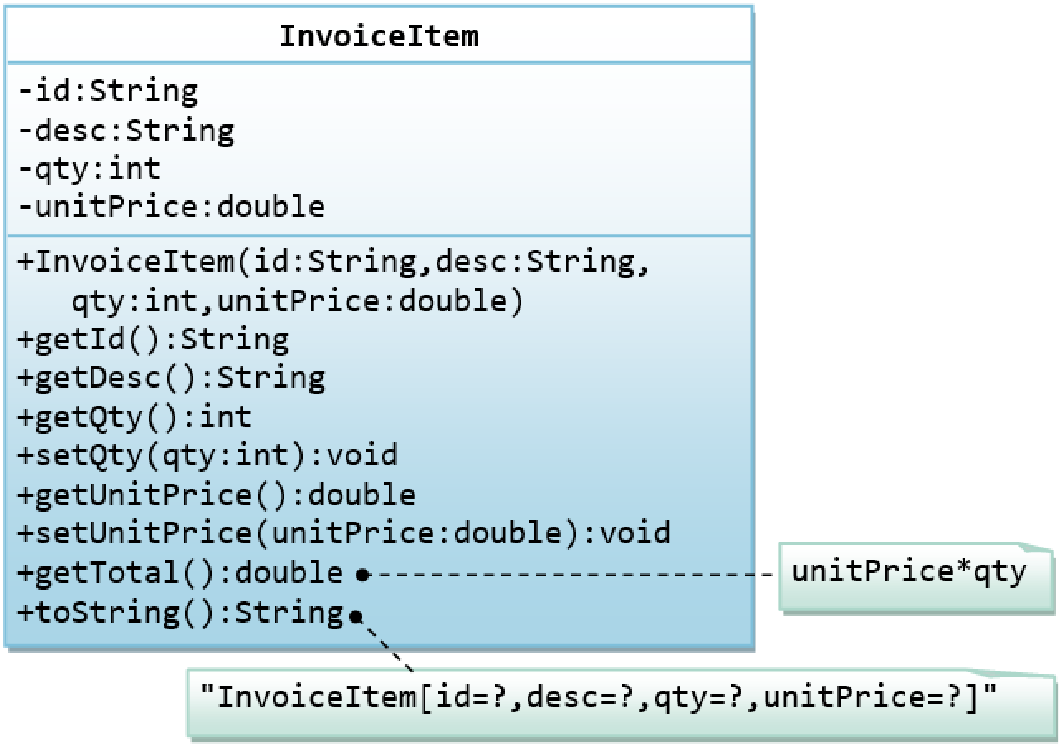
The expected out is:

```
Employee[id=8,name=Peter Tan,salary=2500]
Employee[id=8,name=Peter Tan,salary=999]
id is: 8
firstname is: Peter
lastname is: Tan
salary is: 999
name is: Peter Tan
annual salary is: 11988
1098
Employee[id=8,name=Peter Tan,salary=1098]
```

12.3 The InvoiceItem Class

A class called InvoiceItem, which models an item of an invoice, with ID, description, quantity and unit price, is designed as shown in the following class diagram. Write the InvoiceItem class.

Hints:



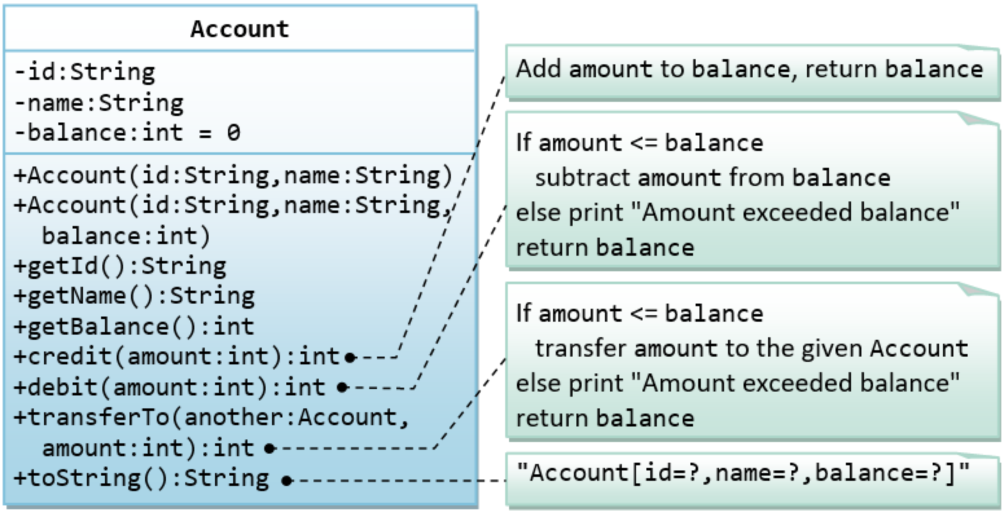
The expected output is:

```
InvoiceItem[id=A101,desc=Pen Red,qty=888,unitPrice=0.08]
InvoiceItem[id=A101,desc=Pen Red,qty=999,unitPrice=0.99]
id is: A101
desc is: Pen Red
qty is: 999
unitPrice is: 0.99
The total is: 989.01
```

12.4 The Account Class

A class called Account, which models a bank account of a customer, is designed as shown in the following class diagram. The methods credit(amount) and debit(amount) add or subtract the given amount to the balance. The method transferTo(anotherAccount, amount) transfers the given amount from this Account to the given anotherAccount. Write the Account class.

Hints:


Account
-id:String
-name:String
-balance:int = 0
+Account(id:String,name:String)
+Account(id:String,name:String,
balance:int)
+getId():String
+getName():String
+getBalance():int
+credit(amount:int):int
+debit(amount:int):int
+transferTo(another:Account,
amount:int):int
+toString():String
Add amount to balance, return balance
If amount <= balance
subtract amount from balance
else print "Amount exceeded balance"
return balance
If amount <= balance
transfer amount to the given Account
else print "Amount exceeded balance"
return balance
"Account[id=?,name=?,balance=?]"

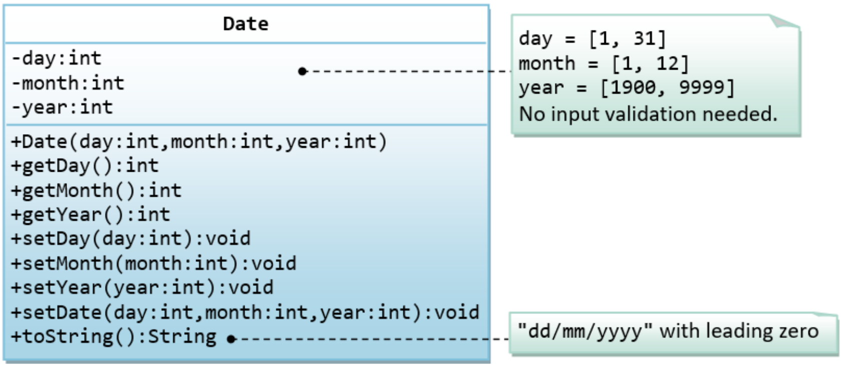
The expected output is:

```
Account[id=A101,name=Tan Ah Teck,balance=88]
Account[id=A102,name=Kumar,balance=0]
ID: A101
Name: Tan Ah Teck
Balance: 88
Account[id=A101,name=Tan Ah Teck,balance=188]
Account[id=A101,name=Tan Ah Teck,balance=138]
Amount exceeded balance
Account[id=A101,name=Tan Ah Teck,balance=138]
Account[id=A101,name=Tan Ah Teck,balance=38]
Account[id=A102,name=Kumar,balance=100]
```

12.5 The Date Class

A class called Date, which models a calendar date, is designed as shown in the following class diagram. Write the Date class.

Hints:



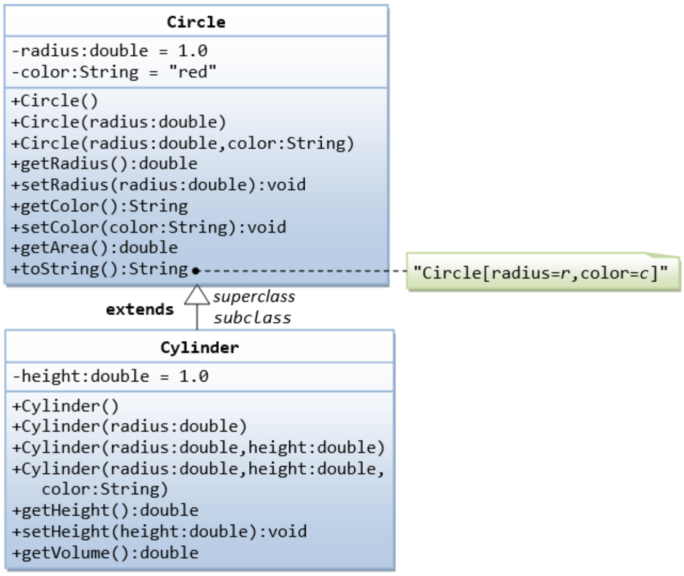
The expected output is:

```
01/02/2014
09/12/2099
Month: 12
Day: 9
Year: 2099
03/04/2016
```

13. Exercises on Inheritance

13.1 An Introduction to OOP Inheritance - The Circle and Cylinder Classes

This exercise shall guide you through the important concepts in inheritance.



In this exercise, a subclass called `Cylinder` is derived from the superclass `Circle` as shown in the class diagram (where an an arrow pointing up from the subclass to its superclass). Study how the subclass `Cylinder` invokes the superclass' constructors (via `super()` and `super(radius)`) and inherits the variables and methods from the superclass `Circle`.

You can reuse the `Circle` class that you have created in the previous exercise. Make sure that you keep "`Circle.class`" in the same directory.

```
public class Cylinder extends Circle {  // Save as "Cylinder.java"
   private double height;  // private variable

   // Constructor with default color, radius and height
   public Cylinder() {
      super();        // call superclass no-arg constructor Circle()
      height = 1.0;
   }
   // Constructor with default radius, color but given height
   public Cylinder(double height) {
      super();        // call superclass no-arg constructor Circle()
      this.height = height;
   }
   // Constructor with default color, but given radius, height
   public Cylinder(double radius, double height) {
      super(radius);  // call superclass constructor Circle(r)
      this.height = height;
   }

   // A public method for retrieving the height
   public double getHeight() {
      return height;
   }

   // A public method for computing the volume of cylinder
   //  use superclass method getArea() to get the base area
   public double getVolume() {
      return getArea()*height;
   }
}
```

Method Overriding and "Super": The subclass `Cylinder` inherits `getArea()` method from its superclass Circle. Try *overriding* the `getArea()` method in the subclass `Cylinder` to compute the surface area (=2π×radius×height + 2×base-area) of the cylinder instead of base area. That is, if `getArea()` is called by a `Circle` instance, it returns the area. If `getArea()` is called by a `Cylinder` instance, it returns the surface area of the cylinder.

If you override the `getArea()` in the subclass `Cylinder`, the `getVolume()` no longer works. This is because the `getVolume()` uses the *overridden* `getArea()` method found in the same class. (Java runtime will search the superclass only if it cannot locate the method in this class). Fix the `getVolume()`.

Hints: After overridding the `getArea()` in subclass `Cylinder`, you can choose to invoke the `getArea()` of the superclass `Circle` by calling `super.getArea()`.

Try:

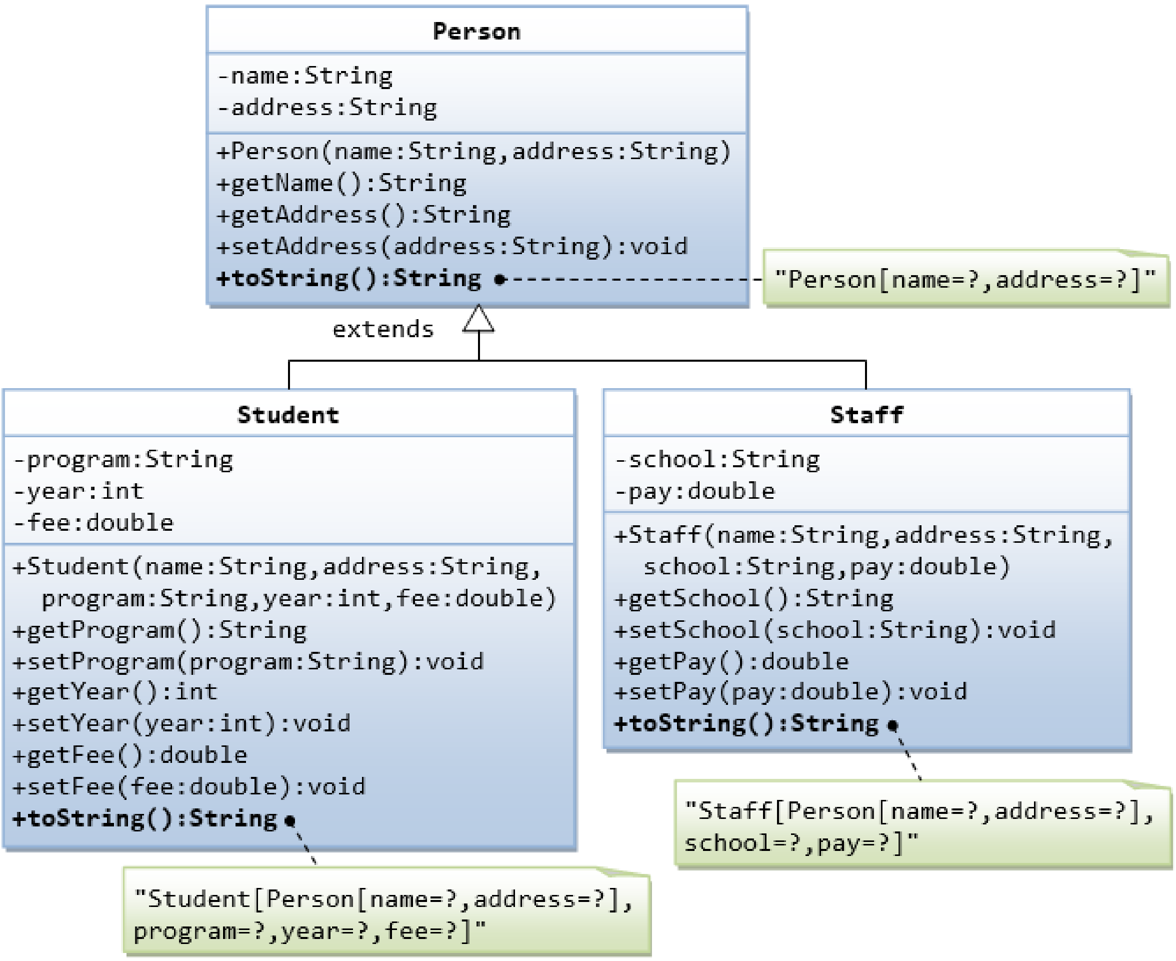
Provide a `toString()` method to the `Cylinder` class, which overrides the `toString()` inherited from the superclass `Circle`, e.g.,

```
@Override
public String toString() {      // in Cylinder class
   return "Cylinder: subclass of " + super.toString()  // use Circle's toString()
          + " height=" + height;
}
```

Try out the `toString()` method in `TestCylinder`.

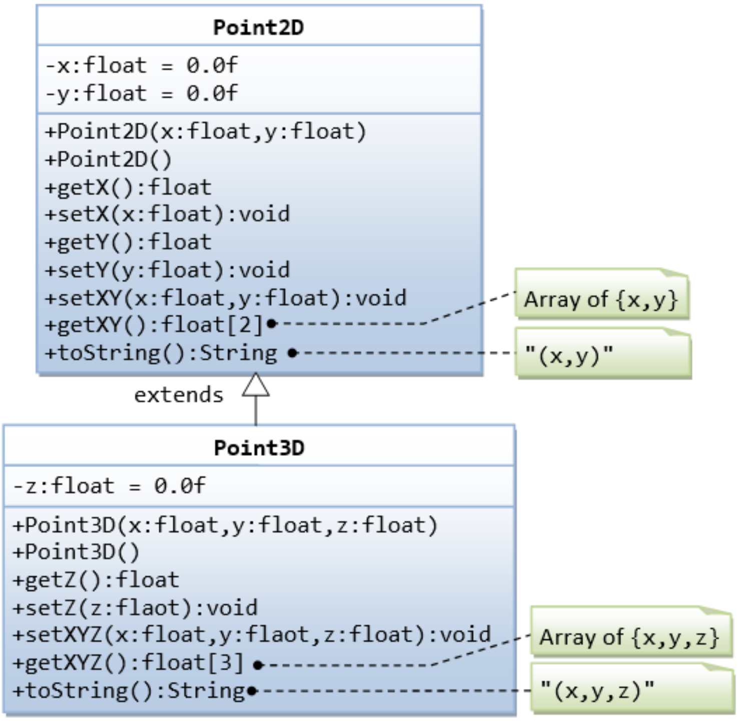
13.2 Superclass Person and its subclasses

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation `@Override`.


Person
-name:String
-address:String
+Person(name:String,address:String)
+getName():String
+getAddress():String
+setAddress(address:String):void
+toString():String
"Person[name=?,address=?]"
extends
Student
-program:String
-year:int
-fee:double
+Student(name:String,address:String, program:String,year:int,fee:double)
+getProgram():String
+setProgram(program:String):void
+getYear():int
+setYear(year:int):void
+getFee():double
+setFee(fee:double):void
+toString():String
"Student[Person[name=?,address=?], program=?,year=?,fee=?]"
Staff
-school:String
-pay:double
+Staff(name:String,address:String, school:String,pay:double)
+getSchool():String
+setSchool(school:String):void
+getPay():double
+setPay(pay:double):void
+toString():String
"Staff[Person[name=?,address=?], school=?,pay=?]"

13.3 Point2D and Point3D

Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation @Override.



Hints:

1. You cannot assign floating-point literal say `1.1` (which is a `double`) to a `float` variable, you need to add a suffix f, e.g. `0.0f`, `1.1f`.
2. The instance variables x and y are `private` in `Point2D` and cannot be accessed directly in the subclass `Point3D`. You need to access via the `public` getters and setters. For example,

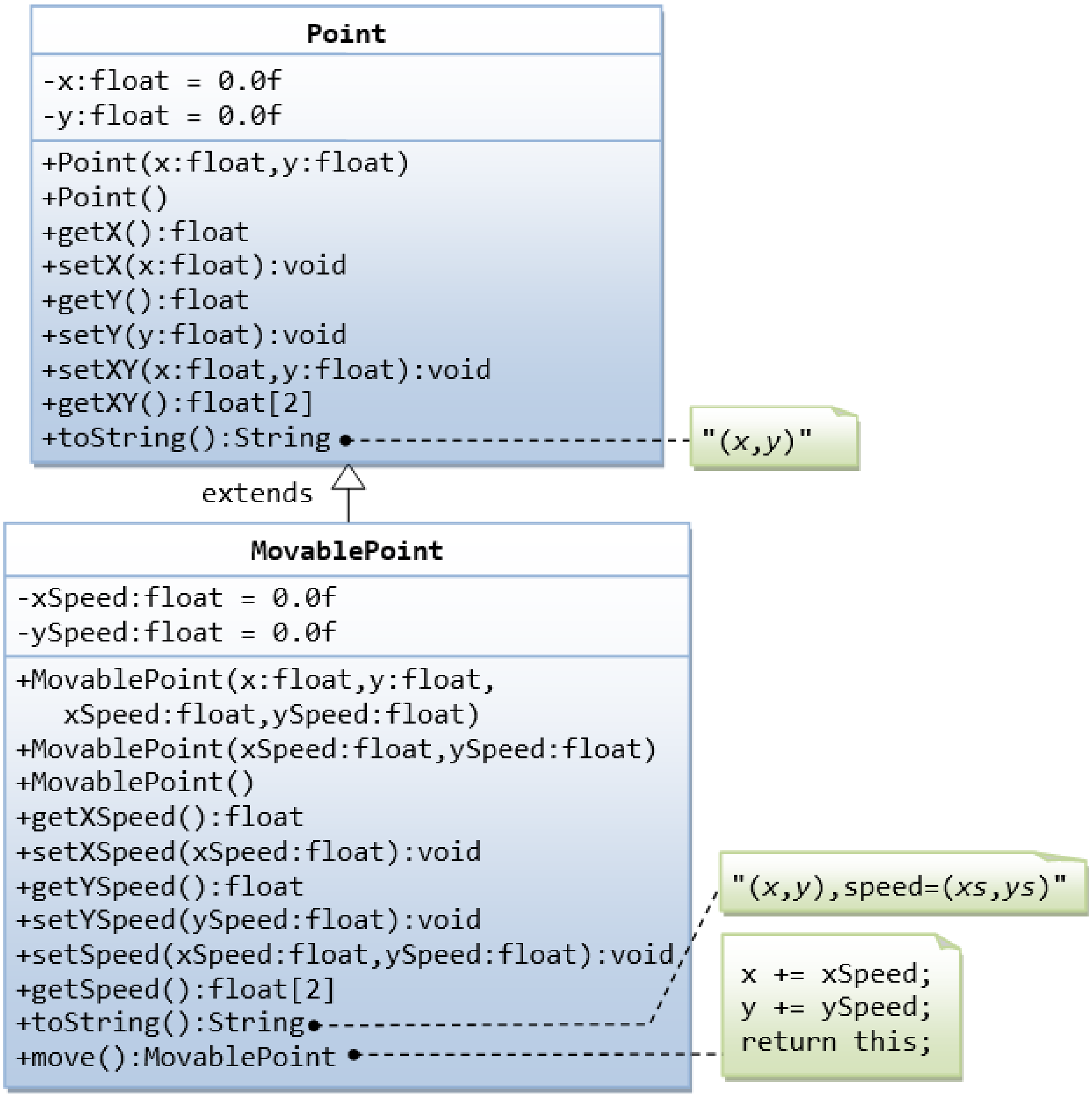
```
public void setXYZ(float x, float y, float z) {
       setX(x);     // or super.setX(x), use setter in superclass
       setY(y);
       this.z = z;
    }
```

3. The method `getXY()` shall return a `float` array:

```
public float[] getXY() {
       float[] result = new float[2];  // construct an array of 2 elements
       result[0] = ...
       result[1] = ...
       return result;  // return the array
    }
```

13.4 Point and MovablePoint

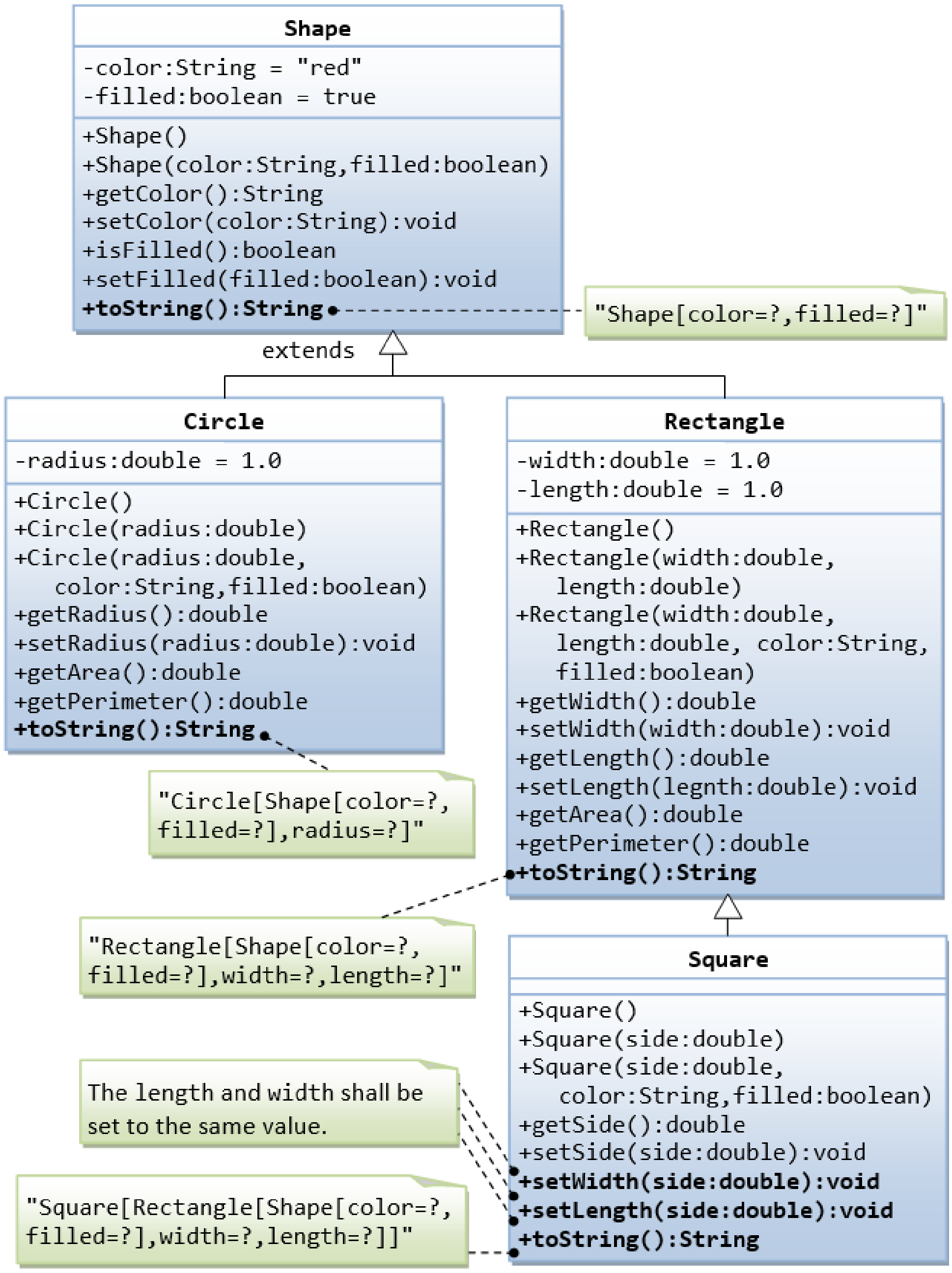
Write the classes as shown in the following class diagram. Mark all the overridden methods with annotation @Override.



Hints

1. You cannot assign floating-point literal say 1.1 (which is a double) to a float variable, you need to add a suffix f, e.g. 0.0f, 1.1f.
2. The instance variables x and y are private in Point and cannot be accessed directly in the subclass MovablePoint. You need to access via the public getters and setters. For example, you cannot write x += xSpeed, you need to write setX(getX() + xSpeed).

13.5 Superclass Shape and its subclasses Circle, Rectangle and Square


Shape
-color:String = "red"
-filled:boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+toString():String
"Shape[color=?,filled=?]"
extends
Circle
-radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?, filled=?],radius=?]"
Rectangle
-width:double = 1.0
-length:double = 1.0
+Rectangle()
+Rectangle(width:double, length:double)
+Rectangle(width:double, length:double, color:String, filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?, filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double, color:String,filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
The length and width shall be set to the same value.
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

Write a superclass called Shape (as shown in the class diagram)
Write a test program to test all the methods defined in Shape.
Write two subclasses of Shape called Circle and Rectangle, as shown in the class diagram.
Write a class called Square, as a subclass of Rectangle. Convince yourself that Square can be modeled as a subclass of Rectangle. Square has no instance variable, but inherits the instance variables width and length from its superclass Rectangle.

- Provide the appropriate constructors (as shown in the class diagram).

Hints:

```
public Square(double side) {
   super(side, side);  // Call superclass Rectangle(double, double)
}
```

- Override the toString() method to return "A Square with side=xxx, which is a subclass of yyy", where yyy is the output of the toString() method from the superclass.

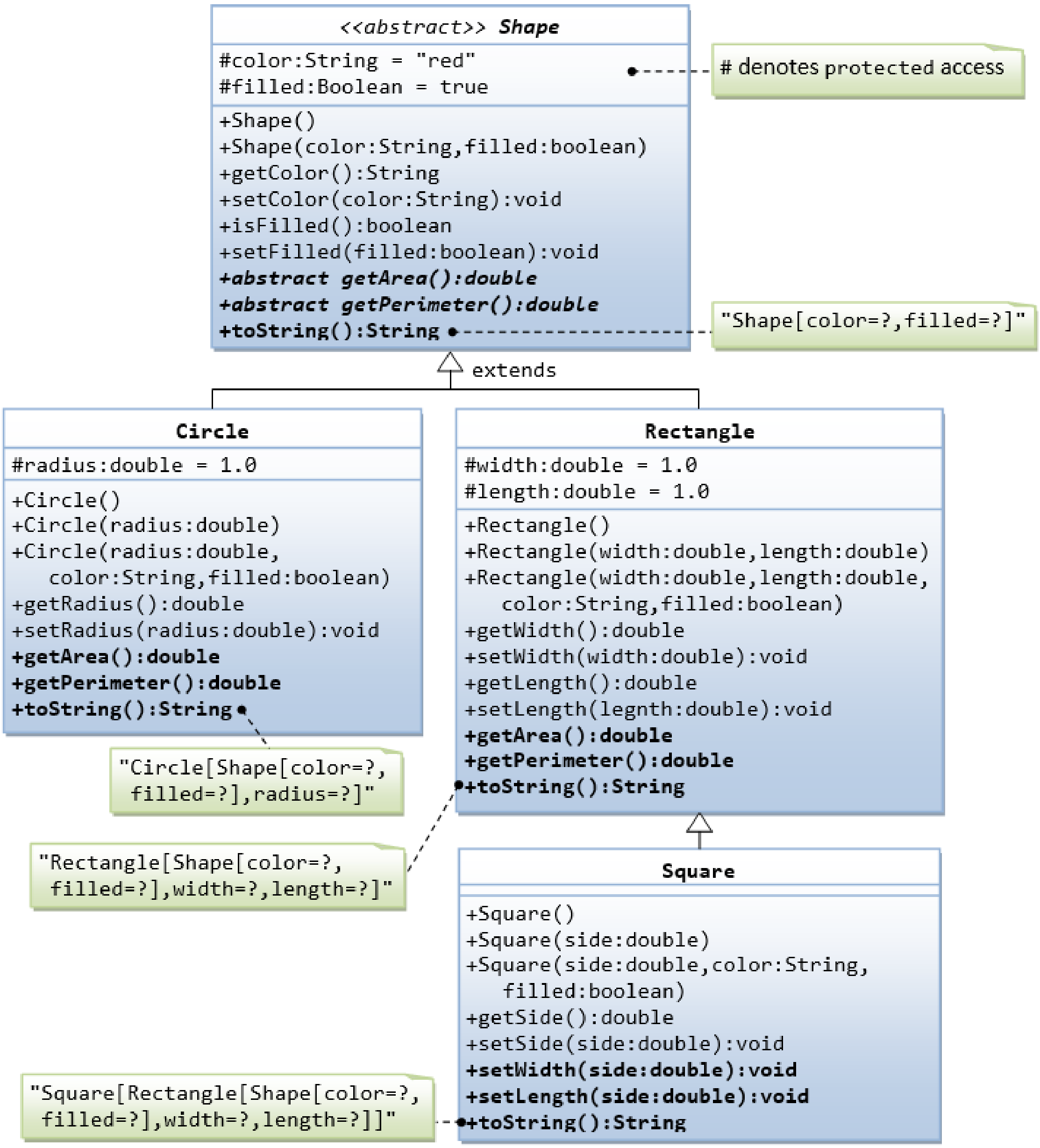
- Do you need to override the `getArea()` and `getPerimeter()`? Try them out.
- Override the `setLength()` and `setWidth()` to change both the `width` and `length`, so as to maintain the square geometry.

14. Exercises on Polymorphism, Abstract Classes and Interfaces

14.1 Ex: Abstract Superclass Shape and Its Concrete Subclasses

Rewrite the superclass Shape and its subclasses `Circle`, `Rectangle` and `Square`, as shown in the class diagram.

`Shape` is an `abstract` class containing 2 `abstract` methods: `getArea()` and `getPerimeter()`, where its concrete subclasses must provide its implementation. All instance variables shall have `protected` access, i.e., accessible by its subclasses and classes in the same package. Mark all the overridden methods with annotation `@Override`.


<<abstract>> Shape
#color:String = "red"
#filled:Boolean = true
+Shape()
+Shape(color:String,filled:boolean)
+getColor():String
+setColor(color:String):void
+isFilled():boolean
+setFilled(filled:boolean):void
+abstract getArea():double
+abstract getPerimeter():double
+toString():String
# denotes protected access
"Shape[color=?,filled=?]"
extends
Circle
#radius:double = 1.0
+Circle()
+Circle(radius:double)
+Circle(radius:double, color:String,filled:boolean)
+getRadius():double
+setRadius(radius:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Circle[Shape[color=?, filled=?],radius=?]"
Rectangle
#width:double = 1.0
#length:double = 1.0
+Rectangle()
+Rectangle(width:double,length:double)
+Rectangle(width:double,length:double, color:String,filled:boolean)
+getWidth():double
+setWidth(width:double):void
+getLength():double
+setLength(legnth:double):void
+getArea():double
+getPerimeter():double
+toString():String
"Rectangle[Shape[color=?, filled=?],width=?,length=?]"
Square
+Square()
+Square(side:double)
+Square(side:double,color:String, filled:boolean)
+getSide():double
+setSide(side:double):void
+setWidth(side:double):void
+setLength(side:double):void
+toString():String
"Square[Rectangle[Shape[color=?, filled=?],width=?,length=?]]"

In this exercise, `Shape` shall be defined as an `abstract` class, which contains:

- Two protected instance variables **color(String)** and **filled(boolean).** The protected variables can be accessed by its subclasses and classes in the same package. They are denoted with a '#' sign in the class diagram.
- Getter and setter for all the instance variables, and toString().
- Two abstract methods getArea() and getPerimeter() (shown in italics in the class diagram).
- Subclasses Circle and Rectangle shall *override* the abstract methods getArea() and getPerimeter() and provide the proper implementation. They also *override* the toString().

Write a test class to test these statements involving polymorphism and explain the outputs. Some statements may trigger compilation errors. Explain the errors, if any.

```
Shape s1 = new Circle(5.5, "red", false);  // Upcast Circle to Shape
System.out.println(s1);                    // which version?
System.out.println(s1.getArea());          // which version?
System.out.println(s1.getPerimeter());     // which version?
System.out.println(s1.getColor());
System.out.println(s1.isFilled());
System.out.println(s1.getRadius());

Circle c1 = (Circle)s1;                   // Downcast back to Circle
System.out.println(c1);
System.out.println(c1.getArea());
System.out.println(c1.getPerimeter());
System.out.println(c1.getColor());
System.out.println(c1.isFilled());
System.out.println(c1.getRadius());

Shape s2 = new Shape();

Shape s3 = new Rectangle(1.0, 2.0, "red", false);   // Upcast
System.out.println(s3);
System.out.println(s3.getArea());
System.out.println(s3.getPerimeter());
System.out.println(s3.getColor());
System.out.println(s3.getLength());

Rectangle r1 = (Rectangle)s3;   // downcast
System.out.println(r1);
System.out.println(r1.getArea());
System.out.println(r1.getColor());
System.out.println(r1.getLength());

Shape s4 = new Square(6.6);     // Upcast
System.out.println(s4);
System.out.println(s4.getArea());
System.out.println(s4.getColor());
System.out.println(s4.getSide());

// Take note that we downcast Shape s4 to Rectangle,
//  which is a superclass of Square, instead of Square
Rectangle r2 = (Rectangle)s4;
System.out.println(r2);
System.out.println(r2.getArea());
System.out.println(r2.getColor());
System.out.println(r2.getSide());
System.out.println(r2.getLength());

// Downcast Rectangle r2 to Square
```

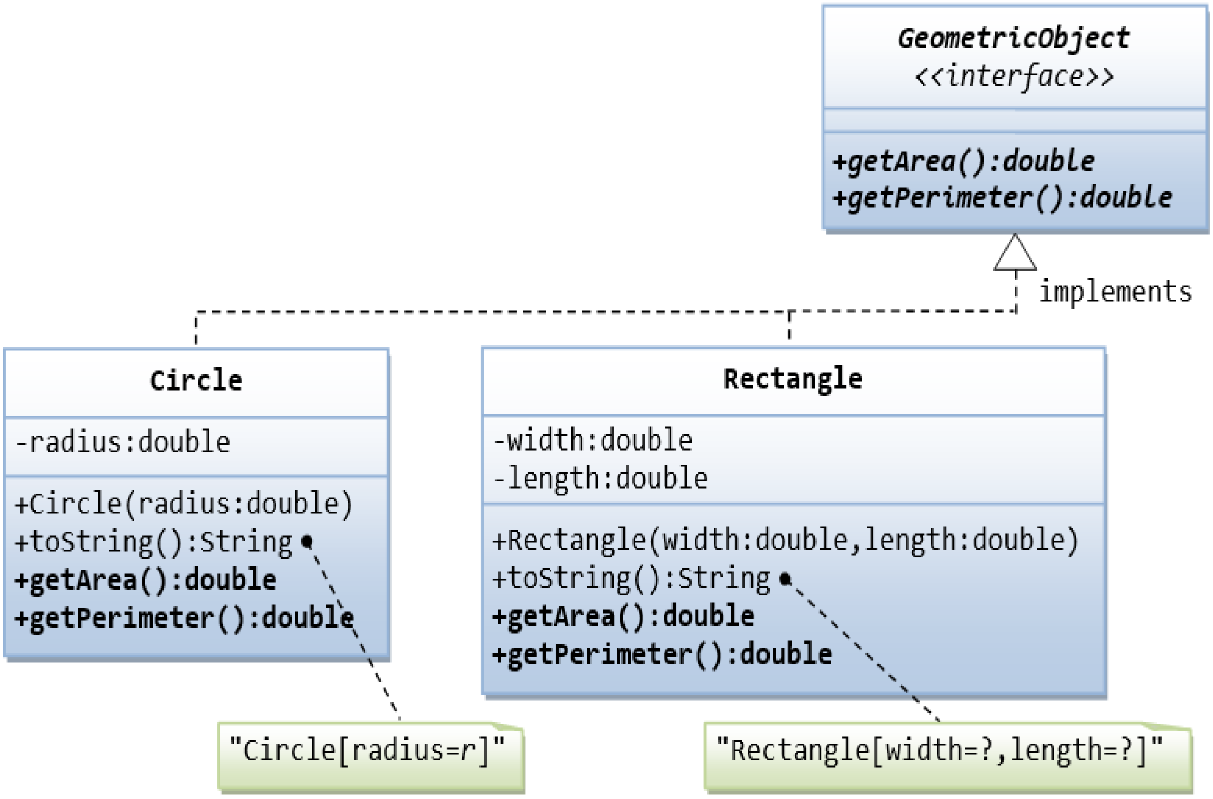
```
Square sq1 = (Square)r2;
System.out.println(sq1);
System.out.println(sq1.getArea());
System.out.println(sq1.getColor());
System.out.println(sq1.getSide());
System.out.println(sq1.getLength());
```

Try:

Explain the usage of the `abstract` method and `abstract` class?

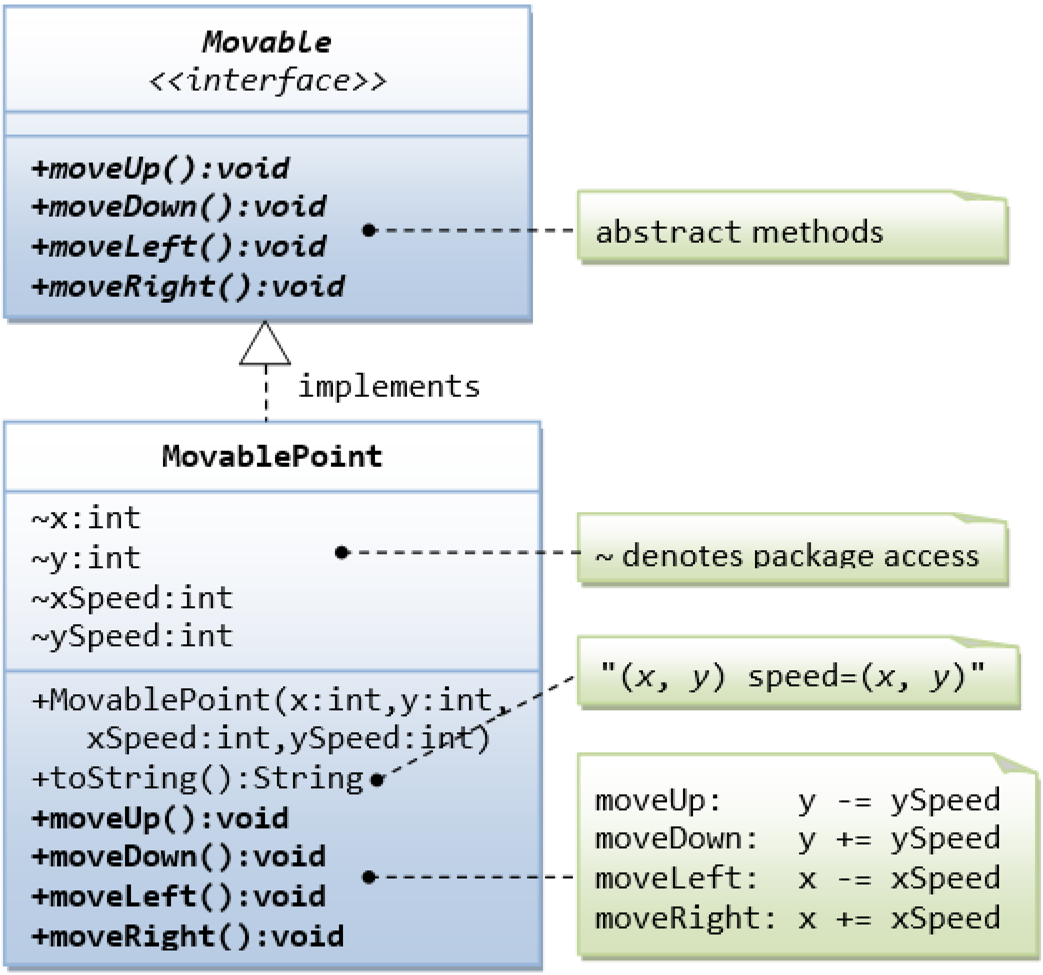
14.2 GeometricObject Interface and its Implementation Classes Circle and Rectangle

Write an interface called `GeometricObject`, which contains 2 abstract methods: getArea() and `getPerimeter()`, as shown in the class diagram. Also write an implementation class called `Circle`. Mark all the overridden methods with annotation `@Override`.



14.3 Ex: Movable Interface and its Implementation MovablePoint Class

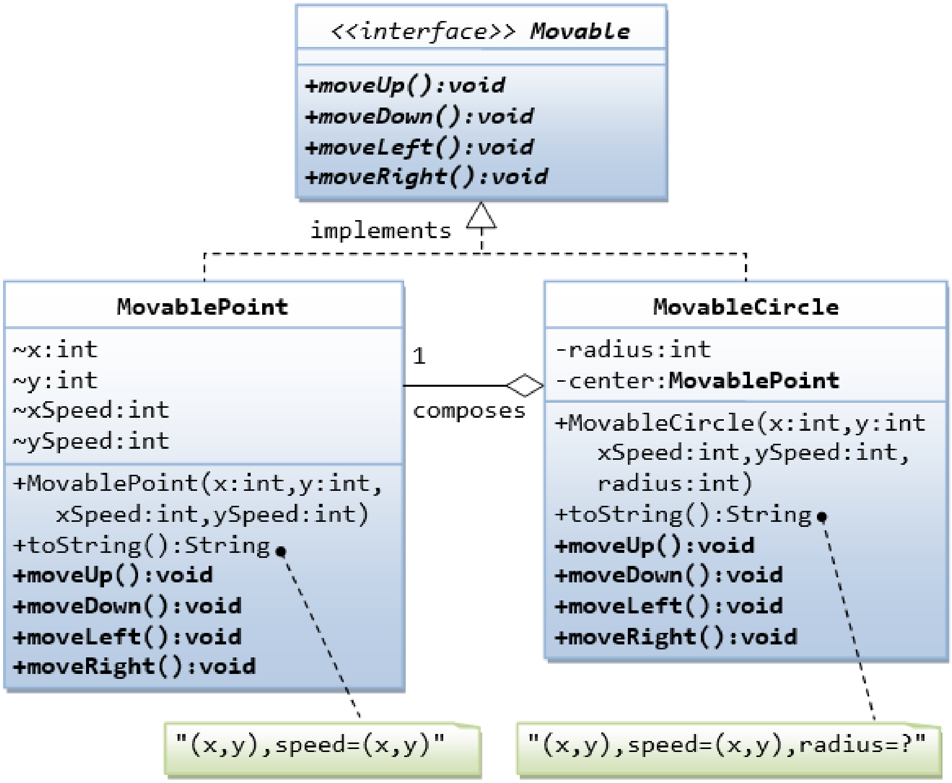
Write an interface called `Movaable`, which contains 4 abstract methods moveUp(), moveDown(), moveLeft() and `moveRight()`, as shown in the class diagram. Also write an implementation class called `MovablePoint`. Mark all the overridden methods with annotation `@Override`.



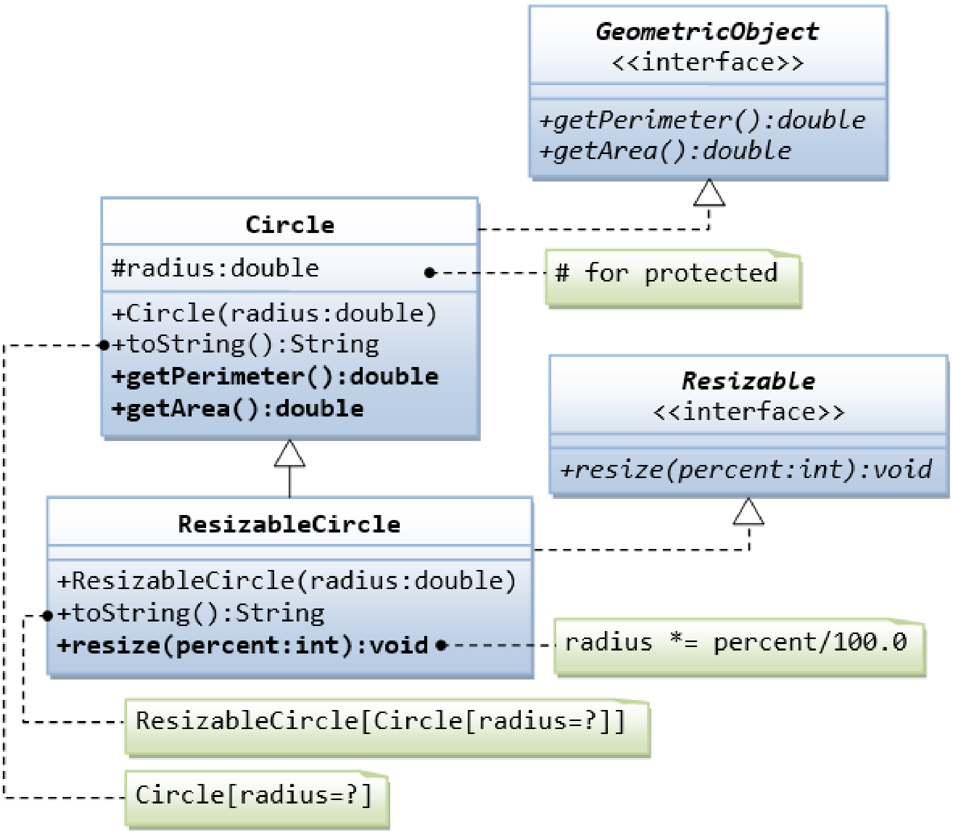
14.4 Movable Interface and Classes MovablePoint and MovableCircle

Write an interface called `Movaable`, which contains 4 abstract methods `moveUp()`, `moveDown()`, `moveLeft()` and `moveRight()`, as shown in the class diagram.

Write the implementation classes called `MovablePoint` and `MovableCircle`. Mark all the overridden methods with annotation `@Override`.



14.5 Interfaces Resizable and GeometricObject



Write the `interface` called `GeometricObject`, which declares two abstract methods: getParameter() and `getArea()`, as specified in the class diagram.

Hints:

```
public interface GeometricObject {
   public double getPerimeter();
   ......
}
```

Write the implementation class Circle, with a protected variable radius, which implements the interface GeometricObject.

Hints:

```
public class Circle implements GeometricObject {
   // Private variable
   ......

   // Constructor
   ......

   // Implement methods defined in the interface GeometricObject
   @Override
   public double getPerimeter() { ...... }

   ......
}
```

Write a test program called TestCircle to test the methods defined in Circle.

The class ResizableCircle is defined as a subclass of the class Circle, which also implements an interface called Resizable, as shown in class diagram. The interface Resizable declares an abstract method resize(), which modifies the dimension (such as radius) by the given percentage. Write the interface Resizable and the class ResizableCircle.

Hints:

```
public interface Resizable {
   public double resize(...);
}
```

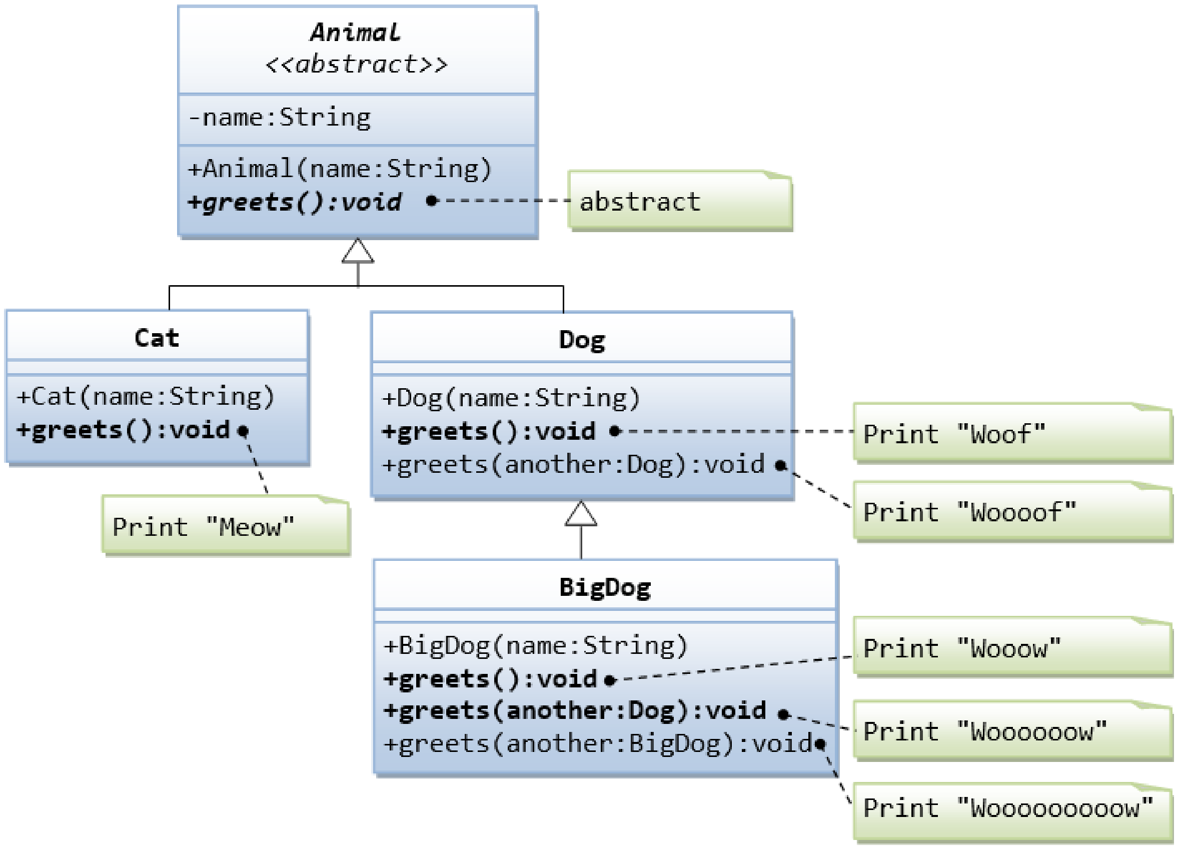
```
public class ResizableCircle extends Circle implements Resizeable {

   // Constructor
   public ResizableCircle(double radius) {
      super(...);
   }

   // Implement methods defined in the interface Resizable
   @Override
   public double resize(int percent) { ...... }
}
```

Try:

Write a test program called TestResizableCircle to test the methods defined in ResizableCircle.

14.6 Abstract Superclass Animal and its Implementation Subclasses

Write the codes for all the classes shown in the class diagram. Mark all the overridden methods with annotation @Override.



14.7 Another View of Abstract Superclass Animal and its Implementation Subclasses

Examine the following codes and draw the class diagram.

```
abstract public class Animal {
   abstract public void greeting();
}
```

```
public class Cat extends Animal {
   @Override
   public void greeting() {
      System.out.println("Meow!");
   }
}
public class Dog extends Animal {
   @Override
   public void greeting() {
      System.out.println("Woof!");
   }

   public void greeting(Dog another) {
      System.out.println("Woooooooooof!");
   }
}
```

```
public class BigDog extends Dog {
   @Override
   public void greeting() {
      System.out.println("Woow!");
   }

   @Override
   public void greeting(Dog another) {
      System.out.println("Woooooowwwww!");
   }
}
```

Try:
Explain the outputs (or error) for the following test program.

```
public class TestAnimal {
   public static void main(String[] args) {
      // Using the subclasses
      Cat cat1 = new Cat();
      cat1.greeting();
      Dog dog1 = new Dog();
      dog1.greeting();
      BigDog bigDog1 = new BigDog();
      bigDog1.greeting();

      // Using Polymorphism
      Animal animal1 = new Cat();
      animal1.greeting();
      Animal animal2 = new Dog();
      animal2.greeting();
      Animal animal3 = new BigDog();
      animal3.greeting();
      Animal animal4 = new Animal();

      // Downcast
      Dog dog2 = (Dog)animal2;
      BigDog bigDog2 = (BigDog)animal3;
      Dog dog3 = (Dog)animal3;
      Cat cat2 = (Cat)animal2;
      dog2.greeting(dog3);
      dog3.greeting(dog2);
      dog2.greeting(bigDog2);
      bigDog2.greeting(dog2);
      bigDog2.greeting(bigDog1);
   }
}
```

15. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).

- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
• CodeChef	20	200
• Leetcode	20	200
• GeeksforGeeks	20	200
• SPOJ	5	50
• InterviewBit	10	1000
• Hackerrank	25	250
• Codeforces	10	100
• BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certifications:

- HackerRank – Java Basic Skills Certification
- Oracle Certified Associate Java Programmer OCAJP
- CodeChef - Learn Java Certification
- NPTEL – Programming in Java
- NPTEL – Data Structures and Algorithms in Java

V. TEXT BOOKS:

1. Farrell, Joyce.*Java Programming*, Cengage Learning B S Publishers, 8th Edition, 2020
2. Schildt, Herbert. *Java: The Complete Reference* 11th Edition, McGraw-Hill Education, 2018.

VI. REFERENCE BOOKS:

1. Deitel, Paul and Deitel, Harvey. *Java: How to Program*, Pearson, 11th Edition, 2018.
2. Evans, Benjamin J. and Flanagan, David. *Java in a Nutshell*, O'Reilly Media, 7th Edition, 2018.
3. Bloch, Joshua. *Effective Java*, Addison-Wesley Professional, 3rd Edition, 2017.
4. Sierra, Kathy and Bates, Bert. *Head First Java*, O'Reilly Media, 2nd Edition, 2005

VII. ELECTRONICS RESOURCES:

1. https://docs.oracle.com/en/java/
2. https://www.geeksforgeeks.org/java
3. https://www.tutorialspoint.com/java/index.htm
4. https://www.coursera.org/courses?query=java

VIII. MATERIALS ONLINE;

1. Syllabus
2. Lab manual

COURSE CONTENT

FRONT-END WEB DEVELOPMENT LABORATORY

I Semester: CSE / IT / CSE (AI&ML) / CSE (DS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSE04	Foundation	0	0	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: NIL	Practical Classes: 30				Total Classes: 30		
Prerequisite: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

The Front-End Web Development Laboratory course provides hands-on experience in building responsive and interactive web applications using HTML5, CSS3, and JavaScript. Students will learn to design structured web pages, apply styling and layout techniques, and implement client-side validations. The course introduces modern frameworks like Bootstrap/Tailwind CSS for responsive design and covers DOM manipulation, event handling, and API integration. Through practical experiments and mini-projects, learners develop problem-solving skills and creativity in web design. By the end of the lab, students will be able to design, validate, and deploy user-friendly front-end interfaces aligned with industry standards.

II. COURSES OBJECTIVES:

The students will try to learn

I Apply HTML5 and CSS3 to design structured, styled, and responsive web pages.
II Implement client-side validation, event handling, and DOM manipulation using JavaScript.
III Analyze CSS frameworks such as Bootstrap and Tailwind to develop responsive layouts.
IV Integrate APIs and asynchronous JavaScript techniques to create dynamic web applications.
V Design a mini-project demonstrating creativity and usability in front-end development.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1	Identify the basic structure and components of a web page using HTML5 elements.
CO 2	Apply CSS3 properties and layouts to enhance the presentation of web pages.
CO 3	Implement JavaScript for client-side validation, event handling, and interactivity.
CO 4	Analyze different CSS frameworks to choose appropriate tools for responsive web design.
CO 5	Develop APIs and asynchronous JavaScript features to build dynamic content-driven applications.
CO 6	Implement APIs and asynchronous JavaScript features to build dynamic content-driven applications

IV. COURE CONTENT:

The course is organized into 14 modules, each corresponding to one week of study. The structure ensures a gradual progression from fundamental concepts to advanced topics, with hands-on exercises and project work integrated throughout.

1. Module 1: Introduction & Web Basics
(a) Overview of how the web works and the client-server model.
(b) Frontend vs Backend vs Full Stack development.
(c) Setting up the development environment (VS Code, Git, browser tools).
(d) Creating and testing the first HTML page.

2. Module 2: HTML Fundamentals
(a) HTML elements and attributes.
(b) Semantic tags and document structure.
(c) Forms and input elements.
(d) Tables and multimedia.
(e) Accessibility features (alt text, ARIA basics).

3. Module 3: CSS Fundamentals
(a) CSS syntax and inclusion methods.
(b) Selectors and combinators.
(c) Box model, margins, padding, and borders.
(d) Typography and colors.
(e) Introduction to responsive design.

4. Module 4: CSS Layouts
(a) Positioning and display types.
(b) Floats and clear.
(c) Flexbox (deep dive).
(d) Grid layout (deep dive).
(e) Media queries for responsiveness.

5. Module 5: Advanced CSS & UI Design
(a) Transitions, transforms, and animations.
(b) Pseudo-classes and pseudo-elements.
(c) CSS variables and custom properties.
(d) Introduction to CSS frameworks (Bootstrap, TailwindCSS).

6. Module 6: JavaScript Basics
(a) Variables, data types, and operators.
(b) Functions and scope.
(c) DOM manipulation.
(d) Events and event handling.
(e) Debugging with browser console.

7. Module 7: JavaScript Control Flow
(a) Arrays and objects.
(b) Loops and conditionals.
(c) Event listeners and form validation.
(d) Mini-projects (e.g., calculator, form validator).

8. Module 8: JavaScript ES6+ Features
(a) let and const.
(b) Arrow functions.
(c) Template literals and destructuring.
(d) Spread and rest operators.
(e) Modules and imports.

9. Module 9: Asynchronous JavaScript
(a) Callbacks and promises.
(b) async/await.
(c) Fetch API for AJAX requests.
(d) Working with JSON.
(e) Mini-project: Weather App or API-based app.

10. Module 10: Advanced JavaScript Topics
(a) Closures and higher-order functions.
(b) DOM traversal.
(c) LocalStorage and SessionStorage.
(d) Error handling in JavaScript.
(e) Introduction to bundlers (Vite, Webpack).

11. Module 11: Introduction to React
(a) Why frameworks? SPA vs MPA.
(b) React basics: components and JSX.
(c) Props and state.
(d) Using React Developer Tools.

12. Module 12: React Advanced Concepts
(a) React Hooks: useState and useEffect.
(b) Handling forms in React.
(c) React Router for navigation.
(d) API calls within React.

13. Module 13: Styling in React & Project Work
(a) Styling with CSS Modules.
(b) Styled Components and TailwindCSS.
(c) Component libraries (Material UI, shadcn/ui).

14. Module 14: Capstone Project & Deployment
(a) Integrating all concepts into a final project.
(b) Using Git and GitHub for collaboration.
(c) Deployment on Netlify, Vercel, or GitHub Pages.
(d) Final project presentation and review.

V. PLATFORMS FOR WEB DEVELOPMENT

1. GitHub Pages
Website: https://pages.github.com/

GitHub Pages is a free hosting service for static websites directly from a GitHub repository.
It supports HTML, CSS, and JavaScript, making it ideal for deploying simple front-end projects. Developers can automate deployments using Git workflows and integrate CI/CD pipelines. It provides custom domain support and HTTPS security by default. However, it is not suitable for dynamic applications that require server-side processing.

2. Netlify
Website: https://www.netlify.com/

Netlify is a powerful platform for deploying static websites and JAMstack applications. It offers features like continuous deployment from Git, built-in form handling, and serverless functions. Netlify's free plan provides generous limits, making it suitable for small to medium-sized projects. It also includes automatic HTTPS, global CDN, and rollbacks for deployments. The platform supports integrations with popular tools like Next.js, Gatsby, and React.

3. Vercel
Website: https://vercel.com/

Vercel is a cloud platform optimized for front-end frameworks like Next.js, React, and Vue.js. It simplifies deployment with automatic builds and serverless functions. With a global edge network, Vercel ensures fast performance and scalability. It supports custom domains, environment variables, and preview deployments for collaboration. The platform is widely used by developers building modern web applications.

4. Firebase Hosting
Website: https://firebase.google.com/

Firebase Hosting is a Google-powered service designed for fast and secure static and dynamic web app deployment. It offers integration with Firebase's real-time database, authentication, and cloud functions. The free tier includes a generous amount of bandwidth and SSL encryption. It provides a seamless developer experience with one-command deployment. Firebase Hosting is ideal for Progressive Web Apps (PWAs) and mobile-friendly websites.

5. Heroku
Website: https://www.heroku.com/

Heroku is a cloud platform-as-a-service (PaaS) that enables developers to deploy full-stack applications effortlessly. It supports multiple programming languages, including Node.js, Python, and Ruby. Heroku provides add-ons for databases, caching, and monitoring, enhancing application scalability. Its free tier includes dynos (containers) that run apps with occasional sleeping periods. While easy to set up, Heroku is best suited for prototyping and small projects due to resource limitations.

6. CodeSandbox
Website: https://codesandbox.io/

CodeSandbox is an online IDE that provides an interactive environment for front-end and full-stack web development. It supports frameworks like React, Vue, and Angular, allowing developers to quickly prototype and share their projects. With real-time collaboration and live preview, CodeSandbox is ideal for testing and showcasing web applications. Its integration with GitHub enables seamless version control and deployment.

7. Glitch
Website: https://glitch.com/

Glitch is a collaborative coding platform that allows developers to create, edit, and deploy web applications instantly. It provides a real-time editor and automatic hosting for Node.js applications, making it an excellent choice for beginners. Glitch supports full-stack development with built-in databases and API integrations. Its remixable projects allow users to build upon existing applications effortlessly.

8. Replit

Website: https://replit.com/

Replit is an online IDE that supports multiple programming languages and frameworks. It offers cloud-based execution, enabling users to write, test, and deploy applications from any device. Replit provides real-time collaboration, making it ideal for pair programming and classroom learning. With built-in hosting and database support, Replit simplifies full-stack development.

9. WordPress
Website: https://wordpress.com/

WordPress is a widely used CMS that enables users to build and customize websites with ease. It offers responsive themes and plugins, making it a versatile platform for blogging, business sites, and e-commerce. The free plan allows users to create and host a basic website with WordPress branding. For more control, developers can self-host WordPress on their own server.

10. Wix
Website: https://www.wix.com/

Wix is a user-friendly website builder that provides drag-and-drop functionality for creating responsive websites. It offers a wide range of templates and customization options, making it ideal for beginners. Wix provides free hosting and domain services, allowing users to publish their websites instantly. While it is best for static sites, advanced users can add custom functionality using Wix Velo, a built-in development platform.

VI. SOFTWARE REQUIREMENTS

To implement web development mini-projects using the listed platforms, certain software requirements must be met. These requirements vary based on the specific tools and technologies used.

Basic Requirements

1. Web Browser: Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari (for testing and development).
2. Internet Connection: Required for cloud-based platforms like GitHub Pages, Netlify, Vercel, Firebase, etc.

Development Tools

1. Code Editor: Visual Studio Code, Sublime Text, or Atom (for writing HTML, CSS, JavaScript).
2. Terminal/Command Line: For running Git commands, package managers (npm, yarn), or deploying applications.
3. Git: For version control and hosting on platforms like GitHub and GitHub Pages.

Additional Software Based on Project Needs

1. Node.js & npm: Required for JavaScript-based frameworks (React, Vue, Angular, Next.js).
2. Python & Pip: Needed for Flask/Django-based web applications.
3. PHP & MySQL: If working with WordPress self-hosted sites or traditional backend development.
4. Docker: If containerizing applications for platforms like Heroku or Firebase.

VII. HARDWARE REQUIREMENTS

The hardware requirements depend on the complexity of the web development projects and the tools being used. Below are the recommended specifications:

Basic Requirements

1. Processor: Intel Core i3 (or equivalent) or higher.
2. RAM: Minimum 4GB (8GB or more recommended for smooth development).
3. Storage: At least 20GB of free space (SSD recommended for faster performance).
4. Display: Minimum resolution of 1366Ö768 (Full HD recommended).
5. Internet Connection: Required for cloud-based platforms and online development tools.

Recommended for Advanced Development

1. Processor: Intel Core i5/i7 or AMD Ryzen 5/7 for faster build times.
2. RAM: 16GB or more for handling multiple applications (e.g., Docker, Node.js, databases).
3. Storage: SSD with at least 256GB+ for better performance.
4. Graphics Card: Not necessary for basic web development, but useful for UI/UX design and graphics-intensive applications.

Additional Peripherals (Optional)
1. External Monitor: For multitasking and better UI/UX design experience.
2. Mechanical Keyboard & Mouse: For improved efficiency in coding.
3. Webcam & Microphone: If collaborating on projects via video conferencing.

EXERCISES: 1 - INTRODUCTION & WEB BASICS

Learning Outcomes

By completing these exercises, students will:
1. Configure a working development environment.
2. Understand and use browser DevTools effectively.
3. Create and publish their first web page.
4. Apply Git and GitHub for version control and project hosting.

1 Environment Setup

1. Install VS Code and add extensions such as Live Server and Prettier.
2. Install Git and configure your username and email.
3. Create a GitHub account.
4. Clone a sample repository from GitHub and open it in VS Code.

2 First HTML Page

1. Create a file named index.html.
2. Add the following elements:

(a) A page title using the <title> tag.
(b) A heading with your name.
(c) A paragraph introducing yourself.
(d) An external link to your favorite website.

3. Open the page in a browser.

3 Browser DevTools Exploration

1. Open your index.html file in Chrome or Firefox.
2. Right-click and select Inspect to open DevTools.
3. Modify the heading text directly in the Elements panel.
4. Change the paragraph color using the Styles panel.
5. Switch to the Network tab and reload the page to observe requests.

4 Git Basics

1. Initialize a Git repository in your project folder using git init.
2. Stage and commit your index.html file with:

```
git add.
git commit -m "Created first HTML page"
```

3. Create a repository on GitHub named frontend-lab1.
4. Push your local repository to GitHub.

5 Mini Project – Course Introduction Page

1. Create a web page that includes:

(a) A heading: “Frontend Development Course”.
(b) A paragraph describing the course in your own words.
(c) An unordered list (<ul>) of at least three things you expect to learn.
(d) A footer with your name and the date.

2. Upload this project to GitHub.
3. (Optional) Enable GitHub Pages and share the hosted link.

EXERCISES: 2 - HTML FUNDAMENTALS

Learning Outcome
By completing these exercises, students will:
1. Write valid and structured HTML documents.
2. Use semantic tags for better readability and accessibility.
3. Design forms with various input types and labels.
4. Create and format tables with headers and merged cells.
5. Integrate multimedia elements into web pages.
6. Develop a portfolio skeleton page as a foundation for future projects.

1 HTML Structure Practice
1. Create a new HTML5 file with the proper <!DOCTYPE html> declaration.
2. Add the basic structure including <html>, <head>, and <body>.
3. Inside the body, include:
(a) A main heading (<h1>) and a subheading (<h2>).
(b) A paragraph describing yourself.
(c) An ordered list of three favorite hobbies.
(d) An unordered list of three favorite foods.
(e) A hyperlink to your favorite website.

2 Semantic HTML Page
1. Create a blog-style page using semantic tags.
2. Include:
(a) A <header> with the blog title.
(b) A <nav> with three links (Home, About, Contact).
(c) A <main> containing one <article> with a heading and two paragraphs.
(d) An <aside> with a short author bio.
(e) A <footer> with your name and year.

3 Forms in HTML
1. Design a registration form with the following fields:
(a) Name (text input).
(b) Email (email input).
(c) Password (password input).
(d) Gender (radio buttons).
(e) Hobbies (checkboxes).
(f) Country (dropdown menu).
2. Add a Submit and Reset button.
3. Ensure each input has a corresponding label.

4 Tables
1. Create a student marks table with the following columns:
(a) Name
(b) Subject
(c) Marks
(d) Grade
2. Include table headers using <th>.
3. Use colspan and rowspan in at least one place.
4. Add borders and cell padding for readability.

5 Multimedia Integration
1. Insert an image with appropriate alt text.
2. Embed a YouTube video using <iframe>.
3. Add an audio clip with playback controls.

6 Mini Project - Portfolio Skeleton
1. Create the skeleton of a personal portfolio page using only HTML.
2. The page should include:
(a) An <header> with your name and a navigation menu.

(b) An <section> for skills (list of at least 5 skills).
(c) An <section> for education (table format with institution, degree, year).
(d) A <section> with a simple contact form (Name, Email, Message).
(e) A <footer> with your name and copyright.

EXERCISES: 3 - CSS FUNDAMENTALS

Learning Outcome

By completing these exercises, students will:
1. Apply CSS to style HTML elements using inline, internal, and external methods.
2. Use various CSS selectors to target elements effectively.
3. Understand and manipulate the CSS box model.
4. Style text, lists, and links for better readability and design.
5. Create visually appealing personal pages using CSS.

1 Applying Basic CSS Styles

1. Create an HTML page with headings, paragraphs, and lists.
2. Apply CSS styles using:
(a) Inline CSS for one element.
(b) Internal CSS using <style> tag.
(c) External CSS by linking a separate CSS file.
3. Change the following styles:
(a) Text color, background color, and font size.
(b) Font family and font style (italic, bold).
(c) Text alignment (left, center, right).

2 CSS Selectors

1. Use different types of CSS selectors:
(a) Element selector
(b) Class selector
(c) ID selector
(d) Group selector
(e) Descendant and child selectors
2. Apply distinct styles to demonstrate each selector.

3 Box Model Practice

1. Create a few <div> boxes with content.
2. Apply the following CSS properties:
(a) Width and height
(b) Padding, margin, and border
(c) Background color and border style
3. Observe how the box model affects layout and spacing.

4 Typography and Colors

1. Experiment with font families, sizes, weights, and styles.
2. Apply different text colors and background colors.
3. Use text decoration (underline, overline, line-through).
4. Experiment with letter-spacing and line-height.

5 Lists and Links Styling

1. Create ordered and unordered lists.
2. Change bullet styles for unordered lists and numbering styles for ordered lists.
3. Style links with different pseudo-classes:
(a) :link
(b) :visited
(c) :hover
(d) :active

6 Mini Project – Styled Personal Page

1. Take the HTML page created in Module 2 mini project (Portfolio Skeleton).
2. Apply CSS to improve styling:
(a) Set colors, fonts, and text alignment.
(b) Apply padding, margin, and borders for sections.
(c) Style the navigation menu and links with hover effects.
(d) Add background colors or images to sections.

EXERCISES: 4 - CSS LAYOUTS

Learning Outcome

By completing these exercises, students will:
1. Understand and apply different CSS positioning methods.
2. Use display and float properties effectively.
3. Build layouts using Flexbox and Grid techniques.
4. Make web pages responsive for multiple screen sizes using media queries.
5. Create complex, structured, and adaptive web page layouts.

1 Positioning Practice

1. Create an HTML page with multiple <div> elements.
2. Apply different CSS positioning properties:
(a) static (default)
(b) relative
(c) absolute
(d) fixed
(e) sticky
3. Observe how each positioning type affects layout.

2 Display and Float

1. Experiment with display properties:
(a) block
(b) inline
(c) inline-block
(d) none
2. Use float to align elements to the left or right.
3. Clear floats using the clear property.

3 Flexbox Layout

1. Create a container with multiple child boxes.
2. Apply Flexbox properties on the container:
(a) display: flex;
(b) flex-direction (row, column, row-reverse, column-reverse)
(c) justify-content (flex-start, center, space-between, space-around)
(d) align-items (flex-start, center, stretch)
(e) flex-wrap (wrap, nowrap)
3. Style individual flex items using order, flex-grow, and flex-shrink.

4 CSS Grid Layout

1. Create a container with multiple boxes.
2. Apply Grid layout properties:
(a) display: grid;
(b) Define columns and rows using grid-template-columns and grid-template-rows.
(c) Set gaps using grid-gap.
(d) Position items using grid-column and grid-row.
(e) Experiment with auto-fit and auto-fill for responsive grids.

5 Responsive Design with Media Queries

1. Create a page with a header, sidebar, content area, and footer.
2. Use media queries to modify layout for different screen sizes:
(a) Make sidebar collapse below content on small screens.
(b) Change font size and padding for mobile devices.
(c) Adjust grid or flex layouts for tablet and mobile views.

6 Mini Project - Responsive Layout Page

1. Design a webpage (e.g., a simple blog or portfolio page) with:
(a) Header with navigation menu
(b) Sidebar for links or information
(c) Main content area
(d) Footer
2. Apply Flexbox or Grid for the layout.
3. Make the page responsive using media queries.

EXERCISES: 5 - ADVANCED CSS & UI DESIGN

Learning Outcome

By completing these exercises, students will:
1. Apply transitions, transforms, and animations to enhance user interfaces.
2. Use pseudo-classes and pseudo-elements for styling.
3. Implement CSS variables for maintainable and consistent styles.
4. Utilize CSS frameworks to quickly build responsive and styled components.
5. Create visually appealing and interactive web pages.

1 CSS Transitions

1. Create a page with multiple buttons or boxes.
2. Apply a transition effect on hover:
(a) Change background color smoothly.
(b) Change text color smoothly.
(c) Apply multiple property transitions (e.g., color and border).
3. Experiment with transition-duration and transition-timing-function.

2 CSS Transforms

1. Use transform properties on images or boxes:
(a) scale() to resize elements.
(b) rotate() to rotate elements.
(c) translate() to move elements.
(d) skew() to tilt elements.
2. Combine multiple transforms for creative effects.

3 CSS Animations

1. Create keyframe animations for a box or image:
(a) Animate position (left, top).
(b) Animate background color or text color.
(c) Animate scale or rotation.
2. Control animation using animation-duration, animation-delay, and animation-iteration-count.

4 Pseudo-classes and Pseudo-elements

1. Style links with pseudo-classes:
(a) :hover
(b) :focus
(c) :active
2. Use pseudo-elements to enhance design:
(a) ::before to add content before elements.
(b) ::after to add content after elements.
3. Create decorative effects like underlines, icons, or background shapes.

5 CSS Variables and Custom Properties

1. Define CSS variables for colors, fonts, and spacing.
2. Use variables throughout the stylesheet for consistency.
3. Change variable values and observe effects on the page.

6 Introduction to CSS Frameworks

1. Use Bootstrap or TailwindCSS to style a simple page:
(a) Create a navigation bar.
(b) Create a responsive card layout with images and text.
(c) Add buttons with framework classes.

(d) Apply spacing and color utilities from the framework.

7 Mini Project - Styled Responsive Page
1. Take your previous portfolio or layout page.
2. Apply advanced CSS techniques:
(a) Add transitions and hover effects to buttons and links.
(b) Use transforms and animations for interactive sections.
(c) Apply pseudo-elements for decorative effects.
(d) Use CSS variables for consistent styling.
(e) Optionally, implement Bootstrap/Tailwind components.

EXERCISES: 6 - JAVASCRIPT BASICS

Learning Outcome
By completing these exercises, students will:
1. Understand basic JavaScript syntax, variables, and data types.
2. Perform operations using operators and expressions.
3. Write functions with parameters and return values.
4. Manipulate the DOM to change content and styles dynamically.
5. Implement event-driven interactions to make web pages interactive.
6. Identify and fix JavaScript errors using the browser console.
7. Understand the difference between log, warn, error, and debugger.
8. Gain confidence in inspecting variables, functions, and DOM elements in real-time.
9. Develop practical interactive projects such as a To-Do List and a Calculator.

1 Task : JavaScript Setup and Basic Output
1. Create a new HTML file and link an external JavaScript file.
2. Display a message using:
(a) alert()
(b) console.log()
(c) Writing to the document using document.write()
3. Experiment with comments and whitespace in JavaScript.

2 Variables and Data Types
1. Declare variables using var, let, and const.
2. Work with different data types: string, number, boolean, null, undefined.
3. Practice type conversion between strings and numbers.
4. Print the variables and their types in the console.

3 Operators and Expressions
1. Perform arithmetic operations (+, -, *, /,
2. Use comparison operators (==, ===, !=, !==, ¿, ¡, ¿=, ¡=).
3. Practice logical operators (&&, ——, !).
4. Display results using console.log().

4 Functions and Scope
1. Create functions to perform simple calculations (e.g., sum of two numbers).
2. Pass parameters to functions and return values.
3. Explore function scope and local vs global variables.
4. Call functions from HTML buttons using onclick.

5 DOM Manipulation
1. Access HTML elements using:
(a) document.getElementById()
(b) document.getElementsByClassName()
(c) document.querySelector()
2. Change element content using innerHTML.
3. Change element styles dynamically.
4. Add or remove elements from the DOM.

6 Event Handling
1. Create HTML buttons or input fields.

2. Add event listeners using onclick, onmouseover, onchange.
3. Change content or style in response to events.
4. Create a simple interactive feature, e.g., button changes text color on click.

7 Debugging with Browser Console

1. Open your HTML page with linked JavaScript in the browser.
2. Write a small JavaScript program with intentional errors, for example:
(a) Using an undefined variable.
(b) Performing invalid arithmetic (e.g., divide by zero).
(c) Misspelling a function name.
3. Use console.log() to display variable values at different stages.
4. Use console.warn() and console.error() to categorize messages.
5. Insert the debugger statement in a function and observe code execution step by step in the console.
6. Use console.table() to display an array of objects (e.g., students with name and marks).
7. Inspect and fix the errors by reading the console messages.

8 Mini Projects

Project 1: Interactive To-Do List

(a) Create an input field for new tasks and a “Add Task” button.
(b) Display the task list dynamically using an unordered list (<ul>).
(c) Add a ”Delete” button next to each task to remove it.
(d) Change the style of completed tasks when clicked (e.g., strikethrough).
(e) Use functions, event listeners, and DOM manipulation for all actions.

Project 2: Simple Calculator

(a) Create a basic calculator layout with input fields for numbers and buttons for operations (+, -, *, /).
(b) Display the result dynamically on the page.
(c) Use JavaScript functions to perform calculations.
(d) Handle invalid input (e.g., division by zero) gracefully.
(e) Enhance interactivity using event listeners and dynamic updates.

EXERCISES: 7 - JAVASCRIPT CONTROL FLOW

Learning Outcome

By completing these exercises, students will:
1. Manipulate arrays and objects effectively in JavaScript.
2. Implement loops and conditional statements for various tasks.
3. Use event listeners to create interactive web pages.
4. Validate forms dynamically and provide user feedback.
5. Develop practical mini-projects including a Calculator, Form Validator, and Student Grades Dashboard.

1 Working with Arrays

1. Create an array of numbers and:
(a) Find the sum and average of all elements.
(b) Find the largest and smallest numbers.
(c) Sort the array in ascending and descending order.
(d) Reverse the array.

2. Create an array of strings (names of students) and:
(a) Add and remove elements at the beginning and end.
(b) Search for a particular name using a loop.
(c) Join all names into a single string using join().

2 Working with Objects

1. Create an object representing a student with properties: Name, Age, Marks (array).
2. Access and modify object properties using dot and bracket notation.
3. Loop through the object properties using for...in and display the values.
4. Create an array of student objects and:
(a) Calculate each student’s total and average marks.
(b) Find the student with the highest total marks.
(c) Display all student details dynamically in a table.

3 Loops and Conditionals
1. Use for, while, and do-while loops to:
(a) Print numbers 1–50, even numbers, and odd numbers.
(b) Generate a multiplication table for a given number.

2. Use if-else to:
(a) Check whether a number is positive, negative, or zero.
(b) Determine eligibility for voting based on age input.
(c) Check if a number is prime.

3. Use switch to:
(a) Display day of the week based on a number (1–7).
(b) Display month name based on a number (1–12).

4 Event Listeners
1. Add a click event to a button to display an alert message.
2. Add a mouseover and mouseout event to change the color of a paragraph.
3. Create multiple buttons dynamically and attach a click event to each using event delegation.
4. Use a keyup event to dynamically display the number of characters typed in an input field.

5 Form Validation
1. Create a registration form with Name, Email, Password, Age fields.
2. Validate:
(a) Name is not empty.
(b) Email contains @ and a valid domain.
(c) Password length is at least 6 characters.
(d) Age is between 18 and 60.
3. Display success or error messages dynamically without refreshing the page.
4. Use loops and conditionals to check multiple inputs at once.

6 Mini-Projects
Project 1: Calculator
(a) Create a simple calculator with two input fields and buttons for +, -, *, /.
(b) Use functions, conditionals, and event listeners to perform operations.
(c) Display results dynamically on the page.
(d) Add validation for empty inputs or invalid operations.

Project 2: Form Validator
(a) Build a registration form with multiple input fields.
(b) Validate inputs using JavaScript event listeners and conditionals.
(c) Display inline error messages dynamically.
(d) Prevent form submission if validation fails using event.preventDefault().

Project 3: Student Grades Dashboard
(a) Create an array of student objects with Name, Marks.
(b) Calculate total and average marks using loops.
(c) Display results in a table with conditional formatting (e.g., highlight students with marks ¿ 80).
(d) Allow adding a new student dynamically using a form and update the table.

EXERCISES: 8 - JAVASCRIPT ES6+ FEATURES

Learning Outcome
By completing these exercises, students will:
1. Differentiate between var, let, and const.
2. Write cleaner code using arrow functions.
3. Use template literals and destructuring for concise syntax.
4. Apply spread and rest operators in arrays, objects, and functions.
5. Organize code into modules and import/export them.
6. Build practical mini-projects using modern ES6+ features.

1 let and const
1. Declare variables using var, let, and const, and test scope differences inside a block.

2. Try reassigning a const variable and observe the error message.
3. Create a constant object and:
(a) Add new properties.
(b) Modify existing properties.
(c) Attempt to reassign the whole object and record the result.
4. Use let in a loop to correctly capture the loop index in a closure and compare it with var.

2 Arrow Functions
1. Convert a traditional function to an arrow function:
(a) Function to square a number.
(b) Function to calculate factorial using a loop.
(c) Function to return the maximum of two numbers.
2. Use arrow functions with map(), filter(), and reduce():
(a) Double each element in an array.
(b) Filter even numbers from an array.
(c) Calculate the sum of all elements in an array.

3. Demonstrate how arrow functions handle this differently than regular functions inside an object.

3 Template Literals and Destructuring
1. Use template literals to:
(a) Print a multi-line string.
(b) Embed expressions like addition or string concatenation inside a string.
(c) Create a string that displays user details dynamically (name, age, city).

2. Apply array destructuring:
(a) Extract first, second, and remaining elements from an array.
(b) Swap two numbers without using a temporary variable.

3. Apply object destructuring:
(a) Extract properties from a student object.
(b) Use default values when a property is missing.
(c) Destructure nested objects (e.g., address inside student).

4 Spread and Rest Operators
1. Use the spread operator to:
(a) Clone an array.
(b) Merge two arrays into a new one.
(c) Spread a string into an array of characters.

2. Use the spread operator with objects:
(a) Merge two objects.
(b) Override properties in the target object.

3. Use the rest operator to:
(a) Write a function that accepts variable number of arguments and returns their sum.
(b) Collect remaining array elements during destructuring.

5 Modules and Imports
1. Create a module file mathUtils.js that exports:
(a) A constant PI.
(b) Functions for addition, subtraction, multiplication, division.

2. Import the module in app.js using:
(a) Named imports.
(b) Import all as a single object.
(c) Default export for a special function.

3. Create another module studentUtils.js that exports:
(a) A function to calculate average marks.
(b) A function to find the top student from an array.

4. Import both mathUtils.js and studentUtils.js into main.js and demonstrate usage.

6 Mini-Projects
Project 1: Expense Tracker (using ES6 features)
(a) Maintain a list of expenses as objects with name, amount, and date.
(b) Use array methods and arrow functions to calculate total expenses.
(c) Display expense details using template literals.
(d) Use spread operator to add new expenses dynamically.

Project 2: Weather App Data Parser
(a) Create a sample weather data object with nested details.
(b) Use destructuring to extract temperature, humidity, and location.
(c) Use template literals to generate a formatted weather report.

Project 3: Modular To-Do List
(a) Create a todo.js module that exports functions to add, remove, and list tasks.
(b) In main.js, import these functions and build a simple text-based To-Do app.
(c) Use arrow functions, rest parameters, and template literals where appropriate.

EXERCISES: 9 - ASYNCHRONOUS JAVASCRIPT

Learning Outcome
At the end of this lab session, students will be able to:
1. Implement callback functions to manage asynchronous execution.
2. Develop programs using Promises to handle success and error states.
3. Apply async/await to simplify asynchronous workflows.
4. Use the Fetch API to retrieve data from external services.
5. Parse and manipulate JSON data in JavaScript applications.
6. Design and build a mini project (e.g., Weather App or API-based application) that integrates multiple asynchronous concepts.

1 Understanding Callbacks
1. Write a function that accepts a callback and prints a message before and after executing it.
2. Simulate fetching user data using setTimeout() and handle the result with a callback.

2 Promises in Action
1. Create a Promise that resolves after 2 seconds with a success message.
2. Create a Promise that rejects with an error and handle it using .catch().
3. Chain multiple .then() calls to transform the result step by step.

3 Using async/await
1. Rewrite the Promise from Exercise 2 using async/await.
2. Create an async function that waits for multiple Promises using Promise.all().
3. Introduce error handling using try...catch with async functions.

4 Fetch API Basics
1. Use the Fetch API to request data from a public API (e.g., https://jsonplaceholder.typicode
2. Parse the response as JSON and log it to the console.
3. Handle network errors gracefully.

5 Working with JSON
1. Convert a JavaScript object into a JSON string using JSON.stringify().
2. Parse a JSON string back into an object using JSON.parse().
3. Fetch a JSON file and dynamically display its data in an HTML table.

6 Mini Projects
Project 1: Weather App
(a) Build a simple weather application using the OpenWeatherMap API (or any free weather API).
(b) Allow the user to enter a city name in an input field.
(c) Fetch current weather data using the Fetch API.
(d) Display temperature, humidity, and weather description dynamically on the page.
(e) Handle errors for invalid city names or failed API requests.

Project 2: Random Joke Generator - API-based App (Alternative)
(a) Create a ”Random Joke Generator” app using a public jokes API (e.g., https://official-joke-api.appspot.com).
(b) Add a button that, when clicked, fetches a new joke using async/await.
(c) Display the joke (setup and punchline) dynamically in the DOM.
(d) Style the app for a clean UI and handle network errors.

EXERCISES: 10 - JAVASCRIPT DOM MANIPULATION

Learning Outcome
By completing these exercises, students will:
1. Select and manipulate DOM elements using JavaScript.
2. Attach and handle different types of events.
3. Dynamically create, append, and remove elements in the DOM.
4. Modify element styles and manage CSS classes.
5. Traverse and explore DOM hierarchies.
6. Build interactive mini-projects that use DOM manipulation.

1 Selecting and Modifying Elements

1. Create an HTML page with a heading, a paragraph, and a button.
2. Use document.getElementById, document.querySelector, and document.querySelecto
to select elements and log them to the console.
3. Change the text of the heading and the color of the paragraph using JavaScript.
4. Modify the button label dynamically.

2 Event Handling

1. Attach a click event listener to the button that changes the background color of the page.
2. Add a double-click event to toggle between two colors.
3. Create a text input field and use the input event to display the typed text live in a paragraph below it.
4. Add a mouseover event on the heading to increase its font size temporarily.

3 Creating and Appending Elements

1. Write a script to dynamically create a list of fruits (ul with li items) and add it to the page.
2. Add a button that appends a new fruit item when clicked.
3. Create a simple form (name and email) and, upon submission, dynamically add the entered details to a list.

4 Modifying Styles and Classes

1. Create a paragraph and two buttons: “Highlight” and “Remove Highlight”.
2. Use classList.add, classList.remove, and classList.toggle to apply/remove a CSS highlight class.
3. Change multiple CSS properties dynamically using element.style.
4. Experiment with hiding and showing elements using display property.

5 DOM Traversal

1. Create a nested HTML structure with a parent div and child elements.
2. Use parentNode, children, firstElementChild, and lastElementChild to explore the structure.
3. Traverse sibling elements using nextElementSibling and previousElementSibling.

6 Mini-Projects

Project 1: To-Do List App
(a) Create an input box and “Add Task” button.
(b) On button click, add the entered task as a new list item.
(c) Add “Delete” buttons for each task.
(d) Use CSS classes to mark tasks as completed on click.

Project 2: Image Gallery
(a) Display a row of thumbnail images.
(b) On clicking a thumbnail, display the full image in a larger view section.
(c) Add “Next” and “Previous” buttons to navigate images.

Project 3: Interactive Counter
(a) Add “+” and “-” buttons and a display showing the counter value.
(b) Increment or decrement the value when the buttons are clicked.
(c) Add a “Reset” button to set the counter back to 0.

EXERCISES: 11 - INTRODUCTION TO REACT

Learning Outcome

By completing these exercises, students will:

1. Understand the difference between SPA and MPA, and the role of frameworks.
2. Set up and run a React development environment.
3. Create and render components using JSX.
4. Pass data between components using props.
5. Manage component-level state using useState.
6. Use React Developer Tools for debugging and inspection.
7. Build small interactive apps with React.

1 Understanding Frameworks and SPA vs MPA

1. Research and write a short note comparing Single Page Applications (SPA) and Multi Page Applications (MPA).
2. Identify three popular frameworks/libraries used for building SPAs and explain why they are preferred.
3. Create a small HTML file that simulates an MPA (two separate pages linked together).
4. Convert it into a basic SPA simulation by showing and hiding sections with JavaScript.

2 React Setup and Hello World

1. Set up a new React project using Vite or Create React App.
2. Create a simple React component called HelloWorld that displays a greeting.
3. Render this component inside the App component.
4. Experiment by modifying the greeting text and observing live reload.

3 Components and JSX

1. Create two new components: Header and Footer.
2. Use JSX to include HTML-like syntax within your components.
3. Render Header, a content section, and Footer inside the App component.
4. Add inline styles using JSX style objects and test different formatting.

4 Props in React

1. Create a component UserCard that accepts name, age, and email as props.
2. Render multiple UserCard components in App with different user details.
3. Experiment with passing strings, numbers, and even JSX as props.
4. Add default props to handle missing values.

5 State in React

1. Create a component Counter with a button to increment a number using useState.
2. Add another button to decrement the number.
3. Extend the component to reset the counter to 0.
4. Experiment with updating state based on the previous state value.

6 React Developer Tools

1. Install React Developer Tools in your browser.
2. Open the DevTools and inspect your React application.
3. Explore the component tree and observe how props and state change in real time.
4. Debug a simple bug (e.g., missing prop or incorrect state update) using DevTools.

7 Mini-Projects

Project 1: Profile Card App
(a) Build a ProfileCard component with props for name, role, and image.
(b) Render multiple profile cards in a grid layout.

Project 2: Simple Counter App
(a) Build a standalone counter app using useState.
(b) Include increment, decrement, and reset functionality.
(c) Style the counter using JSX and CSS.

Project 3: Greeting App
(a) Create a form with an input field for the user’s name.
(b) Display a personalized greeting message using React state.
(c) Add validation for empty input.

EXERCISES: 12 - REACT ADVANCED CONCEPTS

Learning Outcome

By completing these exercises, students will:
1. Understand and apply React Hooks like useState and useEffect.
2. Handle and validate forms in React using controlled components.
3. Implement navigation and routing in a React application using react-router-dom.
4. Perform API calls within React, handle asynchronous data, and manage loading/error states.
5. Build advanced React applications that combine hooks, routing, and API integration.

1 Exploring useState Hook

1. Create a component CounterApp that uses useState to track a counter value.
2. Add buttons for increment, decrement, and reset functionality.
3. Modify the app to track multiple counters (e.g., Counter A and Counter B).
4. Experiment with updating state using both direct values and callback functions.

2 Exploring useEffect Hook

1. Create a component that displays the current time and updates every second using useEffect.
2. Build a simple app where a message is logged in the console whenever a button is clicked (demonstrating dependency arrays).
3. Implement a cleanup function inside useEffect to stop an interval when a component unmounts.
4. Observe how dependency arrays affect the re-rendering of components.

3 Handling Forms in React

1. Create a form with input fields: name, email, and password.
2. Use controlled components to manage input values with useState.
3. Display the entered data dynamically below the form.
4. Add simple validation (e.g., required fields, email format check).

4 Navigation with React Router

1. Install and configure react-router-dom.
2. Create three components: Home, About, and Contact.
3. Set up routes for each component and add navigation links.
4. Implement a “Not Found” page for invalid routes.
5. Add route parameters (e.g., /user/:id) and display user details dynamically.

5 API Calls in React

1. Create a component that fetches and displays a list of posts from the JSONPlaceholder API (https://jsonplaceholder.typicode.com/posts).
2. Use useEffect to make the API call when the component mounts.
3. Display the data in a formatted list or card layout.
4. Add a loading indicator and error handling.
5. Extend the component to fetch a single post based on an ID entered by the user.

6 Mini-Projects

1. Project 1: To-Do List with Hooks
(a) Build a to-do list using useState to store tasks.
(b) Add features: add, delete, and mark tasks as completed.
(c) Persist tasks in LocalStorage.

2. Project 2: Multi-Page Portfolio with React Router
(a) Create a portfolio app with Home, Projects, and Contact pages.
(b) Use React Router for navigation.
(c) Add route parameters for displaying project details.

3. Project 3: API-Based News App
(a) Fetch news articles from a public API.
(b) Display articles in cards with title, description, and link.
(c) Add navigation between categories (e.g., Technology, Sports, Business) using React Router.
(d) Show loading and error messages appropriately.

EXERCISES: 13 - STYLING IN REACT & PROJECT WORK

Learning Outcome

By completing these exercises, students will:

1. Apply different styling techniques in React using CSS Modules, styled-components, and TailwindCSS.
2. Integrate component libraries such as Material UI and shadcn/ui into React projects.
3. Build responsive, user-friendly interfaces with modern styling approaches.
4. Develop polished mini-projects (To-Do App, Portfolio, Theme Switcher) showcasing advanced styling skills.

1 Styling with CSS Modules

1. Create a simple React component (ProfileCard) and style it using a CSS Module.
2. Apply styles for headings, paragraphs, and a profile image.
3. Demonstrate scoping by creating another component that uses a different CSS Module with the same class names.
4. Modify styles dynamically based on props (e.g., highlight a card if it's marked as "featured").

2 Styled Components

1. Install and configure styled-components.
2. Create a button component with different styles for primary and secondary variants.
3. Apply conditional styling using props (e.g., disabled state).
4. Build a reusable card component styled entirely with styled-components.

3 TailwindCSS Basics

1. Set up TailwindCSS in a React project.
2. Build a responsive navigation bar using Tailwind utility classes.
3. Style a login form with Tailwind, ensuring proper spacing, colors, and hover effects.
4. Experiment with responsive design by adjusting layouts for mobile, tablet, and desktop.

4 Using Component Libraries

1. Install Material UI and build a form with pre-designed components (e.g., TextField, Button, Card).
2. Use icons from Material UI in the application.
3. Install and configure shadcn/ui in a React project.
4. Build a dashboard layout using cards, buttons, and navigation components from shadcn/ui.

5 Mini-Projects

Project 1: Styled To-Do App

(a) Build a To-Do list with add, delete, and mark-as-complete functionality.
(b) Apply styling using TailwindCSS or styled-components.
(c) Add filters for "All", "Completed", and "Pending" tasks.
(d) Store tasks in LocalStorage for persistence.

Project 2: Portfolio Website

(a) Create a personal portfolio site with sections: Home, About, Projects, and Contact.
(b) Use Material UI or shadcn/ui for layout and components.
(c) Style each section with either CSS Modules or TailwindCSS.
(d) Add responsiveness to ensure mobile and desktop compatibility.

Project 3: Theme Switcher App

(a) Build a React app with a light/dark theme toggle.
(b) Use styled-components or TailwindCSS for theming.
(c) Save the user's theme preference in LocalStorage.
(d) Apply the theme globally across multiple components.

EXERCISES: 14 - CAPSTONE PROJECT & DEPLOYMENT

Learning Outcome

By completing these exercises, students will:

1. Integrate all JavaScript and React concepts into a full-fledged project.
2. Collaborate effectively using Git and GitHub.
3. Gain hands-on experience in deploying React applications on modern platforms.
4. Present and document their project in a professional manner.

1 Project Planning and Setup

1. Brainstorm and finalize an idea for the capstone project (e.g., E-commerce site, Blogging platform, Social media dashboard, etc.).

2. Create a project structure in React using Vite or Create React App.
3. Define the core features, assign roles if working in teams, and document the plan in a README file.

2 Version Control with Git and GitHub
1. Initialize a Git repository for the project.
2. Create a remote repository on GitHub and push the initial code.
3. Practice branching by creating feature branches for different modules.
4. Perform pull requests and merge changes into the main branch.
5. Add a .gitignore file to exclude unnecessary files.

3 Building the Capstone Project
1. Integrate key React concepts: components, props, state, hooks, and routing.
2. Connect to an external API or a backend service for data (optional).
3. Implement form handling, error handling, and validation.
4. Style the project using TailwindCSS, styled-components, or Material UI.
5. Add authentication or local storage features if applicable.

4 Deployment
1. Prepare the project for production build.
2. Deploy the project on one platform:
(a) Netlify – connect GitHub repo and auto-deploy on push.
(b) Vercel – use CLI or GitHub integration for deployment.
(c) GitHub Pages – configure and deploy static site build.
3. Test the deployed site on different devices/browsers.
4. Document deployment steps in the project README.

5 Final Project Presentation
1. Prepare a short presentation covering:
(a) Project idea and objectives.
(b) Key features implemented.
(c) Technical stack used.
(d) Challenges faced and solutions.
2. Showcase a live demo of the deployed project.
3. Submit the project repository link, deployment link, and presentation slides.

6 Capstone Project Ideas
1. Student Management System Build a full-stack web application to manage student data.
(a) User authentication (admin, teacher, student).
(b) CRUD operations for student records (add, edit, delete, view).
(c) Search and filter functionality.
(d) Report generation in PDF/Excel format.

2. E-Commerce Website Create a modern shopping website with secure checkout.
(a) Product listing with categories and filters.
(b) Shopping cart and checkout system.
(c) User authentication and order history.
(d) Payment gateway integration (dummy or sandbox).

3. Blog Platform A multi-user blogging system with interactive features.
(a) User registration and login.
(b) Create, update, delete, and read posts.
(c) Add categories, tags, and featured images.
(d) Commenting and like/dislike system.

4. Portfolio Website with CMS A portfolio site that allows non-technical users to update content.
(a) Admin panel to manage projects and experience.
(b) Contact form with email integration.
(c) Option to upload images and media.
(d) Responsive design with dark/light mode.

5. Chat Application Build a real-time chat platform for communication.
(a) User registration and login.

(b) One-to-one and group chat features.
(c) Real-time message updates using WebSockets or Firebase.
(d) Option to send emojis and files.

6. Online Quiz Platform A system where teachers can create quizzes and students can attempt them.
(a) Question bank with multiple question types.
(b) Timed quiz attempts with auto-submit.
(c) Result and score calculation with analytics.
(d) Leaderboard and ranking system.

7. Task Management Tool Develop a Kanban-style productivity app.
(a) Create, assign, and track tasks.
(b) Drag-and-drop Kanban board.
(c) Deadlines with notifications.
(d) Collaboration features with multiple users.

8. Learning Management System (LMS) A complete platform for delivering online courses.
(a) Course and lesson creation.
(b) Video lectures and file uploads.
(c) Student progress tracking.
(d) Quizzes and certificate generation.

9. Event Booking Application A system to browse and book events online.
(a) Event listing with search and filters.
(b) User registration for booking tickets.
(c) Ticket generation (QR code or unique ID).
(d) Calendar integration for upcoming events.

10. Health & Fitness Tracker Track and visualize fitness activities.
(a) User registration and personalized dashboard.
(b) Daily workout and diet logging.
(c) Graphs and charts for progress visualization.
(d) Goal setting and achievement badges.

V. TEXT BOOKS:

1. Duckett, Jon. “HTML & CSS: Design and Build Websites”, John Wiley & Sons, 1st Edition, 2011.
2. Duckett, Jon. “JavaScript and JQuery: Interactive Front-End Web Development”, John Wiley & Sons, 1st Edition, 2014.
3. Wieruch, Robin. “The Road to Learn React: Your journey to master plain yet pragmatic React.js”, Independently Published, 1st Edition, 2017

VI. REFERENCE BOOKS:

1. Robbins, Jennifer Niederst. “Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics”, O’Reilly Media, 5th Edition, 2018.
2. Crockford, Douglas. “JavaScript: The Good Parts”, O’Reilly Media, 1st Edition, 2008.
3. Thomas, Mark Tielens. “React in Action”, Manning Publications, 1st Edition, 2018.

VII. ELECTRONICS RESOURCES:

5. https://developer.mozilla.org/en-US/docs/Learn_web_development
6. https://www.freecodecamp.org
7. https://www.codecademy.com
8. https://www.frontendmentor.io
9. https://www.w3schools.com

VIII. MATERIALS ONLINE;

3. Course Content
4. Lab manual

COURSE CONTENT

COMPUTER AIDED ENGINEERING GRAPHICS

I Semester: CSE / IT
II Semester: CSE (AI&ML) / CSE(DS) / ECE / EEE / AERO / MECH / CE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
AMEE03	Foundation	L	T	P	C	CIA	SEE	Total
		1	0	2	2	40	60	100
Contact Classes: 15	Tutorial Classes: Nil	Practical Classes: 30				Total Classes: 45		

Prerequisite: There is no prerequisite required to this course

I. COURSE OVERVIEW:

Engineering Drawing is the technique that develops the ability to visualize any object with all physical and dimensional configurations. The AutoCAD software assists in preparation of drawings to carry out sophisticated design and analysis of machine components and structures. This is the foundation course for civil engineering, mechanical engineering and aeronautical engineering that are improving their technologies in the era of digital manufacturing and construction.

II. COURSE OBJECTIVES:

The students will try to learn:

I. The illustration of different objects using technical drawings using concepts of engineering drawing.
II. The standard principles of orthographic projection of objects for making technical drawings.
III. The representation of draw sectional views and pictorial views of solids.
IV. The computer aided drafting skills for producing the 2D and 3D drawings.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Demonstrate the use of draw, modify and dimension commands of AutoCAD for development of drawings used in design and analysis of structures.
CO2	Explain the constructional procedure of scales, conic sections and special curves used in engineering practices.
CO3	Utilize the principles of orthographic projection for projections of points, lines, planes and regular solids using first angle projections.
CO4	Interpret the sectional views and true shape of the section for revealing interior features of an object veal interior features of an object
CO5	Illustrate the development of surfaces for construction of storage vessels, chemical vessels, boilers, and chimneys in industrial applications
CO6	Make use of the concept of orthographic and isometric projections for converting isometric view to orthographic views and Vice-versa for engineering applications.

IV. COURE CONTENT:

MODULE – I: Introduction to engineering graphics

Principles of engineering graphics and their significance, scales, plain & diagonal, conic sections including the rectangular hyperbola, general method, cycloid, epicycloid and hypocycloid, introduction to computer aided drafting, views, commands.

MODULE – II: Orthographic projections

Principles of orthographic projections, conventions, projections of points and lines, projections of plane regular geometric figures. Computer aided orthographic projections, points, lines and planes.

MODULE – III: Projections of regular solids

Projections of regular solids, auxiliary views, sections or sectional views of right regular solids, prism. Cylinder, pyramid, cone, computer aided projections of solids, sectional views.

MODULE – IV: Development of surfaces

Development of surfaces of right regular solids, prism, cylinder, pyramid and cone, development of surfaces using computer aided drafting.

MODULE – V: Isometric projections

principles of isometric projection, isometric scale, isometric views, conventions, isometric views of lines, plane figures, simple and compound solids, isometric projection of objects having non-isometric lines. Isometric projection of spherical parts, conversion of isometric views to orthographic views and vice-versa, conventions, conversion of orthographic projection into isometric view using computer aided drafting.

V.TEXT BOOKS:

1. N.D. Bhatt; *Engineering Drawing* Charotar Publishing House PVT Ltd, 15th edition 2011.
2. K. Venugoplal; *Engineering Drawing and graphics Using AutoCAD*, 3rd edition 2007.

VI. REFERENCE BOOKS:

1. Basant Agrawal and C M Agrawal; *Engineering Drawing*, McGraw Hill, 3rd Edition 2011.
2. K L Narayana, P Kannaiah; *Engineering Drawing*, New age international (P) limited, 3rd edition, 2022.
3. M. B. Shah, B.C. Rane; Engineering *Drawing*, Pearson publications.

VII. ELECTRONIC RESOURCES:

1. https://archieve.nptel.ac.in/cources/112/103/112103019.
2. https://archieve.nptel.ac.in/cources/112/105/112105294.

VIII. MATERIALS ONLINE:

1. Course Template
2. Laboratory manual

EXERCISES ON COMPUTER AIDED ENGINEERING GRAPHICS

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Introduction to AUTOCAD

AutoCAD is a widely-used computer-aided design (CAD) software application developed by Autodesk. It has been an industry standard for drafting and designing since its inception in the early 1980s. AutoCAD provides a versatile platform for creating and editing 2D and 3D drawings and models, making it an essential tool in various fields such as architecture, engineering, construction, manufacturing, and more.

i. Install AUTO CAD
ii. Purpose and Application
iii. Interface and Tools
iv. Precision and Accuracy
v. 2D and 3D Modeling
vi. Collaboration and Sharing
vii. Customization
viii. Industry Usage
ix. Versions and Licensing

1.1 Commands

The main purpose of using commands and shortcuts in AutoCAD boils down to increased productivity. They allow you to execute functions more quickly, as you don't need to search through the entire AutoCAD interface for the right tool. You can just type the command, and the function window appears.

i. Basic Drawing Commands
ii. Editing Commands
iii. Dimensioning Commands
iv. Advanced and Miscellaneous Commands

Basic Drawing Commands:

- Line (LINE): Draws straight line segments between two points.
- Circle (CIRCLE): Creates circles by specifying a center point and radius.
- Rectangle (RECTANGLE): Constructs rectangles by defining two opposing corners.
- Arc (ARC): Draws arcs based on different methods, such as specifying start, end, and radius or center, start, and angle.

Editing Commands:

- Erase (ERASE): Deletes selected objects from the drawing.
- Copy (COPY): Copies objects to a specified location.
- Move (MOVE): Relocates selected objects to a different position.
- Trim (TRIM): Cuts selected objects at the cutting edges defined by other objects.
- Extend (EXTEND): Extends objects to meet the boundaries of other objects.

Dimensioning Commands:

- Line Dimension (DIMLINEAR): Adds linear dimensions to objects.
- Aligned Dimension (DIMALIGNED): Creates dimensions aligned with an angle of the object.
- Radial Dimension (DIMRADIUS): Add radius dimensions to arcs and circles.

- Diameter Dimension (DIMDIAMETER): Creates diameter dimensions for circles.

Advanced and Miscellaneous Commands:

- Hatch (HATCH): Fills enclosed areas with a pattern or gradient.
- Offset (OFFSET): Creates parallel copies of objects at a specified distance.
- Block (BLOCK): Defines reusable blocks (collections of objects) in the drawing.
- Insert (INSERT): Inserts predefined blocks into the drawing.
- Viewport (VPORTS): Manages viewports for layout and plotting in paper space.
- Layer (LAYER): Manages layers for organizing and controlling object visibility.

TRY: Observe Exercise 1.1 in Solid works and in Creo software.

2. Introduction to Engineering Drawing

Engineering drawing, often referred to as technical drawing or drafting, is a graphical representation of an object, system, or structure used in various fields of engineering, manufacturing, and architecture. These drawings serve as a universal language that communicates design ideas, specifications, and instructions in a precise and standardized manner.

2.1 Basic Exercises

To be proficient in engineering drawing, basic exercises are required.

i. Identify the basic tools used for drafting
ii. Types of lines
iii. Arcs
iv. Circles

2.2 Practicing the standard lettering and numbering

Practicing standard lettering and numbering in engineering drawing is crucial for creating clear, professional, and easily understandable technical drawings. Proper lettering and numbering enhance communication and ensure that your drawings convey information accurately.

The following exercises are to be practiced to become proficient in lettering and numbering.

1. Use the correct fonts
2. Maintain uniformity
3. Lettering style
4. Height and spacing

Try: The following questions are to be answered in Solid works

1. How to use correct fonts in Solid Works
2. What are the commands are used to maintain uniformity of lettering and numbering.

3. Dimensioning

Dimensioning in engineering drawing is a crucial aspect that involves adding measurements and annotations to convey the size, location, and tolerances of objects, features, and components accurately. Proper dimensioning is essential for manufacturing, construction, and other engineering processes.

3.1 Exercises on Dimensioning

1. Understanding and use of the conventional dimensioning techniques.
2. Placing the dimension lines

3. Extension lines
4. Dimensions on angles

Hint: 1. The following Fig.3.1 shows the type of dimensioning on 2D drawing.

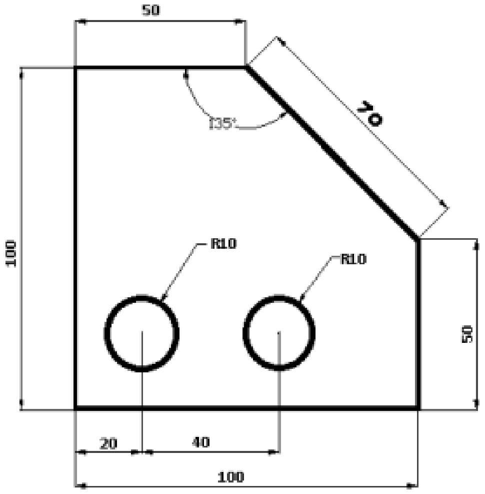


Fig.3.1 Dimensioning on 2D drawing

Hint: 2. The following Fig.3.2 shows the type of dimensioning on concentric circles.

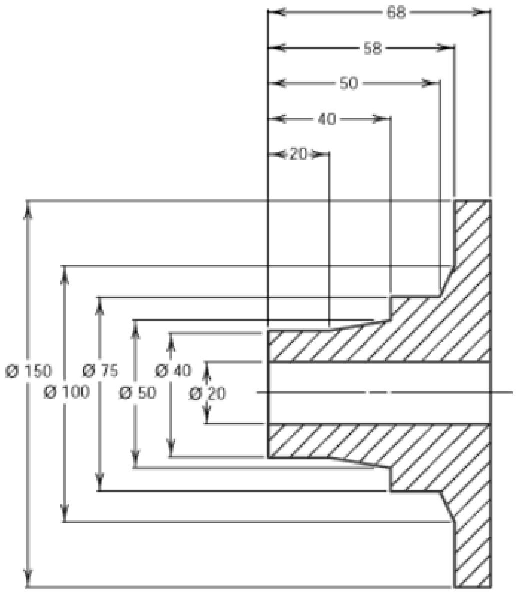


Fig.3.2 Dimensioning on Concentric circle

Try: Demonstrate the exercise 3 in Solid Works and CREO software.

4. Geometrical Constructions

Geometric construction is useful for learning how to use geometric tools like a ruler, compass, and straightedge to draw various angles, line segments, bisectors, and other forms of polygons, arcs, circles, and other geometric figures. Fig.4.1 shows the various geometric shapes to draw orthographic projections of lines, planes and solids.

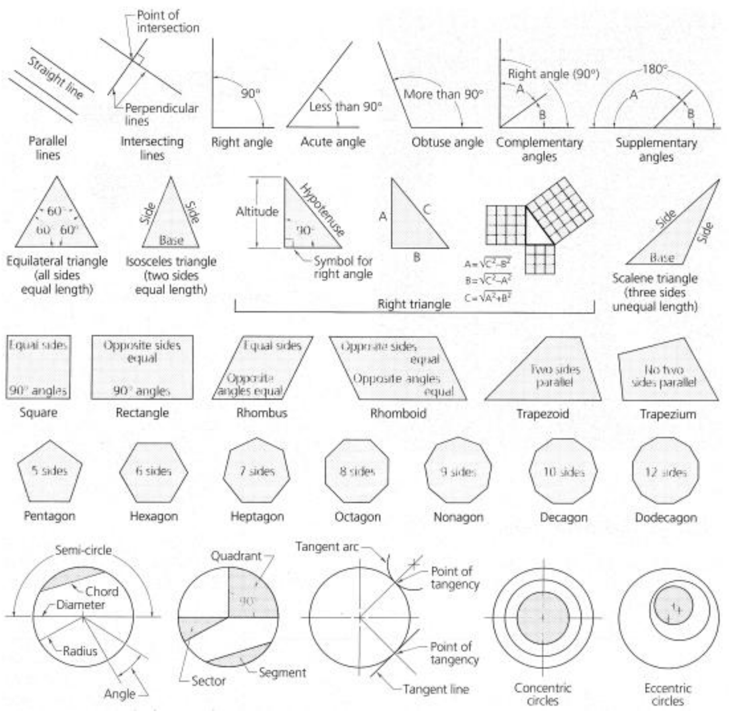


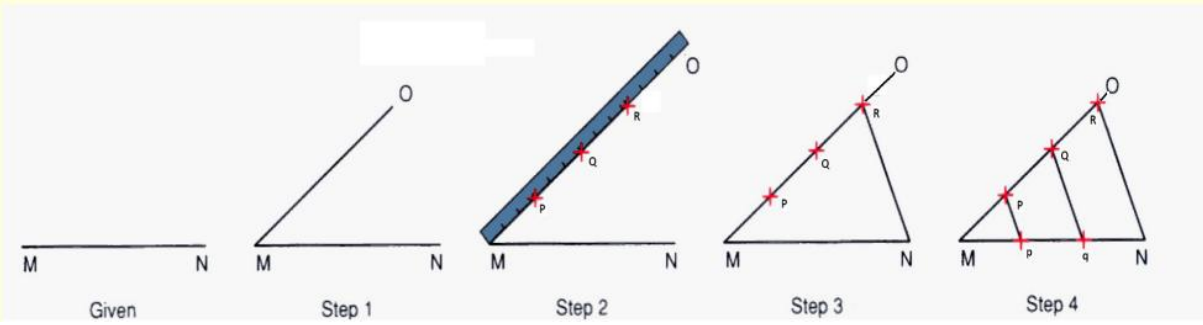
Fig.4.1 **Geometric shapes**

4.1. Exercises on Geometrical Constructions

To become proficient in engineering graphics geometrical constructions are required: Drawing lines, angles, triangle, square, pentagon, hexagon, octagon. Dividing line into equal or proportional parts. Drawing lines and arcs tangent to each other.

1. Divide a 16 cm straight line into a given number of equal parts say 5.
2. Divide a 8 cm line into 9 number of parts.
3. Bisect a given 45 degree sector.
4. Bisect a given straight line.
5. To draw a perpendicular to a given line from a point within it.
6. Construct a regular polygon, given the length of its side.

Hint: Dividing a line into equal number of parts



Try: The exercise 4 in Solid Works and CREO software.

5. Conic Sections

A conic section, conic or a quadratic curve is a curve obtained from a cone's surface intersecting a plane. The three types of conic section are the hyperbola, the parabola, and the ellipse.

5.1. Exercises on Conic Sections

1. Draw an ellipse with the distance of the focus from the directrix at 50mm and eccentricity = 2/3 (Eccentricity method)
2. Draw an ellipse with the distance of the focus from the directrix at 60mm and eccentricity = 2/3 (Eccentricity method)
3. Draw an ellipse with the distance of the focus from the directrix at 80mm and eccentricity = 2/3 (Eccentricity method)
4. Draw a parabola with the distance of the focus from the directrix at 50 mm (Eccentricity method).
5. Draw a parabola with the distance of the focus from the directrix at 40mm (Eccentricity method).
6. A vertex of a hyperbola is 60 mm from its focus. Draw two parts of the hyperbola; if the eccentricity is 3/2.
7. A vertex of a hyperbola is 50 mm from its focus. Draw two parts of the hyperbola; if the eccentricity is 1.5.

Try: The exercise 5 in Solid Works and CREO software.

6. Technical Sketching and Shape Description

6.1. Projections of planes and regular solids

1. Draw the projections of a regular pentagon of 30 mm side, having its surface inclined at 30° to the H.P. and a side parallel to the H.P. and inclined at an angle of 60° to the V.P.
2. Draw the projections of a regular hexagon of 40 mm side, having one of its sides in the H.P. and inclined at 60° to the V.P., and its surface making an angle of 45° with the H.P.
3. Draw the projections of a regular hexagon of 35 mm side, having one of its sides in the H.P. and inclined at 50° to the V.P., and its surface making an angle of 45° with the H.P.
4. Draw the projections of a regular pentagon of 40 mm side, having one of its sides in the H.P. and inclined at 30° to the V.P., and its surface making an angle of 45° with the H.P.

Try: The exercise 6.3 in Solid Works and CREO software.

7. Sectional views

A sectional view represents the part of an object remaining after a portion is assumed to have been cut and removed. The exposed cut surface is then indicated by section lines. Hidden features behind the cutting plane are omitted, unless required for dimensioning or for definition of the part.

7.1. Exercise on Sectional views of right regular solids, prism, cylinder, pyramid, cone.

1. A pentagonal pyramid, base 40 mm side and axis 60 mm long has its base horizontal and an edge of the base parallel to the V.P. A horizontal section plane cuts it at a distance of 20 mm above the base. Draw its front view and sectional top view.
2. A hexagonal prism, side of base 40 mm and height 70 mm is resting on one of its corners on the H.P. with a longer edge containing that corner inclined at 40° to the H.P. and a rectangular face parallel to the V.P. Draw the front view and sectional top view of the cut

prism when a horizontal section plane cuts the prism in two equal halves. Draw the front view and sectional top view of the cut prism.

3. A pentagonal pyramid, base 40 mm side and axis 70 mm long has one of its triangular faces in the V.P. and the edge of the base contained by that face makes an angle of 40° with the H.P. Draw its projections.
4. Draw the projections of a cone, base 50 mm diameter and axis 75 mm long, lying on a generator on the ground with the top view of the axis making an angle of 45° with the V.P.

Try: The exercise 7.1 and 7.2 in Solid Works and CREO software.

8. Development of surfaces

Knowledge of development is very useful in sheet metal work, construction of storage vessels, chemical vessels, boilers, and chimneys. Such vessels are manufactured from plates that are cut according to these developments and then properly bend into desired shaped.

8.1. Exercise on Basics of development of surfaces

1. Draw the development of the lateral surfaces of a right square prism of edge of base 30 mm and axis 50 mm long.
2. Draw the development of the complete surface of a cylindrical drum. Diameter is 40 mm and height 60 mm.

8.2. Exercise on Development of surfaces of Prisms

1. A hexagonal prism of base side 20 mm and height 45 mm is resting on one of its ends on the HP with two of its lateral faces parallel to the VP. It is cut by a plane perpendicular to the VP and inclined at 30° to the HP. The plane meets the axis at a distance of 20 mm above the base. Draw the development of the lateral surfaces of the lower portion of the prism.
2. A hexagonal prism, edge of base 20 mm and axis 50 mm long, rests with its base on HP such that one of its rectangular faces is parallel to VP. It is cut by a plane perpendicular to VP, inclined at 45° to HP and passing through the right corner of the top face of the prism. (i) Draw the sectional top view. (ii)Develop the lateral surfaces of the truncated prism.
3. A pentagonal prism, side of base 25 mm and altitude 50 mm, rests on its base on the HP such that an edge of the base is parallel to VP and nearer to the observer. It is cut by a plane inclined at 45° to HP, perpendicular to VP and passing through the center of the axis. (i) Draw the development of the complete surfaces of the truncated prism.
4. A pentagonal prism of side of base 30 mm and altitude 60 mm stands on its base on HP such that a vertical face is parallel to VP and away from observer. It is cut by a plane perpendicular to VP, inclined at an angle of 50° to HP and passing through the axis 35 mm above the base. Draw the development of the lower portion of the prism.

Try : The exercise 8.1 and 8.2 in Solid Works and CREO software.

9. Exercise on Development of surfaces-2

9.1. Exercise on Development of surfaces of cylinder and cone

1. Draw the development of the lateral surface of the lower portion of a cylinder of diameter 50 mm and axis 70 mm when sectioned by a plane inclined at 40° to HP and perpendicular to VP and bisecting axis.
2. A Cone of base diameter 60 mm and height 70 mm is resting on its base on HP. It is cut

by a plane perpendicular to VP and inclined at 30° to HP. The plane bisects the axis of the cone. Draw the development of its lateral surface.

9.2. Exercise on Development of surfaces of pyramid

1. Draw the development of the lateral surfaces of a square pyramid, side of base 25 mm and height 50 mm, resting with its base on HP and an edge of the base parallel to VP.
2. A square pyramid of base side 25 mm and altitude 50 mm rests on it base on the HP with two sides of the base parallel to the VP. It is cut by a plane bisecting the axis and inclined a 30° to the base. Draw the development of the lateral surfaces of the lower part of the cut pyramid.
3. A pentagonal pyramid side of base 30 mm and height 52 mm stands with its base on HP and an edge of the base is parallel to VP and nearer to it. It is cut by a plane perpendicular to VP, inclined at 40° to HP and passing through a point on the axis 32 mm above the base. Draw the sectional top view. Develop the lateral surface of the truncated pyramid.

Try: The exercise 9.1 and 9.2 in Solid Works and CREO software.

10. Orthographic views

Orthographic views are two-dimensional views of three-dimensional objects. Orthographic views are created by projecting a view of an object onto a plane which is usually positioned so that it is parallel to one of the planes of the object.

10.1. Exercise on Conversion of isometric view to orthographic projections using CAD

1. Draw the front view, side view and top view for the below Fig.10.1

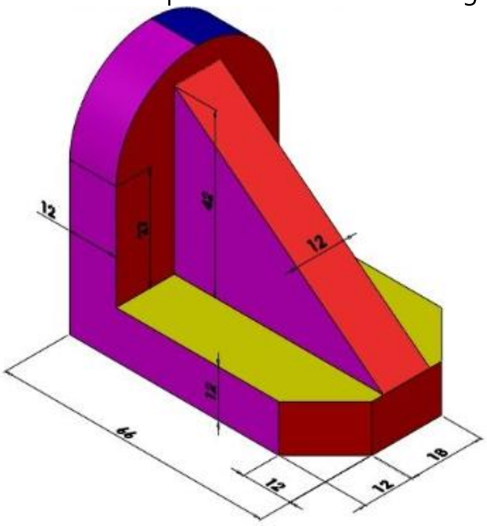


Fig.10.1

2. Draw the front view, side view and top view for the below Fig.10.2.

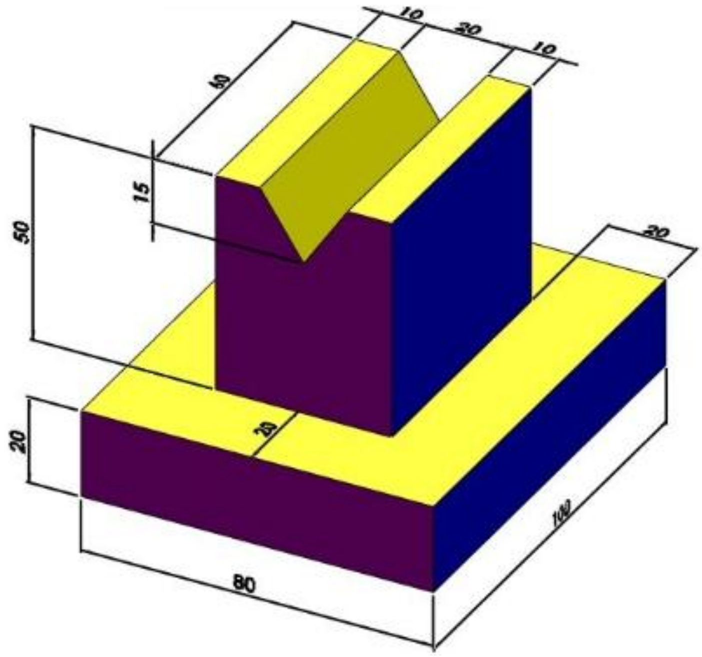


Fig.10.2

3. Draw the front view, side view and top view for the below Fig.10.3.

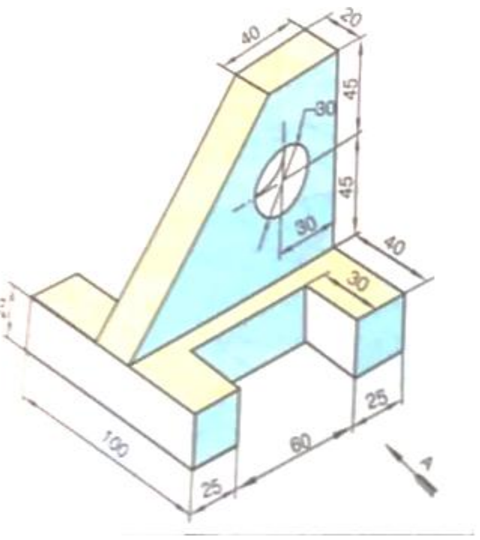


Fig.10.3

4. Draw the front view, side view and top view for the below Fig.10.4.

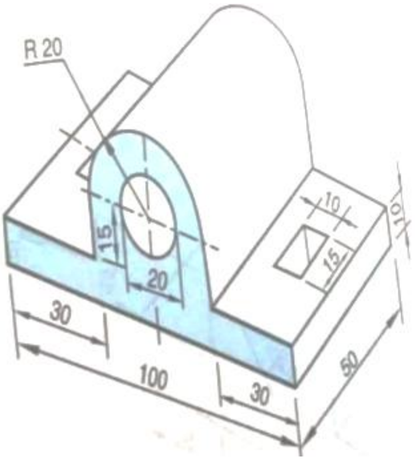


Fig.10.4

Try: Practice Exercise 10 in Solid Works

11. Isometric projection of planes

Isometric projection is a method for visually representing three-dimensional objects in two dimensions in technical and engineering drawings. It is an axonometric projection in which the three coordinate axes appear equally foreshortened and the angle between any two of them is 120 degrees.

11.1. Isometric scale

In engineering and technical drawing, an isometric scale is a method used to create an isometric projection of a three-dimensional object onto a two-dimensional surface, such as a drawing sheet or a computer screen. Isometric projection is a type of pictorial representation that shows an object in a three-dimensional view with all three principal axes (x, y, and z) at equal angles to the picture plane. An isometric scale is used to ensure that the dimensions and proportions of objects in the isometric drawing are accurate and maintain the proper relationships.

11.2. Exercise on Isometric projections of circle, square and rectangle and solids

1. Draw the isometric view of a circle of 60 mm diameter whose surface is parallel to the V.P.
2. Draw the isometric view of a square of side 60 mm whose surface is parallel to the H.P.
3. Draw the isometric view of a circle of 40 mm radius whose surface is parallel to the H.P.
4. Draw the isometric view of a square prism, side of the base 20 mm long and the axis 40 mm long, when its axis is i) Veritcal and ii) Horizontal.
5. Draw the isometric view of the semi-circle whose front view of its surface is parallel to the V.P.. The diameter of semi-circle is 60 mm.

Try : The exercise 11.2 in Solid Works and CREO software.

12. Isometric projections of solids

12.1. Exercise on conversion of orthographic view to isometric view using CAD

1. Draw the isometric view for the given orthographic views Fig.12.1

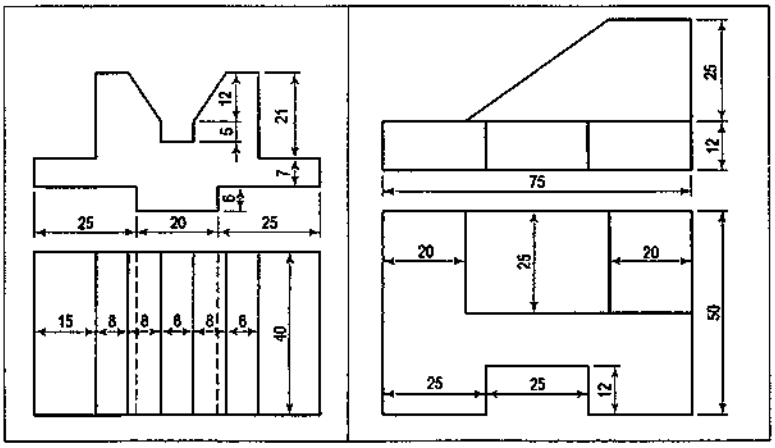


Fig.12.1

2. Draw the isometric view for the given orthographic views for Fig.12.2

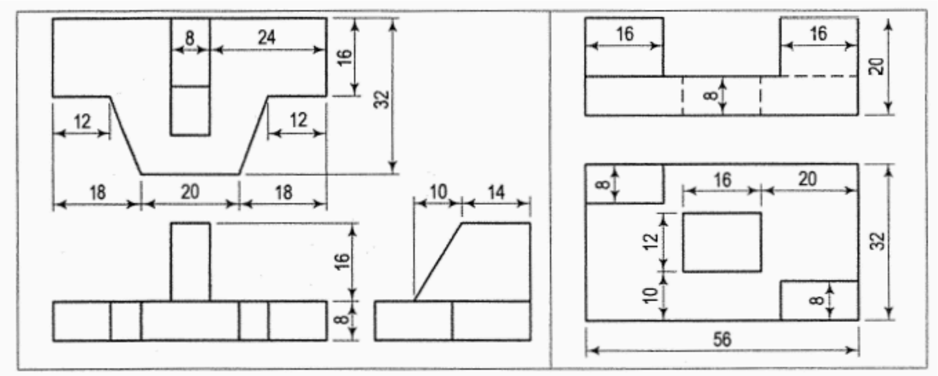


Fig.12.2

3.Draw the isometric view for the given orthographic views by assuming the dimensions for Fig.12.3

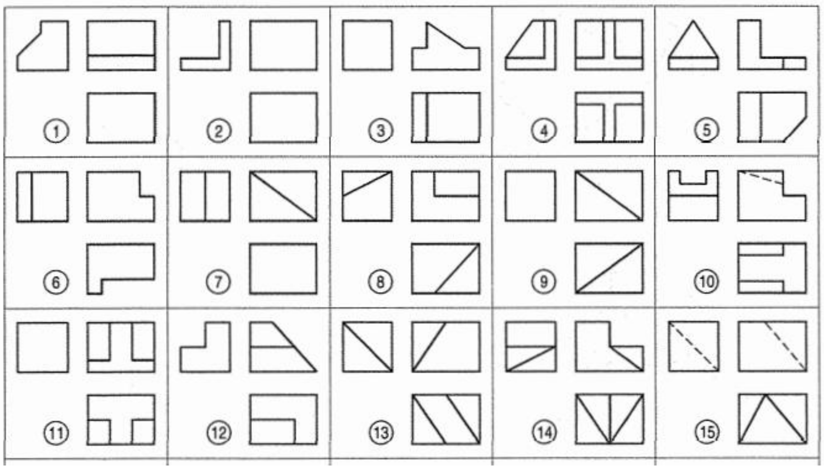


Fig.12.3

Try : The exercise 12 in Solid Works and CREO software.

13.Demonstration of SOLID WORKS Software

1. Introduction to SOLID WORKS
2. Demonstration of commands
3. 2D drawings
3. 3D drawings

V. TEXT BOOKS:

1. Frederick E Giesecke, Alva Mitchell, Henry C Spencer, Ivan L Hill, John T Dygdon, James E. Novak, R. O. Loving, Shawna Lockhart, Cindy Johnson, *Technical Drawing with Engineering Graphics*, Pearson Education, 15th Edition, 2016.
2. Kulkarni D.M, Rastogi A.P. and Sarkar A.K., *Engineering Graphics with Auto CAD*. (Revised Edition), Prentice Hall India, New Delhi, 2011.
3. Donald Hearn, "*Computer Graphics*", 12th Edition, Pearson, 2021.

VI. REFERENCE BOOKS:

1. Basant Agrawal and C M Agrawal, *Engineering Drawing*, McGraw Hill, 3rd Edition, 2018.
2. James M. Leake, Molly Hathaway Goldstein, Jacob L. Borgerson, *Engineering Design Graphics, Modelling and Visualization*, Wiley, 3rd Edition, 2020.

VII. ELECTRONICS RESOURCES:

4. https://archive.nptel.ac.in/courses/112/103/112103019.
5. https://archive.nptel.ac.in/courses/112/105/112105294.

VIII. MATERIALS ONLINE:

1. Course Template
2. Laboratory manual

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

ENGINEERING PHYSICS

I Semester: AE | ME | CE | ECE | EEE | CSE (AI & ML) | CSE (DS)
II Semester: CSE | IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
AHSE02	Foundation	L	T	P	C	CIA	SEE	Total
		3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		

Prerequisite: Basic principles of physics

I. COURSE OVERVIEW:

The aim of this course is to enhance understanding of fundamental knowledge in physics needed for the future technological advances. The framework prepares students to engage in scientific questioning and extend thinking to investigations. The concepts cover current topics in the fields of solid state physics, modern physics, superconductors and nanotechnology. This knowledge helps to develop the ability to apply the principles in many technological sectors such as nanotechnology, optical fiber communication, quantum technology etc.

II. COURSES OBJECTIVES:

The students will try to learn

I. Fundamental concepts needed to explain a crystal structure in terms of atom positions, unit cells, and crystal symmetry.
II. Basic formulations in wave mechanics for the evolution of energy levels and quantization of energies for a particle in a potential box with the help of mathematical description.
III. The metrics of optoelectronic components, lasers, optical fiber communication and be able to incorporate them into systems for optimal performance.
IV. The appropriate magnetic, superconducting and basics of quantum computing required for various engineering applications.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Use the general rules of indexing of directions and planes in lattices to identify the crystal systems and the Bravais lattices.
CO2	Use the concepts of dual nature of matter and Schrodinger wave equation to a particle enclosed in simple systems.
CO3	Analyze the concepts of laser with normal light in terms of mechanism for applications in different fields and scientific practices.
CO4	Strengthen the knowledge on functionality of components in optical fiber communication system by using the basics of signal propagation, attenuation and dispersion.
CO5	Gain deeper understanding on properties of magnetic and superconducting materials suitable for engineering applications.
CO6	Review the basic principle, types, entanglement and the logic gates of quantum computers.

IV. COURSE CONTENT:

MODULE - I: CRYSTAL STRUCTURES

Introduction, space lattice, basis, unit cell, lattice parameter, Bravais lattices, crystal systems, structure and packing fractions of simple cubic, body centered cubic, face centered cubic crystals, directions and planes in crystals, Miller indices, separation between successive [h k l] planes.

MODULE –II: QUANTUM PHYSICS

Waves and particles, de Broglie hypothesis, matter waves, Davisson and Germer's experiment, Schrödinger's time independent wave equation, physical significance of the wave function, infinite square well potential.

MODULE –III: LASERS AND FIBER OPTICS

Characteristics of lasers, spontaneous and stimulated emission of radiation, population inversion, lasing action, Ruby laser, He-Ne laser, applications of lasers.

Principle and construction of an optical fiber, acceptance angle, numerical aperture, types of optical fibers (Single mode, multimode, step index, graded index), optical fiber communication system with block diagram, applications of optical fibers.

MODULE –IV: MAGNETIC AND SUPERCONDUCTING PROPERTIES

Permeability, field intensity, magnetic field induction, magnetization, magnetic susceptibility, origin of magnetic moment, Bohr magneton, classification of dia, para and ferro magnetic materials on the basis of magnetic moment, Hysteresis curve.

Superconductivity, general properties, Meissner effect, effect of magnetic field, type-I & type-II superconductors, BCS theory, applications of superconductors.

MODULE –V: QUANTUM COMPUTING

Introduction, linear algebra for quantum computation, Dirac's Bra and Ket notation and their properties, Hilbert space, Bloch's sphere, concept of quantum computer, classical bits, Qubits, multiple Qubit system, quantum computing system for information processing, evolution of quantum systems, quantum measurements, entanglement, quantum gates, challenges and advantages of quantum computing over classical computation.

V. TEXTBOOKS:

3. Arthur Beiser, Shobhit Mahajan and Rai Choudhary, *Concepts of Modern Physics*, Tata McGraw Hill, 7th Edition, 2017.
4. Thomas G. Wong, Introduction to Classical and Quantum Computing, Rooted Grove

VI. REFERENCE BOOKS:

1. H J Callister, *A Textbook of Materials Science and Engineering,* Wiley Eastern Edition, 8th Edition, 2013.
2. Halliday, Resnick and Walker, *Fundamentals of Physics,* John Wiley &Sons,11th Edition, 2018.
3. Charles Kittel, *Introduction to Solid State Physics,* Wiley Eastern, 2019.
4. S.L. Gupta and V. Kumar, *Elementary Solid State Physics,* Pragathi Prakashan, 2019.
5. Michael A. Nielsen & Isaac L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press.

VII. ELECTRONICS RESOURCES:

1. NPTEL :: Physics - NOC:Quantum Mechanics I
2. NPTEL :: Physics - NOC:Introduction to Solid State Physics
3. NPTEL :: Physics - NOC:Solid State Physics
4. https://nptel.ac.in/courses/104104085
5. NPTEL :: Metallurgy and Material Science - NOC:Nanotechnology, Science and Applications

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open end experiments
5. Definitions and terminology
6. Assignments

7. Model question paper – I
8. Model question paper - II
9. Lecture notes
10. E-learning readiness videos (ELRV)
11. Power point presentation

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

ORDINARY DIFFERENTIAL EQUATIONS AND VECTOR CALCULUS								
II Semester: AE / ME / CE / ECE / EEE / CSE / CSE (AI&ML) / CSE (DS) / IT								
Course Code	Category	Hours/Week			Credits	Maximum Marks		
AHSE08	Foundation	L	T	P	C	CIA	SEE	Total
		3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite: Basic Principles of Matrices and Calculus								

I. COURSE OVERVIEW:

This course serves as a foundation course on differential equations and vector calculus. It includes techniques for solving ordinary differential equations, partial differential equations, vector differentiation and vector integration. It is designed to extract the mathematical developments, skills, from basic concepts to advance level of engineering problems to meet the technological challenges.

II. COURSE OBJECTIVES:

The students will try to learn:

I The analytical methods for solving first and higher order differential equations with constant coefficients.

II The analytical methods for formation and solving partial differential equations.

III The physical quantities of vector valued functions involved in engineering field.

IV The logic of vector theorems for finding line, surface and volume integrals.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1 Utilize the methods of differential equations for solving the orthogonal trajectories and Newton’s law of cooling. cooling.

CO2 Solve the higher order linear differential equations with constant coefficients by using method of variation of parameters.

CO3 Make use of analytical methods for PDE formation to solve boundary value problems.

CO4 Identify various techniques of Lagrange’s method for solving linear partial differential equations which occur in science and engineering.

CO5 Interpret the vector differential operators and their relationships for solving engineering problems.

CO6 Apply the integral transformations to surface, volume and line of different geometrical models in the domain of engineering.

IV. COURSE CONTENT:

MODULE-I: FIRST ORDER ORDINARY DIFFERENTIAL EQUATIONS (09)

Exact differential equations, equations reducible to exact differential equations, linear and Bernoulli's equations, orthogonal trajectories (only in cartesian coordinates). Applications: Newton's law of cooling, law of natural growth and decay.

MODULE-II: ORDINARY DIFFERENTIAL EQUATIONS OF HIGHER ORDER (10)

Higher order linear differential equations with constant coefficients: non-homogeneous terms of the type $e^{ax}, \sin ax , \cos ax,$ polynomials in x, $e^{ax}V(x)$ and $x\ V(x)$, method of variation of parameters.

MODULE-III: LAPLACE TRANSFORMS (10)

Laplace transforms: Laplace transform of standard functions, first shifting theorem, Laplace transforms of functions multiplied by 't' and divided by 't', Laplace transforms of derivatives and integrals of function, evaluation of integrals by Laplace transforms, Laplace transform of periodic functions.

Inverse Laplace transform by different methods, Convolution theorem (without proof). Applications: solving initial value problems by Laplace transform method

MODULE–IV: VECTOR DIFFERNTIATION (09)

Vector point functions and scalar point functions, gradient, divergence and curl, directional derivatives, vector identities, scalar potential functions, solenoidal and irrotational vectors.

.

MODULE–V: VECTOR INTEGRATION (10)

Line, surface and volume integrals. theorems of Green, Gauss and Stokes (without proofs) and their applications.

V. TEXT BOOKS:

1. B. S. Grewal, *Higher Engineering Mathematics*, 44/e, Khanna Publishers, 2017.
2. Erwin Kreyszig, *Advanced Engineering Mathematics*, 10/e, John Wiley & Sons, 2011.

VI. REFERENCE BOOKS:

1. R. K. Jain and S. R. K. Iyengar, *Advanced Engineering Mathematics*, 3/ed, Narosa Publications, 5th Edition, 2016.
2. George B. Thomas, Maurice D. Weir and Joel Hass, Thomas, *Calculus*, 13/e, Pearson Publishers, 2013.
3. N.P.Bali and Manish Goyal, *A text book of Engineering Mathematics*, Laxmi Publications, Reprint, 2008
4. Dean G. Duffy, *Advanced Engineering Mathematics with MATLAB*, CRC Press.
5. Peter O'Neil *Advanced Engineering Mathematics*, Cengage Learning.
6. B.V. Ramana, *Higher Engineering Mathematics*, McGraw Hill Education.

VII. ELECTRONIC RESOURCES:

1. Engineering Mathematics - I, By Prof. Jitendra Kumar |https://onlinecourses.nptel.ac.in/noc23_ma88/preview
2. Advanced Calculus for Engineers, By Prof. Jitendra Kumar, Prof. Somesh Kumar https://onlinecourses.nptel.ac.in/noc23_ma86/preview
3. http://www.efunda.com/math/math_home/math.cfm
4. http://www.ocw.mit.edu/resourcs/#Mathematics
5. http://www.sosmath.com
6. http://www.mathworld.wolfram.com

VIII. MATERIAL ONLINE:

12. Course template
13. Tutorial question bank
14. Tech talk topics

15. Open end experiments
16. Definitions and terminology
17. Assignments
18. Model question paper – I
19. Model question paper - II
20. Lecture notes
21. E-learning readiness videos (ELRV)
22. Power point presentation

I A R E

EDUCATION FOR LIBERATION

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DATA STRUCTURES

II Semester: AE / ME / CE / ECE / EEE / CSE / IT / CSE (AI&ML) / CSE (DS)

Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACSE05	Core	L	T	P	C	CIA	SEE	Total
		3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite: Essentials of Problem Solving								

I. COURSE OVERVIEW:

The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSES OBJECTIVES:

The students will try to learn

I. The skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.

II. The basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.

III. The fundamentals of how to store, retrieve, and process data efficiently.

IV. The implementing these data structures and algorithms in Python.

V. The essential for future programming and software engineering courses.

III.COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Interpret the complexity of the algorithm using the asymptotic notations.

CO 2 Select the appropriate searching and sorting technique for a given problem

CO 3 Construct programs on performing operations on linear and nonlinear data structuresfor organization of a data

CO 4 Make use of linear data structures and nonlinear data structures solving real-time applications.

CO 5 Describe hashing techniques and collision resolution methods for accessing data with respect to performance

CO 6 Compare various types of data structures; in terms of implementation, operations and performance.

IV. COURSE CONTENT:

MODULE – I: INTRODUCTION TO DATA STRUCTURES, SEARCHING AND SORTING (09)

Basic concepts: Introduction to data structures, classification of data structures, operations on data structures, Algorithm Specification, Recursive algorithms, Data Abstraction, Performance analysis- time complexity and space complexity, Introduction to Linear and Non Linear data structures,Searching techniques: Linear and Binary search, Uniform Binary Search, Interpolation Search, Fibonacci Search; Sorting techniques: Bubble, Selection, Insertion, and Quick, Merge, Radix and Shell Sort and comparison of sorting algorithms.

MODULE – II: LINEAR DATA STRUCTURES (09)

Stacks: Stack ADT, definition and operations, Implementations of stacks using array, applications of stacks, Arithmetic expression conversion and evaluation; Queues: Primitive operations; Implementation of queues using Arrays, applications of linear queue, circular queue and double ended queue (deque).

MODULE – III: LINKED LISTS (09)

Linked lists: Introduction, singly linked list, representation of a linked list in memory, operations on a single linked list; Applications of linked lists: Polynomial representation and sparse matrix manipulation.

Types of linked lists: Circular linked lists, doubly linked lists; Linked list representation and operations of Stack, linked list representation and operations of queue.

MODULE – IV: NON LINEAR DATA STRUCTURES (09)

Trees: Basic concept, binary tree, binary tree representation, array and linked representations, binary tree traversal, binary tree variants, threaded binary trees, application of trees, Graphs: Basic concept, graph terminology, Graph Representations - Adjacency matrix, Adjacency lists, graph implementation, Graph traversals – BFS, DFS, Application of graphs, Minimum spanning trees – Prims and Kruskal algorithms.

MODULE – V: BINARY TREES AND HASHING (09)

Binary search trees: Binary search trees, properties and operations; Balanced search trees: AVL trees; Introduction to M- Way search trees, B trees; Hashing and collision: Introduction, hash tables, hash functions, collisions, applications of hashing.

V. TEXT BOOKS:

1. Rance D. Necaise, “Data Structures and Algorithms using Python”, Wiley Student Edition.
2. Benjamin Baka, David Julian, “Python Data Structures and Algorithms”, Packt Publishers, 2017.

VI. REFERENCE BOOKS:

1. S. Lipschutz, “Data Structures”, Tata McGraw Hill Education, 1st edition, 2008.
2. D. Samanta, “Classic Data Structures”, PHI Learning, 2nd edition, 2004.

VII. ELECTRONIC RESOURCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

VIII. MATERIALS ONLINE

1 Course template
2 Tutorial question bank
3 Tech talk topics
4 Assignments
5 Definitions and terminology
6 Open ended experiments
7 Model question paper-I
8 Model question paper-II
9 Lecture notes
10 Power point presentations
11 ELRV videos

COURSE CONTENT

ENGLISH FOR SKILL ENHANCEMENT								
I Semester: AE / ME / CE / ECE / EEE / CSE (AI &ML) / CSE (DS) II Semester: CSE / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AHSE04	Foundation	L	T	P	C	CIA	SEE	Total
		3	-	-	3	40	60	100
Contact Classes: 48	Tutorial Classes: Nil	Practical Classes: Nil				Total Classes: 48		
Prerequisite:								

I. COURSE OVERVIEW:

The principle aim of the course is that the students will have awareness about the importance of English language in the contemporary times and also it emphasizes the students to learn this languageas a skill (listening skill, speaking skill, reading skill and writing skill). Moreover, the course benefitsthe students how to solve their day-to-day problems in speaking English language. Besides, it assiststhe students to reduce the mother tongue influence and acquire the knowledge of neutral accent. Thecourse provides theoretical and practical knowledge of English language and it enables students to participate in debates about informative, persuasive, didactic, and commercial purposes.

II. COURSE OBJECTIVES:

The students will try to learn:

I Standard pronunciation, appropriate word stress, and necessary intonation patterns foreffective communication towards achieving academic and professional targets.

II Appropriate grammatical structures and also using the nuances of punctuation tools forpractical purposes.

III Critical aspect of speaking and reading for interpreting in-depth meaning between thesentences.

IV Conceptual awareness on writing in terms of unity, content, coherence, and linguisticaccuracy.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO 1	Demonstrate the essential listening and communication skills required for academic and non-academic purposes.
CO 2	Explain ideas and discuss issues effectively in spoken English with a high level of fluency and accuracy across different social contexts.
CO 3	Enhance language proficiency to strengthen life skills and effectively navigate challenges in a professional environment.
CO 4	Interpret grammatical and lexical forms of English and apply them in specific communicative contexts.
CO 5	Develop the ability to comprehend, analyze, and interpret a variety of texts, enhancing critical thinking, vocabulary, and the application of reading strategies for academic, professional, and personal growth.
CO 6	Improve the ability to produce clear, coherent, and well-structured written content and organization for academic, professional, and creative tenacities.

IV. COURSE CONTENT:

MODULE – I: PERSPECTIVES (13)

Lesson on 'The Generation Gap' by Benjamin M. Spock from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Vocabulary: The Concept of Word Formation -The Use of Prefixes and Suffixes - Words Often Misspelt - Synonyms and Antonyms

Grammar: Identifying Common Errors in Writing with Reference to Parts of Speech particularly Articles and Prepositions – Degrees of Comparison

Reading: Reading and Its Importance- Sub Skills of Reading – Skimming and Scanning.

Writing: Sentence Structures and Types -Use of Phrases and Clauses in Sentences- Importance of Proper Punctuation- Techniques for Writing Precisely –Nature and Style of Formal Writing.

MODULE – II: DIGITAL TRANSFORMATION (13)

Lesson on '*Emerging Technologies*' from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Vocabulary: Homophones, Homonyms and Homographs,

Grammar: Identifying Common Errors in Writing with Reference to Noun-pronoun Agreement and Subject-verb Agreement.

Reading: Reading Strategies-Guessing Meaning from Context – Identifying Main Ideas – Exercises for Practice,

Writing: Paragraph Writing – Types, Structures and Features of a Paragraph - Creating Coherence – Linkers and Connectives - Organizing Principles in a Paragraph – Defining- Describing People, Objects, Places and Events – Classifying- Providing Examples or Evidence - Essay Writing - Writing Introduction and Conclusion.

MODULE – III: ATTITUDE AND GRATITUDE (13)

Poems on '*Leisure*' by William Henry Davies and '*Be Thankful*' - Unknown Author from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Vocabulary: Words Often Confused - Words from Foreign Languages and their Use in English.

Grammar: Identifying Common Errors in Writing with Reference to Misplaced Modifiers and Tenses.

Reading: Sub-Skills of Reading – Identifying Topic Sentence and Providing Supporting Ideas - Exercises for Practice.

Writing: Format of a Formal Letter-Writing Formal Letters E.g.., Letter of Complaint, Letter of Requisition, Job Application with CV/Resume –Difference between Writing a Letter and an Email - Email Etiquette.

MODULE – IV: ENTREPRENEURSHIP (12)

Lesson on '*Why a Start-Up Needs to Find its Customers First*' by Pranav Jain from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Vocabulary: Standard Abbreviations in English – Inferring Meanings of Words through Context – Phrasal Verbs – Idioms.

Grammar: Redundancies and Clichés in Written Communication – Converting Passive to Active Voice and Vice-Versa.

Reading: Prompt Engineering Techniques– Comprehending and Generating

Appropriate Prompts - Exercises for Practice

Writing: **Writing Practices- Note Making-Précis Writing.**

MODULE – V: INTEGRITY AND PROFESSIONALISM (13)

Lesson on '*Professional Ethics*' from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Lesson on '*Professional Ethics*' from the prescribed textbook titled *English for the Young in the Digital World* published by Orient Black Swan Pvt. Ltd.

Vocabulary: Technical Vocabulary and their Usage– One Word Substitutes – Collocations.

Grammar: Direct and Indirect Speech **-** Common Errors in English (Covering all the other aspects of grammar which were not covered in the previous units)

Reading: Survey, Question, Read, Recite and Review (SQ3R Method) – Inferring the Meaning and Evaluating a Text- Exercises for Practice

Writing: ***Report Writing - Technical Reports- Introduction – Characteristics of a Report – Categories of Reports Formats- Structure of Reports (Manuscript Format) -Types of Reports - Writing a Technical Report.***

V. TEXT BOOKS:

1. Board of Editors. 2025. English for the Young in the Digital World. Orient Black Swan Pvt. Ltd.

VI. REFERENCE BOOKS:

1 Swan, Michael. (2016). Practical English Usage. Oxford University Press. New Edition.

2 Karal, Rajeevan. 2023. English Grammar Just for You. Oxford University Press. New Delhi

3 2024. Empowering with Language: Communicative English for Undergraduates. Cengage Learning India Pvt. Ltd. New Delhi

4 Sanjay Kumar & Pushp Lata. 2022. Communication Skills – A Workbook. Oxford University Press. New Delhi

5 Wood,F.T. (2007). Remedial English Grammar. Macmillan.

6 Vishwamohan, Aysha. (2013). English for Technical Communication for Engineering Students. Mc Graw-Hill Education India Pvt. Ltd

VII. ELECTRONICS RESOURCES:

1. https://akanksha.iare.ac.in/index?route=course/details&course_id=954
2. https://akanksha.iare.ac.in/index?route=course/details&course_id=10
3. https://akanksha.iare.ac.in/index?route=course/details&course_id=352
4. https://akanksha.iareac.in/index?route=publicprofile&id=5075

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Open end experiments
5. Definitions and terminology
6. Assignments
7. Model question paper – I
8. Model question paper - II
9. Lecture notes
10. E-learning readiness videos (ELRV)
11. Power point presentation

COURSE CONTENT

AI FOUNDATIONS								
II Semester: CSE / IT / CSE (AI&ML) / CSE (DS)								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSE06	**Foundation**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		3	0	0	3	40	60	100
Contact Classes: 48	**Tutorial Classes: NIL**	**Practical Classes: NIL**				**Total Classes: 48**		
Prerequisite: Python Programming								

I. COURSE OVERVIEW:

Artificial intelligence (AI) is the simulation that examines to achieve intelligent human behaviors on machines especially on a computer system. This course provides the ideas, methods, and problem-solving paradigms that helps in providing solutions to real-world problems without human effort. Furthermore, it is a mathematical language that enables knowledge to be expressed precisely and unambiguously, making it perfect for usage in AI systems. AI applications are becoming increasingly common in a wide variety of applications including machine language, deep learning, natural language processing, computer vision, and robotics.

II. COURSES OBJECTIVES:

The students will try to learn:

I The fundamental concepts of Artificial Intelligence, including the characteristics of intelligent agents and how AI systems solve problems through state space search and production systems.

II The various problem-solving strategies such as heuristic search, optimization, and logic-based reasoning using predicate logic, along with effective knowledge representation techniques.

III the structure and functionality of software agents, their communication mechanisms, and explore real-world AI applications in natural language processing, robotics, machine learning, and expert systems.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1 Explain the ability to design a plan for the real-world problems and mapping it to the digital world.

CO2 Choose appropriate problem-solving methods and optimize the search results.

CO3 Develop agents through knowledge representation for any given AI based problem using logic programming.

CO4 Discover how planning helps to automate complicated tasks, manage complex procedures, and optimize them for better results.

CO5 Apply principles of negotiation, bargaining, argumentation, and trust management in multi-agent systems to develop cooperative and competitive software solutions

CO6 Apply artificial intelligence techniques to real-world domains by utilizing language models, information retrieval and extraction.

IV. COURSE SYLLABUS:

MODULE – 1: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

Fundamentals of Artificial Intelligence: Definitions, Introduction, key concepts, Evolution, Terminology, Approaches and Goals. Ethical aspects of Artificial Intelligence. Relation between Artificial Intelligence, Machine Learning and Deep Learning. Intelligent Agents: Structure, Types, and interaction with the environment.

MODULE – 2: KNOWLEDGE REASONING (10)

Introduction to Knowledge representation, Building a Knowledge Base: propositional logic,first order predicate logic and inferencing, resolution, representing Knowledge using rules: Procedural versus Declarative Knowledge, Forward versus Backward reasoning. Uncertain Knowledge and Reasoning, Statistical Reasoning: Probability and Bayes theorem, Bayesian Networks and Dempster-Shafer Theory.

MODULE – 3: PROBLEM SOLVING (09)

State space search; production systems, search space control, Uninformed and Informed Search: depth first search, breadth-first search. Heuristic Search: Best First Search, Hill Climbing, AND/OR Heuristic Search. Game Playing: Minimax, alpha-beta pruning.

MODULE – 4: LEARNING IN ARTIFICIAL INTELLIGENCE (10)

Definition, process, types - unsupervised and supervised learning. Regression, Classification, Bias-Variance trade-off, Overfitting-Underfitting, loss function, cross-validation

MODULE – 5: CASE STUDIES, ISSUES, CHALLENGES AND APPLICATIONS (10)

Healthcare: Diagnosis, treatment, and medical imaging; Finance: Fraud detection, algorithmic trading, and risk assessment; Transportation: Autonomous vehicles and traffic optimization; Customer service and chatbots; Education: Personalized learning and intelligent tutoring systems.

V. TEXT BOOKS:

1. S. Russel, P. Norvig, “Artificial Intelligence – A Modern Approach,” Third Edition, Pearson Education, 2015.
2. Patrick Henry Winston ,“Artificial Intelligence”, Third Edition, Addison-Wesley Publishing Company, 2004
3. Nils J. Nilsson,” Principles of Artificial Intelligence”, Illustrated Reprint Edition, Springer Heidelberg, 2014

VI. REFERENCE BOOKS:

1. Kevin Night, Elaine Rich, Nair B., “Artificial Intelligence (SIE)”, Third Edition, McGraw Hill, 2017.
2. Dan W. Patterson, “Introduction to AI and ES”, Pearson Education, 2007.
3. Khemani, Deepak,”A first course in artificial intelligence”, McGraw-Hill Education, 2014

VII. ELECTRONICS RESOURCES:

1. Department of Computer Science, University of California, Berkeley, http://www.youtube.com/playlist?list=PLD52D2B739E4D1C5F
2. NPTEL: Artificial Intelligence, https://nptel.ac.in/courses/106105077/
3. http://www.udacity.com/ 4. http://www.library.thinkquest.org/2705/
4. http://www.ai.eecs.umich.edu/

VIII. MATERIALS ONLINE

1. Course template
2. Tutorial question bank
3. Tech talk topics
4. Assignments
5. Definitions and terminology
6. Open ended experiments
7. Model question paper-I
8. Model question paper-II
9. Lecture notes
10. Power point presentations
11. ELRV videos

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

ENGINEERING PHYSICS LABORATORY								
I Semester: AE l ME l CE l ECE l EEE l CSE (AI&ML) l CSE (DS) II Semester: CSE l IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AHSE05	Foundation	L	T	P	C	CIA	SEE	Total
		-	-	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 45				Total Classes: 45		
Prerequisite: Basic Principles of Physics								

I. COURSE OVERVIEW:

The aim of the course is to provide hands on experience for experiments in different areas of physics. Students will be able to perform the experiments with interest and an attitude of learning. This laboratory includes experiments involving electromagnetism and optoelectronics. These also develop student's expertise in applying physical concepts to practical problem and apply it for different applications.

II. COURSES OBJECTIVES:

The students will try to learn:

I Familiarize with the lab facilities, equipment, standard operating procedures.

II The different kinds of functional magnetic materials which paves a way for them to use in various technical and engineering applications.

III The analytical techniques and graphical analysis to study the experimental data for optoelectronicdevices.

IV The application of characteristics of lasers and its propagation in optical fiber communication.

III. COURSE OUTCOMES:

After successful completion of the course, students should be able to:

CO1 Identify the type of semiconductor using the principle of Hall effect and also determine the energy gap and resistivity of a semiconductor diode using four probe method.

CO2 Illustrate principle, working and application of wave propagation and compare the results of frequency with theoretical harmonics and overtones.

CO3 Investigate the energy losses, curie temperature and properties associated with a given Ferro magnetic material.

CO4 Examine launching of light through optical fiber from the concept of light gathering capacity of numerical aperture and determine the divergence of Laser beam

CO5 Graph V-I /L-I characteristics of various optoelectronic devices like Light Emitting diode, Solar cell at different intensities to understand their basic principle of functioning as well as to infer the value of Planck's constant.

CO6 Analyze the variation of magnetic field induction produced at various points along the axis of current carrying coil.

IV.COURSE SYLLABUS:

1. GETTING STARTED EXCERCISES

1.1 On Errors and Uncertainty in a measurement:

When a number represents a physical measurement, it is never exact because of the limitations of the instrument used or the way it was employed etc. It is essential, therefore, that each experimental result be presented in a way that indicates its reliability. The accuracy of result is important, for example, the calibration of the measuring instruments or systematic errors on the part of whoever is taking the data.

The following table is useful in thinking about these concepts:

Problem	Remedy
Mistakes and blunders	Repeat measurements several times to check yourself
Systematic errors	Use calibrated instruments properly and carefully
Random errors	Treat data statistically and report on the average magnitude of errors

1.2 Making a good graph:

- Keep your axes straight: If you need to plot "A vs B", or "A as a function of B", then A is on the vertical axis and B is on the horizontal axis.
- The crucial part is choosing the range and scale for each axis. The range must be just large enough to accommodate all data and small enough that the scale is readable.
- The scale should be spread out enough so that data take up most of the graph area and labeled so that plotting (and reading) is easy. Where appropriate, error bars should be included to indicate the uncertainty in measurements.
- When a line is drawn, it should be a smooth one that best fits data. In general, there should be as many points on one side of the line as on the other. If data is taken properly, the line should pass inside of the error bars for each point.
- If graph shows that one quantity is proportional to another, it should be a straight line that starts at the origin and passes through the plotted data with as many points on one side as the other.
- If the slope of the line is to be found, choose two points on the line that are as far apart as possible. This will minimize the error that is introduced in reading the value of those points.
- The slope is the difference between the vertical values of those points divided by the difference in the horizontal values of those points.

1.3 Data recording and worksheets

- Write the work sheets for the allotted experiment and keep them ready before the beginning of each lab.
- Perform the experiment and record the observations in the worksheets.
- Analyze the results and get the work sheets evaluated by the faculty.
- Upload the evaluated reports online from CMS LOGIN within the stipulated time.

2. HALL EFFECT (LORENTZ FORCE)

2.1 Study the phenomenon of Hall effect and determine the charge carrier density and Hall coefficient of a given sample.

2.2 Hint whether the given semiconductor is p - type or n - type using the principle of hall effect.

3. ENERGY GAP OF A SEMICONDUCTOR DIODE

3.1 Determination of energy gap of a given semiconductor diode by measuring the variation of current as a function of temperature.

3.2 Try to find the Fermi level of the given semiconductor

4. RESISTIVITY – FOUR PROBE METHOD

4.1 Determination of the resistivity by forcing current through two outer probes

4.2 Formulate the reading of voltage across the two inner probes of semiconductor by four probe method.

5. MAGNETIC MOMENT

5.1 Determination of magnetic moment of a bar magnet.

5.2 Try to find horizontal component of earth's magnetic field.

6. B-H CURVE WITH CRO

6.1 Evaluate the energy loss per unit volume of a given magnetic material per cycle by tracing the hysteresis loop (B-H curve).

6.2 Observe the hysteresis loss of ferro magnetic materials.

7. FERROMAGNETIC MATERIAL

7.1 Determine the curie temperature (Tc) of a ferromagnetic material.

7.2 Evaluate the relative permeability (μ_r) of a ferromagnetic material.

8. OPTICAL FIBER

8.1 Determine the numerical aperture of a given optical fiber.

8.2 Calculate the acceptance angle of a given optical fiber.

9. LASER DIVERGENCE

9.1 Determination of the beam divergence of the given laser beam.

9.2 Try to estimate the laser output

10. SOLAR CELL

10.1 Studying the characteristics of solar cell at different intensities

10.2 Try to get the maximum workable power.

11. LIGHT EMITTING DIODE

11.1 Studying V-I characteristics of LED in forward bias for different LEDs.

11.2 Measure the threshold voltage and forward resistance, and try for the dynamic Resistance

12. PLANCK’S CONSTANT

12.1 Determination of Planck’s constant by measuring threshold voltage of given LED.

12.2 Draw the L -I characteristics of the given LED.

13. STEWART GEE’S APPARATUS

13.1 Study the magnetic field along the axis of current carrying coil – Stewart and Gee’s method.

13.2 Estimate the magnetic lines of force.

14. MELDE’S EXPERIMENT

14.1 Determination of frequency of a given tuning fork in longitudinal wave propagation.

14.2 Try to establish the transverse mode of wave propagation by understanding the theoretical harmonics and overtones

V. TEXTBOOKS:

1. Laboratory Experiments in College Physics”, C.H. Bernard and C.D. Epp, John Wiely and Sons, Inc., New York, 1995.

VI. REFERENCE BOOKS:

1. C. L. Arora, “Practical Physics”, S. Chand & Co., New Delhi, 3rd Edition, 2012.
2. Vijay Kumar, Dr. T Radhakrishna, “Practical Physics for Engineering Students”, SM Enterprises, 2nd Edition, 2014.
3. Dr. Rizwana, “Engineering Physics Manual”, Spectrum Techno Press, 2018.

VII. ELECTRONICS RESOURCES:

1. https://nptel.ac.in/translation
2. https://nptel.ac.in/courses/115105120
3. NPTEL:: courses-Sem 1 and 2 - Engineering Physics and Applied Physics I
4. Experimental Physics I - Course (nptel.ac.in)
5. NPTEL:: Physics - Waves and Oscillations

VIII. MATERIALS ONLINE:

1. Course template
2. Lab manual

IARE

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DATA STRUCTURES LABORATORY								
II Semester: AE / ME / CE / ECE / EEE / CSE / IT / CSE (AI&ML) / CSE (DS)								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
ACSE08	Foundation	L	T	P	C	CIA	SEE	Total
		-	-	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 45				Total Classes: 45		
Prerequisite: Essentials of Problem Solving								

I. COURSE OVERVIEW:

The course covers some of the general-purpose data structures and algorithms, and software development. Topics covered include managing complexity, analysis, static data structures, dynamic data structures and hashing mechanisms. The main objective of the course is to teach the students how to select and design data structures and algorithms that are appropriate for problems that they might encounter in real life. This course reaches to student by power point presentations, lecture notes, and lab which involve the problem solving in mathematical and engineering areas.

II. COURSES OBJECTIVES:

The students will try to learn

I. To provide students with skills needed to understand and analyze performance trade-offs of different algorithms / implementations and asymptotic analysis of their running time and memory usage.

II. To provide knowledge of basic abstract data types (ADT) and associated algorithms: stacks, queues, lists, tree, graphs, hashing and sorting, selection and searching.

III. The fundamentals of how to store, retrieve, and process data efficiently.

IV. To provide practice by specifying and implementing these data structures and algorithms in Python.

V. Understand essential for future programming and software engineering courses.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Interpret the complexity of algorithm using the asymptotic notations.

CO 2 Select appropriate searching and sorting technique for a given problem.

CO 3 Construct programs on performing operations on linear and nonlinear data structures for organization of a data.

CO 4 Make use of linear data structures and nonlinear data structures solving real time applications.

CO 5 Describe hashing techniques and collision resolution methods for efficiently accessing data with respect to performance.

CO 6 Compare various types of data structures; in terms of implementation, operations and performance.

DATA STRUCTURES LABORATORY COURSE CONTENT

IV. COURSE CONTENT:

EXERCISES FOR DATA STRUCTURES LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Sum of last digits of two given numbers

Rohit wants to add the last digits of two given numbers. For example, If the given numbers are 267 and 154, the output should be 11.
Below is the explanation -
Last digit of the 267 is 7
Last digit of the 154 is 4
Sum of 7 and 4 = 11
Write a program to help Rohit achieve this for any given two numbers.
The prototype of the method should be -
int addLastDigits(int input1, int input2);
where input1 and input2 denote the two numbers whose last digits are to be added.
Note: The sign of the input numbers should be ignored.
if the input numbers are 267 and 154, the sum of last two digits should be 11
if the input numbers are 267 and -154, the sum of last two digits should be 11
if the input numbers are -267 and 154, the sum of last two digits should be 11
if the input numbers are -267 and -154, the sum of last two digits should be 11
Input: 267 154
Output: 11
Input: 267 -154
Output: 11
Input: -267 154
Output: 11
Input: -267 -154
Output: 11

```
import java.util.Scanner;

class AddLastDigitsFunction
{
       int addLastDigits(int n1, int n2)
       {
               # Write code here
       }

       public static void main(String args[])
       {
             AddLastDigitsFunction obj = new AddLastDigitsFunction();
             # Write code here
             System.out.println(obj.addLastDigits(n1,n2));
       }

}
```

1.2 Is N an exact multiple of M?

Write a function that accepts two parameters and finds whether the first parameter is an exact multiple of the second parameter. If the first parameter is an exact multiple of the second parameter, the function should return 2 else it should return 1.

If either of the parameters are zero, the function should return 3.

Assumption: Within the scope of this question, assume that - the first parameter can be positive, negative or zero the second parameter will always be >=0

Input: num1 = 10, num2 = 5
Output: 2
Input: num1 = -10, num2 = 5
Output: 2
Input: num1 = 0, num2 = 5
Output: 3
Input: num1 = 10, num2 = 3
Output: 1

```
public class MultipleChecker
{
    public static int checkMultiple(int num1, int num2)
    {
        # Write code here
    }

    public static void main(String[] args)
    {
        # Write code here
    }
}
```

1.3 Combine Strings

Given 2 strings, a and b, return a new string of the form short+long+short, with the shorter string on the outside and the longer string in the inside. The strings will not be the same length, but they may be empty (length 0).

If input is "hi" and "hello", then output will be "hihellohi"

Input: Enter the first string: “hi”
Enter the second string: “hello”
Output: “hihellohi”
Input: Enter the first string: “iare”
Enter the second string: “college”
Output: “iarecollegeiare”

```
public class StringCombiner
{
    public static void main(String[] args)
    {
        # Write code here
    }

    public static String combineStrings(String a, String b)
    {
        # Write code here
    }
}
```

1.4 Even or Odd

Write a function that accepts 6 input parameters. The first 5 input parameters are of type int. The sixth input parameter is of type string. If the sixth parameter contains the value "even", the function is supposed to return the count of how many of the first five input parameters are even. If the sixth parameter contains the value "odd", the function is supposed to return the count of how many of the first five input parameters are odd.

Example:

If the five input parameters are 12, 17, 19, 14, and 115, and the sixth parameter is "odd", the function must return 3, because there are three odd numbers 17, 19 and 115.

If the five input parameters are 12, 17, 19, 14, and 115, and the sixth parameter is "even", the function must return 2, because there are two even numbers 12 and 14.

Note that zero is considered an even number.

Input: num1 = 12;
num2 = 17;
num3 = 19;
num4 = 14;
num5 = 115;
type = "odd"

Output: 3

Input: num1 = 12;
num2 = 17;
num3 = 19;
num4 = 14;
num5 = 115;
type = "even"

Output: 2

```
public class NumberCounter
{
    public static int countNumbers(int num1, int num2, int num3, int num4, int
num5, String type)
   {
        # Write code here
   }

    public static void main(String[] args)
    {
        # Write code here
    }
}
```

1.5 Second last digit of a given number

Write a function that returns the second last digit of the given number. Second last digit is being referred to the digit in the tens place in the given number.

Example: if the given number is 197, the second last digit is 9.

Note 1: The second last digit should be returned as a positive number. i.e. if the given number is -197, the second last digit is 9.

Note 2: If the given number is a single digit number, then the second last digit does not exist. In such cases, the function should return -1. i.e. if the given number is 5, the second last digit should be returned as -1.

Input: 197
Output: 9
Input: 5
Output: -1
Input: -197
Output: 9

```
public class SecondLastDigit
{
    public static int getSecondLastDigit(int number)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.6 Alternate String Combiner

Given two strings, a and b, print a new string which is made of the following combination-first character of a, the first character of b, second character of a, second character of b and so on.
Any characters left, will go to the end of the result.
Hello,World
HWeolrllod
Input: "Hello,World"
Output: "HWeolrllod"
Input: "Iare,College"
Output: "ICaorlelege"

```
public class AlternateStringCombiner
{
    public static void main(String[] args)
    {
        # write code here
    }
    public static String combineStrings(String a, String b)
    {
        # write code here
    }
}
```

1.7 Padovan Sequence

The Padovan sequence is a sequence of numbers named after Richard Padovan, who attributed its discovery to Dutch architect Hans van der Laan. The sequence was described by Ian Stewart in his Scientific American column Mathematical Recreations in June 1996. The Padovan sequence is defined by the following recurrence relation:
P(n) = P(n-2) + P(n-3)
with the initial conditions P(0) = P(1) = P(2) = 1.
In this sequence, each term is the sum of the two preceding terms, similar to the Fibonacci sequence. However, the Padovan sequence has different initial conditions and exhibits different growth patterns.

The first few terms of the Padovan sequence are: 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, ...

Input: num = 10
Output: Padovan Sequence up to 10:

```
1 1 1 2 2 3 4 5 7 9 12
```

Input: num = 20
Output: Padovan Sequence up to 20:

```
1 1 1 2 2 3 4 5 7 9 12 16 21 28 37 49 65 86 114 151 200
```

```
public class PadovanSequence
{
    public static int padovan(int n)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.8 Leaders in an array

Given an array arr of n positive integers, your task is to find all the leaders in the array. An element of the array is considered a leader if it is greater than all the elements on its right side or if it is equal to the maximum element on its right side. The rightmost element is always a leader.

Input: n = 6, arr[] = (16, 17, 4, 3, 5, 2}

Output: 17 5 2

Input: n = 5, arr[] = {10, 4, 2, 4, 1}

Output: 10 4 4 1

Input: n = 4, arr[] = {5, 10, 20, 40}

Output: 40

Input: n = 4, arr[] = {30, 10, 10, 5}

Output: 30 10 10 5

```
import java.util.ArrayList;
import java.util.List;
public class ArrayLeaders
{
    public static List<Integer> findArrayLeaders(int[] arr)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.9 Find the Value of a Number Raised to its Reverse

Given a number N and its reverse R. The task is to find the number obtained when the number is raised to the power of its own reverse

Input : N = 2, R = 2

Output: 4

Explanation: Number 2 raised to the power of its reverse 2 gives 4 which gives 4 as a result after performing modulo 10^9+7

Input: N = 57, R = 75
Output: 262042770
Explanation: 57^{75} modulo 10^9+7 gives us the result as 262042770

```
public class NumberPower
{
    public static long powerOfReverse(int N, int R)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

1.10 Mean of Array using Recursion

Find the mean of the elements of the array.
Mean = (Sum of elements of the Array) / (Total no of elements in Array)
Input: 1 2 3 4 5
Output: 3.0
Input: 1 2 3
Output: 2.0
To find the mean using recursion assume that the problem is already solved for N-1 i.e. you have to find for n
Sum of first N-1 elements = (Mean of N-1 elements) * (N-1)
Mean of N elements = (Sum of first N-1 elements + N-th elements) / (N)

```
public class ArrayMean
{
    public static double findArrayMean(int[] arr)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

Try:

1. **Kth Smallest Element:** Given an array arr[] and an integer k where k is smaller than the size of the array, the task is to find the kth smallest element in the given array. It is given that all array elements are distinct.
Note: l and r denotes the starting and ending index of the array.
Input: n = 6, arr[] = {7, 10, 4, 3, 20, 15}, k = 3, l = 0, r = 5
Output: 7
Explanation: 3rd smallest element in the given array is 7.
Input: n = 5, arr[] = {7, 10, 4, 20, 15}, k = 4, l=0 r=4
Output: 15

Explanation: 4th smallest element in the given array is 15.
Your task is to complete the function **kthSmallest()** which takes the array arr[], integers l

and r denoting the starting and ending index of the array and an integer k as input and returns the kth smallest element.

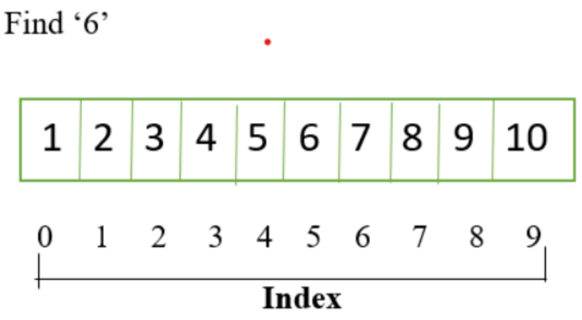
2. **Count pairs with given sum:** Given an array of N integers, and an integer K, find the number of pairs of elements in the array whose sum is equal to K. Your task is to complete the function **getPairsCount()** which takes arr[], n and k as input parameters and returns the number of pairs that have sum K.

Input: N = 4, K = 6, arr[] = {1, 5, 7, 1}
Output: 2
Explanation: arr[0] + arr[1] = 1 + 5 = 6 and arr[1] + arr[3] = 5 + 1 = 6.
Input: N = 4, K = 2, arr[] = {1, 1, 1, 1}
Output: 6
Explanation: Each 1 will produce sum 2 with any 1.

2. Searching

2.1 Linear / Sequential Search

Linear search is defined as the searching algorithm where the list or data set is traversed from one end to find the desired value. Given an array arr[] of n elements, write a recursive function to search a given element x in arr[].



Note : We find ‘6’ at index ‘5’ through linear search

Linear search procedure:

1. Start from the leftmost element of arr[] and one by one compare x with each element of arr[]
2. If x matches with an element, return the index.
3. If x doesn’t match with any of the elements, return -1.

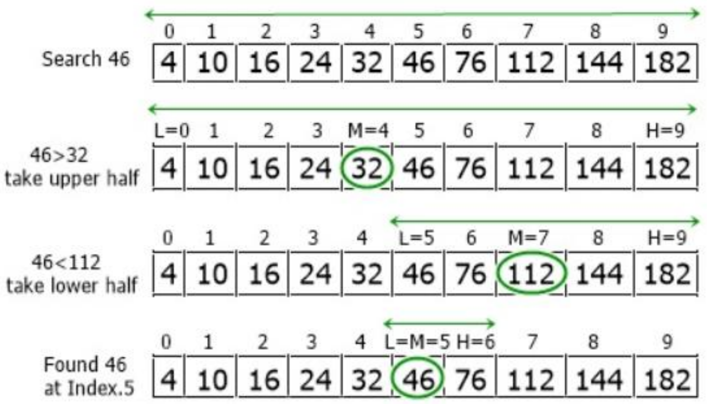
Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 110;
Output: 6
Element x is present at index 6

Input: arr[] = {10, 20, 80, 30, 60, 50, 110, 100, 130, 170}
x = 175;
Output: -1
Element x is not present in arr[].

```
public class RecursiveLinearSearch
{
    public static int recursiveLinearSearch(int[] arr, int key, int index)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

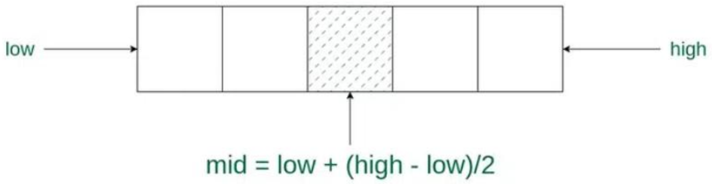
2.2 Binary Search

Binary Search is defined as a searching algorithm used in a sorted array by repeatedly dividing the search interval in half. The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(log N).



Conditions for Binary Search algorithm:

1. The data structure must be sorted.
2. Access to any element of the data structure takes constant time.



Binary Search Procedure:

1. Divide the search space into two halves by finding the middle index “mid”.
2. Compare the middle element of the search space with the key.
3. If the key is found at middle element, the process is terminated.
4. If the key is not found at middle element, choose which half will be used as the next search space.
    a. If the key is smaller than the middle element, then the left side is used for next search.
    b. If the key is larger than the middle element, then the right side is used for next search.
5. This process is continued until the key is found or the total search space is exhausted.

Input: arr = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
Output: target = 23
Element 23 is present at index 5

```
public class RecursiveBinarySearch
{
    public static int recursiveBinarySearch(int[] arr, int key, int left, int
right)
    {
         # write code here
    }

    public static void main(String[] args)
    {
         # write code here
    }
}
```

2.3 Uniform Binary Search

Uniform Binary Search is an optimization of Binary Search algorithm when many searches are made on same array or many arrays of same size. In normal binary search, we do arithmetic operations to find the mid points. Here we precompute mid points and fills them in lookup table. The array look-up generally works faster than arithmetic done (addition and shift) to find the mid-point.

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=3
Output: Position of 3 in array = 2

Input: array = {1, 3, 5, 6, 7, 8, 9}, v=7
Output: Position of 7 in array = 5

The algorithm is very similar to Binary Search algorithm, the only difference is a lookup table is created for an array and the lookup table is used to modify the index of the pointer in the array which makes the search faster. Instead of maintaining lower and upper bound the algorithm maintains an index and the index is modified using the lookup table.

```
public class RecursiveUniformBinarySearch
{
    public static int recursiveUniformBinarySearch(int[] arr, int key, int[]
lookupTable, int left, int right)
    {
        # write code here
    }
    public static void main(String[] args)
    {
        # write code here
    }
}
```

2.4 Interpolation Search

Interpolation search works better than Binary Search for a Sorted and Uniformly Distributed array. Binary search goes to the middle element to check irrespective of search-key. On the other hand, Interpolation search may go to different locations according to search-key. If the value of the search-key is close to the last element, Interpolation Search is likely to start search toward the end side. Interpolation search is more efficient than binary search when the elements in the list are uniformly distributed, while binary search is more efficient when the elements in the list are not uniformly distributed.

Interpolation search can take longer to implement than binary search, as it requires the use of additional calculations to estimate the position of the target element.

Input: arr = [1, 2, 3, 4, 5, 6, 7, 8, 9]
Output: target = 5

```
public class InterpolationSearch
{
    public static int interpolationSearch(int[] arr, int key)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

2.5 Fibonacci Search

Given a sorted array arr[] of size n and an element x to be searched in it. Return index of x if it is present in array else return -1.

Input: arr[] = {2, 3, 4, 10, 40}, x = 10
Output: 3
Element x is present at index 3.

Input: arr[] = {2, 3, 4, 10, 40}, x = 11
Output: -1
Element x is not present.

Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1. First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Fibonacci Search Procedure:
Let the searched element be x. The idea is to first find the smallest Fibonacci number that is greater than or equal to the length of the given array. Let the found Fibonacci number be fib (m'th Fibonacci number). We use (m-2)'th Fibonacci number as the index (If it is a valid index). Let (m-2)'th Fibonacci Number be i, we compare arr[i] with x, if x is same, we return i. Else if x is greater, we recur for subarray after i, else we recur for subarray before i.

Let arr[0..n-1] be the input array and the element to be searched be x.

1. Find the smallest Fibonacci number greater than or equal to n. Let this number be fibM [m'th Fibonacci number]. Let the two Fibonacci numbers preceding it be fibMm1 [(m-1)'th Fibonacci Number] and fibMm2 [(m-2)'th Fibonacci Number].
2. While the array has elements to be inspected:
    i. Compare x with the last element of the range covered by fibMm2
    ii. If x matches, return index
    iii. Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
    iv. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate the elimination of approximately front one-third of the remaining array.

3. Since there might be a single element remaining for comparison, check if fibMm1 is 1. If Yes, compare x with that remaining element. If match, return index.

```
public class FibonacciSearch
{
    public static int fibonacciSearch(int[] arr, int key)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

3. Sorting

3.1 Bubble Sort

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

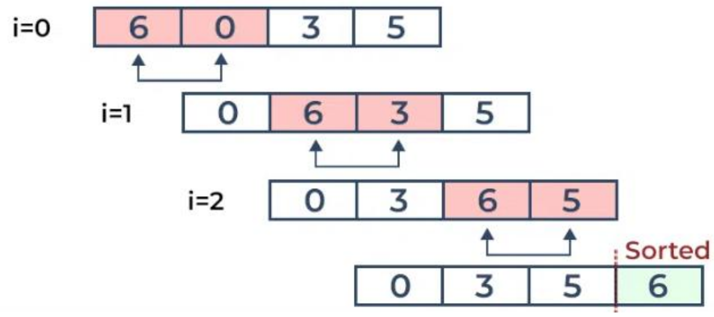
Bubble Sort Procedure:

1. Traverse from left and compare adjacent elements and the higher one is placed at right side.
2. In this way, the largest element is moved to the rightmost end at first.
3. This process is then continued to find the second largest and place it and so on until the data is sorted.

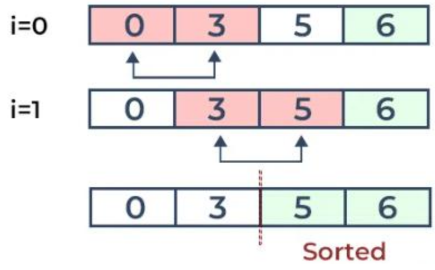
Input: arr = [6, 3, 0, 5]

Output:

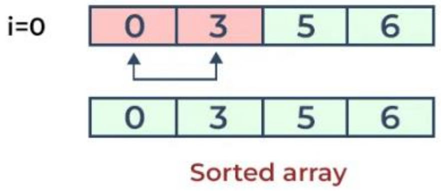
First Pass:



Second Pass:



Third Pass:



```
import java.util.Scanner;

class BubbleSortExample
{
    public static void main(String[] args)
    {
         # write code here

    }

    public static void bubbleSort(int[] arr)
    {
         # write code here

    }
}
```

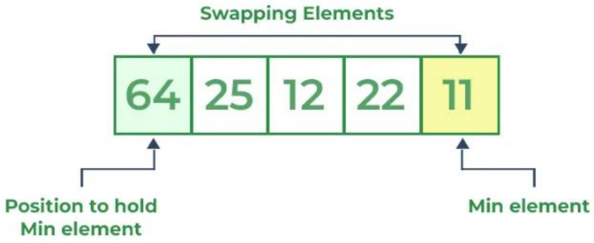
3.2 Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list. The algorithm repeatedly selects the smallest (or largest) element from the unsorted portion of the list and swaps it with the first element of the unsorted part. This process is repeated for the remaining unsorted portion until the entire list is sorted.

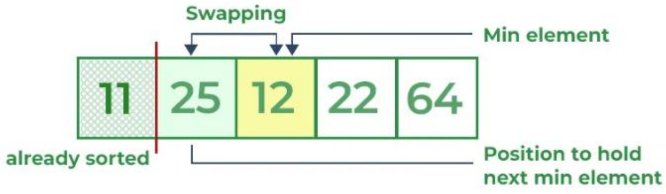
Input: arr = [64, 25, 12, 22, 11]

Output: arr = [11, 12, 22, 25, 64]

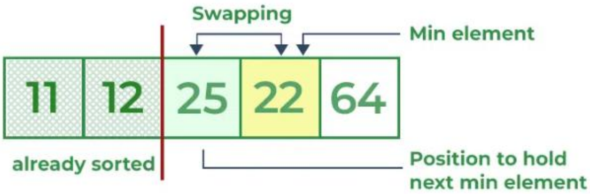
First Pass: For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value. Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.



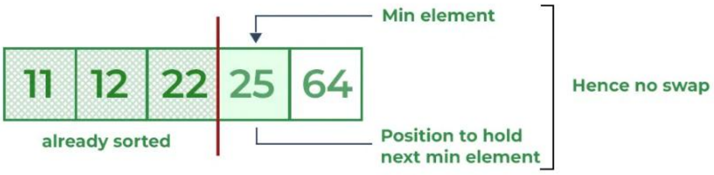
Second Pass: For the second position, where 25 is present, again traverse the rest of the array in a sequential manner. After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.



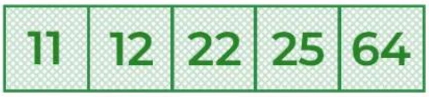
Third Pass: Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array. While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



Fourth Pass: Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array. As 25 is the 4th lowest value hence, it will place at the fourth position.



Fifth Pass: At last the largest value present in the array automatically get placed at the last position in the array. The resulted array is the sorted array.



Sorted array

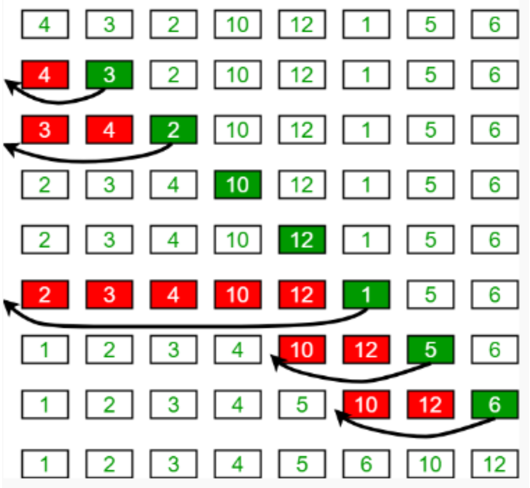
```
import java.util.Scanner;

class SelectionSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void selectionSort(int[] arr)
    {
        # write code here
    }
}
```

3.3 Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Insertion Sort Procedure:

1. To sort an array of size N in ascending order iterate over the array and compare the current element (key) to its predecessor, if the key element is smaller than its predecessor, compare it to the elements before.
2. Move the greater elements one position up to make space for the swapped element.



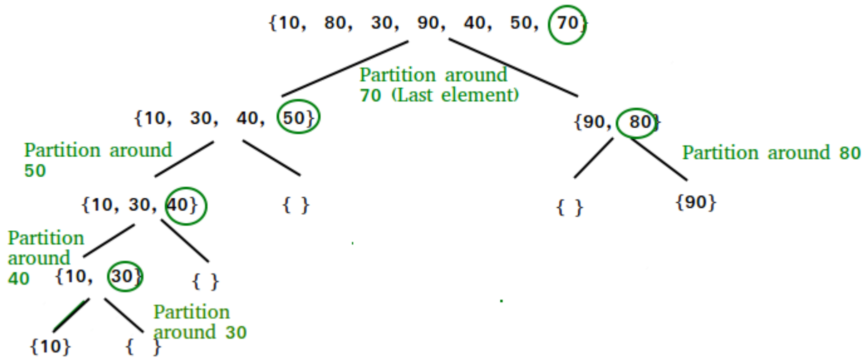
Input: arr = [4, 3, 2, 10, 12, 1, 5, 6]
Output: arr = [1, 2, 3, 4, 5, 6, 10, 12]

```
import java.util.Scanner;

class InsertionSortExample
{
    public static void main(String[] args)
    {
       # write code here
    }

    public static void insertionSort(int[] arr)
    {
        # write code here
    }
}
```

4. Divide and Conquer

4.1 Quick Sort

QuickSort is a sorting algorithm based on the Divide and Conquer algorithm that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. The key process in quickSort is a partition(). The target of partitions is to place the pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and put all smaller elements to the left of the pivot, and all greater elements to the right of the pivot. Partition is done recursively on each side of the pivot after the pivot is placed in its correct position and this finally sorts the array.



The quick sort method can be summarized in three steps:
1. **Pick:** Select a pivot element.
2. **Divide:** Split the problem set, move smaller parts to the left of the pivot and larger items to the right.
3. **Repeat and combine:** Repeat the steps and combine the arrays that have previously been sorted.

Algorithm for Quick Sort Function:

```
//start –> Starting index,  end --> Ending index
Quicksort(array, start, end)
{
        if (start < end)
        {
                pIndex = Partition(A, start, end)
                Quicksort(A,start,pIndex-1)
                Quicksort(A,pIndex+1, end)
        }
}
```

Algorithm for Partition Function:

```
partition (array, start, end)
{
        // Setting rightmost Index as pivot
        pivot = arr[end];

        i = (start - 1)  // Index of smaller element and indicates the
            // right position of pivot found so far
        for (j = start; j <= end- 1; j++)
        {
                // If current element is smaller than the pivot
                if (arr[j] < pivot)
                {
                        i++;    // increment index of smaller element
                        swap arr[i] and arr[j]
                }
        }
        swap arr[i + 1] and arr[end])
        return (i + 1)
```

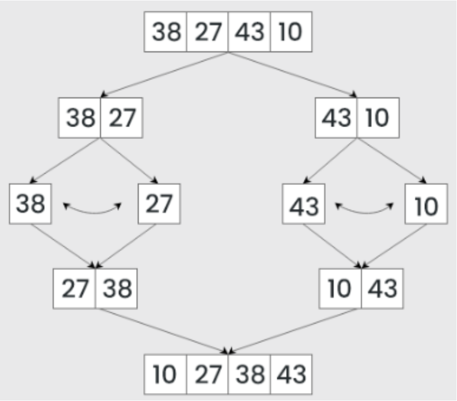
}

Input: arr = [10, 80, 30, 90, 40, 50, 70]
Output: arr = [10, 30, 40, 50, 70, 80, 90]

```
import java.util.Scanner;

class QuickSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void quickSort(int[] arr, int low, int high)
    {
        # write code here
    }

    public static int partition(int[] arr, int low, int high)
    {
        # write code here
    }
}
```

4.2 Merge Sort

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array. In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.



Input: arr = [12, 11, 13, 5, 6, 7]
Output: arr = [5, 6, 7, 11, 12, 13]

```

import java.util.Scanner;

class MergeSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void mergeSort(int[] arr, int low, int high)
    {
        # write code here
    }

    public static void merge(int[] arr, int low, int mid, int high)
    {
        # write code here
    }
}
```

4.3 Heap Sort

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to the selection sort where we first find the minimum element and place the minimum element at the beginning. Repeat the same process for the remaining elements.

Heap Sort Procedure:

First convert the array into heap data structure using heapify, then one by one delete the root node of the Max-heap and replace it with the last node in the heap and then heapify the root of the heap. Repeat this process until size of heap is greater than 1.

- Build a heap from the given input array.
- Repeat the following steps until the heap contains only one element:
  - Swap the root element of the heap (which is the largest element) with the last element of the heap.
  - Remove the last element of the heap (which is now in the correct position).
  - Heapify the remaining elements of the heap.
- The sorted array is obtained by reversing the order of the elements in the input array.

Input: arr = [12, 11, 13, 5, 6, 7]
Output: Sorted array is 5 6 7 11 12 13

```
import java.util.Scanner;

class HeapSortExample
{
    public static void main(String[] args)
    {
                # write code here
    }

    public static void heapSort(int[] arr)
    {
                # write code here
    }

    public static void heapify(int[] arr, int n, int i)
    {
        # write code here
    }
}
```

4.4 Radix Sort

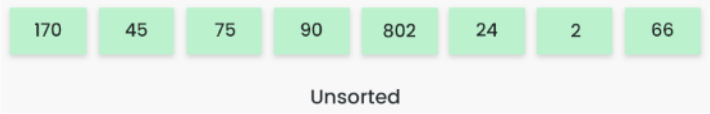
Radix Sort is a linear sorting algorithm that sorts elements by processing them digit by digit. It is an efficient sorting algorithm for integers or strings with fixed-size keys. Rather than comparing elements directly, Radix Sort distributes the elements into buckets based on each digit's value. By repeatedly sorting the elements by their significant digits, from the least significant to the most significant, Radix Sort achieves the final sorted order.

Radix Sort Procedure:

The key idea behind Radix Sort is to exploit the concept of place value.

1. It assumes that sorting numbers digit by digit will eventually result in a fully sorted list.
2. Radix Sort can be performed using different variations, such as Least Significant Digit (LSD) Radix Sort
   or Most Significant Digit (MSD) Radix Sort.

To perform radix sort on the array [170, 45, 75, 90, 802, 24, 2, 66], we follow these steps:



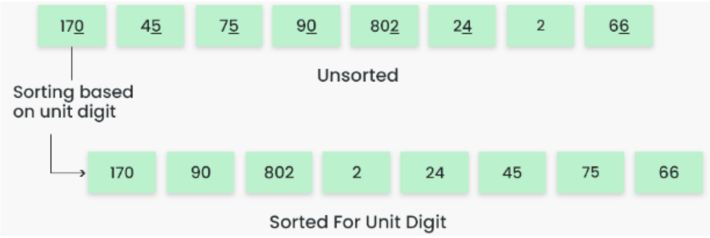
Step 1: Find the largest element in the array, which is 802. It has three digits, so we will iterate three times, once for each significant place.

Step 2: Sort the elements based on the unit place digits (X=0). We use a stable sorting technique, such as counting sort, to sort the digits at each significant place.

Sorting based on the unit place:

Perform counting sort on the array based on the unit place digits.

The sorted array based on the unit place is [170, 90, 802, 2, 24, 45, 75, 66]

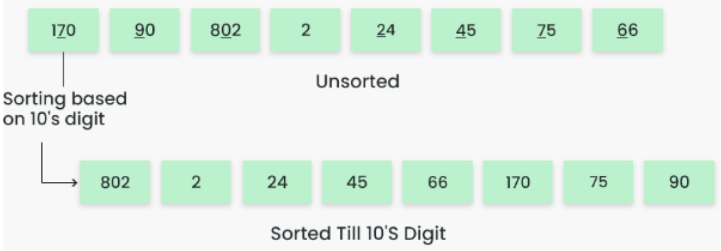


Step 3: Sort the elements based on the tens place digits.

Sorting based on the tens place:

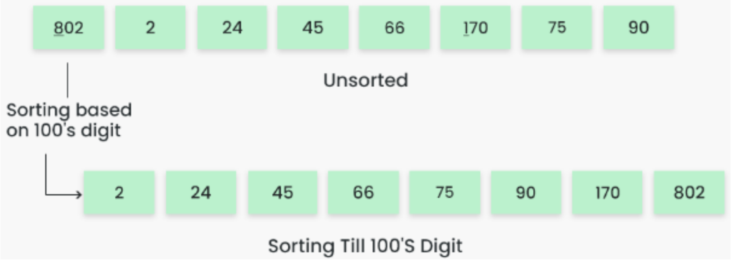
Perform counting sort on the array based on the tens place digits.

The sorted array based on the tens place is [802, 2, 24, 45, 66, 170, 75, 90]



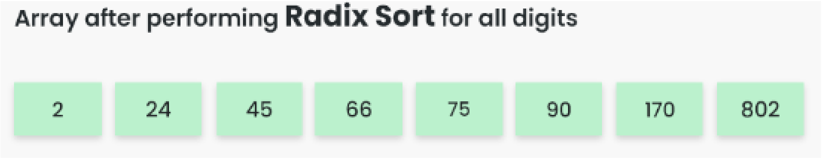
Step 4: Sort the elements based on the hundreds place digits.

Sorting based on the hundreds place:
Perform counting sort on the array based on the hundreds place digits.
The sorted array based on the hundreds place is [2, 24, 45, 66, 75, 90, 170, 802]



Step 5: The array is now sorted in ascending order.

The final sorted array using radix sort is [2, 24, 45, 66, 75, 90, 170, 802]



```
import java.util.Arrays;

class RadixSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void radixSort(int[] arr)
    {
        # write code here
    }

    public static int getMax(int[] arr)
    {
        # write code here
    }

    public static void countSort(int[] arr, int exp)
    {
        # write code here
    }
}
```

4.5 Shell Sort

Shell sort is mainly a variation of Insertion Sort. In insertion sort, we move elements only one position ahead. When an element has to be moved far ahead, many movements are involved. The idea of ShellSort is to allow the exchange of far items. In Shell sort, we make the array h-sorted for a large value of h. We keep reducing the value of h until it becomes 1. An array is said to be h-sorted if all sublists of every h'th element are sorted.

Shell Sort Procedure:

1. Initialize the value of gap size h
2. Divide the list into smaller sub-part. Each must have equal intervals to h
3. Sort these sub-lists using insertion sort
4. Repeat this step 1 until the list is sorted.
5. Print a sorted list.

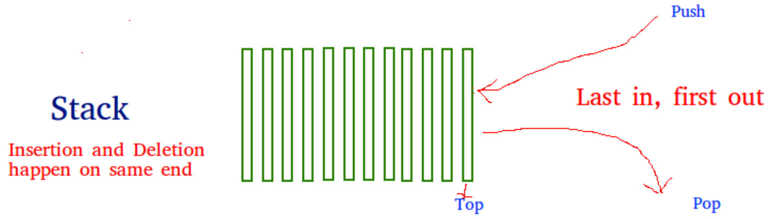
```
Procedure Shell_Sort(Array, N)
   While Gap < Length(Array) /3 :
                    Gap = ( Interval * 3 ) + 1
   End While Loop
   While Gap > 0 :
       For (Outer = Gap; Outer < Length(Array); Outer++):
             Insertion_Value = Array[Outer]
                    Inner = Outer;
             While Inner > Gap-1 And Array[Inner – Gap] >= Insertion_Value:
                    Array[Inner] = Array[Inner – Gap]
                    Inner = Inner – Gap
              End While Loop
                  Array[Inner] = Insertion_Value
       End For Loop
       Gap = (Gap -1) /3;
   End While Loop
End Shell_Sort
```

```
import java.util.Scanner;

class ShellSortExample
{
    public static void main(String[] args)
    {
        # write code here
    }

    public static void shellSort(int[] arr)
    {
        # write code here
    }
}
```

5. Stack

5.1 Implementation of Stack

A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty
- **size()** – Returns the size of the stack
- **top() / peek()** – Returns a reference to the topmost element of the stack
- **push(a)** – Inserts the element 'a' at the top of the stack
- **pop()** – Deletes the topmost element of the stack

```
class Stack
{
    private int maxSize;
    private int top;
    private int[] stackArray;

    public Stack(int size)
    {
        # write code here
    }

    public void push(int value)
    {
        # write code here
    }

    public int pop()
    {
        # write code here
    }

    public int peek()
    {
        # write code here
    }

    public boolean isEmpty()
    {
        # write code here
    }

    public boolean isFull()
    {
        # write code here
    }
}

class StackExample
{
    public static void main(String[] args)
    {
        Stack stack = new Stack(5);
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.pop();
        stack.peek();
        stack.push(40);
        stack.push(50);
        stack.push(60);
    }
}
```

5.2 Balanced Parenthesis Checking

Given an expression string, write a java program to find whether a given string has balanced parentheses or not.

Input: "{(a+b)*(c-d)}"

Output: true

Input: "{(a+b)*[c-d)}"

Output: false

One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return true otherwise, false

```
import java.util.Stack;

class BalancedParenthesisChecker
{
    public static boolean isBalanced(String expression)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        String expression1 = "{(a+b)*(c-d)}";
        String expression2 = "{(a+b)*[c-d)}";

        # write code here
    }
}
```

5.3 Evaluation of Postfix Expression

Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form “a b operator” (ab+) i.e., when a pair of operands is followed by an operator.

Input: str = “2 3 1 * + 9 -“

Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = “100 200 + 2 / 5 * 7 +”

Output: 757

Procedure for evaluation postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

```
import java.util.Stack;

class PostfixEvaluator
{
    public static int evaluatePostfix(String expression)
    {
        # write code here
    }

    public static int performOperation(char operator, int operand1, int operand2)
    {
        # write code here
    }

    public static void main(String[] args)
    {
        # write code here
    }
}
```

5.4 Infix to Postfix Expression Conversion

For a given Infix expression, convert it into Postfix form.

Infix expression: The expression of the form “a operator b” (a + b) i.e., when an operator is in-between every pair of operands.

Postfix expression: The expression of the form “a b operator” (ab+) i.e., When every pair of operands is followed by an operator.

Infix to postfix expression conversion procedure:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, put it in the postfix expression.
3. Otherwise, do the following

- If the precedence and associativity of the scanned operator are greater than the precedence and associativity of the operator in the stack [or the stack is empty or the stack contains a ‘(‘], then push it in the stack. [‘^‘ operator is right associative and other operators like ‘+‘,’–‘,’*‘ and ‘/‘ are left-associative].
    - Check especially for a condition when the operator at the top of the stack and the scanned operator both are ‘^‘. In this condition, the precedence of the scanned operator is higher due to its right associativity. So it will be pushed into the operator stack.
    - In all the other cases when the top of the operator stack is the same as the scanned operator, then pop the operator from the stack because of left associativity due to which the scanned operator has less precedence.

- Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator.
    - After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is a '(', push it to the stack.
5. If the scanned character is a ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-5 until the infix expression is scanned.
7. Once the scanning is over, Pop the stack and add the operators in the postfix expression until it is not empty.
8. Finally, print the postfix expression.

Input: A + B * C + D
Output: A B C * + D +

Input: ((A + B) – C * (D / E)) + F
Output: A B + C D E / * - F +

```
import java.util.Stack;

class Conversion
        {

      # Write Code Here

        }
```

5.5 Reverse a Stack

The stack is a linear data structure which works on the LIFO concept. LIFO stands for last in first out. In the stack, the insertion and deletion are possible at one end the end is called the top of the stack. Define two recursive functions BottomInsertion() and Reverse() to reverse a stack using Python. Define some basic function of the stack like push(), pop(), show(), empty(), for basic operation like respectively append an item in stack, remove an item in stack, display the stack, check the given stack is empty or not.

BottomInsertion(): this method append element at the bottom of the stack and BottomInsertion accept two values as an argument first is stack and the second is elements, this is a recursive method.

Reverse(): the method is reverse elements of the stack, this method accept stack as an argument Reverse() is also a Recursive() function. Reverse() is invoked BottomInsertion() method for completing the reverse operation on the stack.

Input: Elements = [1, 2, 3, 4, 5]
Output: Original Stack
5
4
3
2
1
Stack after Reversing
1
2
3
4

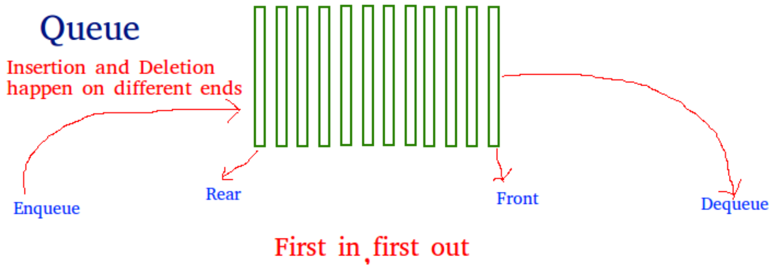
5

```
 import java.util.Stack;
class StackClass {
        # Write Code Here
    }
```

6. Queue

6.1 Linear Queue

Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



```
import java.util.Scanner;

public class LinearQueue
       {
       # Write Code Here
       }
    public static boolean isEmpty() {
        return front == rear;
    }
    public static boolean isFull() {
        return rear == MAX;
    }
    public static void enqueue(int item)
       {
       # Write Code Here
       }

    public static void dequeue()
       {
       # Write Code Here
       }
public static void display()
       {
        # Write Code Here
       }
    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

6.2 Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Input:

["MyStack", "push", "push", "top", "pop", "empty"]
[[], [1], [2], [], [], []]

Output:

[null, null, null, 2, 2, false]

```
import java.util.LinkedList;
import java.util.Queue;

class MyStack
          {
             # Write Code Here
          }
    public void push(int x)
      {
        # Write Code Here
      }
    }

    public int pop()
      {
        return queue.remove();
      }

    public int top() {
        return queue.peek();
    }

    public boolean empty() {
        return queue.isEmpty();
    }
    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

6.3 Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Input:

["MyQueue", "push", "push", "peek", "pop", "empty"]
[[], [1], [2], [], [], []]

Output:
[null, null, null, 1, 1, false]

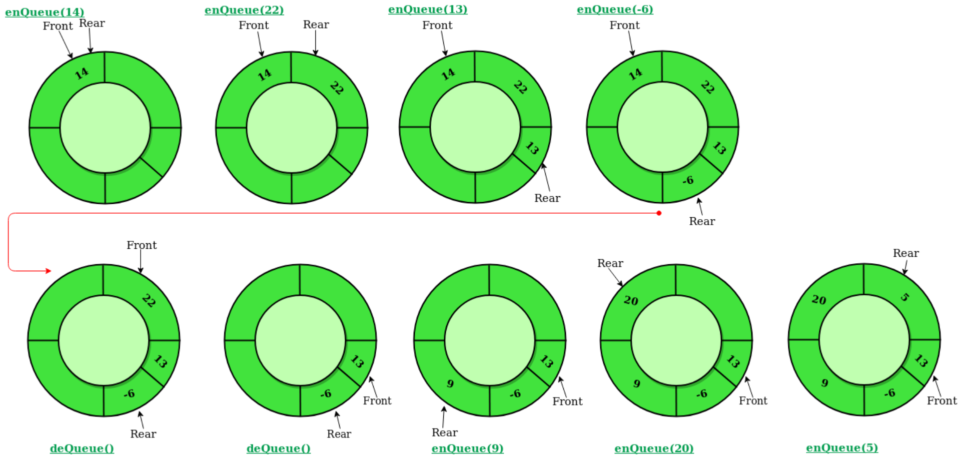
```
import java.util.Stack;

class MyQueue {

    private Stack<Integer> stack1;
    private Stack<Integer> stack2;
    public MyQueue() {
        stack1 = new Stack<>();
        stack2 = new Stack<>();
    }
    public void push(int x) {
        stack1.push(x);
    }
    public int pop()
      {
        # Write Code Here
      }
    public int peek()
      {
      # Write Code Here
      }
    public boolean empty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }
    public static void main(String[] args)
      {
        # Write Code Here
      }
}
```

6.4 Circular Queue

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
        - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.



Implement Circular Queue using Array:

1. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
2. Initialize two variables front and rear to -1.
3. **Enqueue:** To enqueue an element **x** into the queue, do the following:
   - Increment rear by 1.
     - If **rear** is equal to n, set **rear** to 0.
   - If **front** is -1, set **front** to 0.
   - Set queue[rear] to x.
4. **Dequeue:** To dequeue an element from the queue, do the following:
   - Check if the queue is empty by checking if **front** is -1.
     - If it is, return an error message indicating that the queue is empty.
   - Set **x** to queue [front].
   - If **front** is equal to **rear**, set **front** and **rear** to -1.
   - Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
   - Return x.

```
class CircularQueue {

    private int size;
    private int front, rear;
    private int[] queue;

    public CircularQueue(int size) {
        this.size = size;
        this.queue = new int[size];
        this.front = this.rear = -1;
    }


    public void enqueue(int data)
      {
      # Write Code Here
      }
    public int dequeue()
      {
        # Write Code Here
      }
    public void display()
      {
```

```
      # Write Code Here
      }
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

6.5 Deque (Doubly Ended Queue)

In a Deque (Doubly Ended Queue), one can perform insert (append) and delete (pop) operations from both the ends of the container. There are two types of Deque:

1. **Input Restricted Deque:** Input is limited at one end while deletion is permitted at both ends.
2. **Output Restricted Deque:** Output is limited at one end but insertion is permitted at both ends.

Operations on Deque:

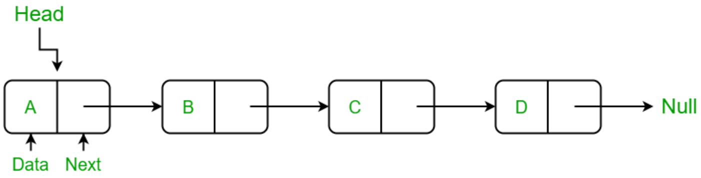
1. **append():** This function is used to insert the value in its argument to the right end of the deque.
2. **appendleft():** This function is used to insert the value in its argument to the left end of the deque.
3. **pop():** This function is used to delete an argument from the right end of the deque.
4. **popleft():** This function is used to delete an argument from the left end of the deque.
5. **index(ele, beg, end):** This function returns the first index of the value mentioned in arguments, starting searching from beg till end index.
6. **insert(i, a):** This function inserts the value mentioned in arguments(a) at index(i) specified in arguments.
7. **remove():** This function removes the first occurrence of the value mentioned in arguments.
8. **count():** This function counts the number of occurrences of value mentioned in arguments.
9. **len(dequeue):** Return the current size of the dequeue.
10. **Deque[0]:** We can access the front element of the deque using indexing with de[0].
11. **Deque[-1]:** We can access the back element of the deque using indexing with de[-1].
12. **extend(iterable):** This function is used to add multiple values at the right end of the deque. The argument passed is iterable.
13. **extendleft(iterable):** This function is used to add multiple values at the left end of the deque. The argument passed is iterable. Order is reversed as a result of left appends.
14. **reverse():** This function is used to reverse the order of deque elements.
15. **rotate():** This function rotates the deque by the number specified in arguments. If the number specified is negative, rotation occurs to the left. Else rotation is to right.

```
import java.util.ArrayDeque;
import java.util.Deque;

public class DequeOperations
{
    # Write Code Here
}
```

7. Linked List

7.1 Singly Linked List

A singly linked list is a linear data structure in which the elements are not stored in contiguous memory locations and each element is connected only to its next element using a pointer.



Creating a linked list involves the following operations:

1. Creating a Node class:
2. Insertion at beginning:
3. Insertion at end
4. Insertion at middle
5. Update the node
6. Deletion at beginning
7. Deletion at end
8. Deletion at middle
9. Remove last node
10. Linked list traversal
11. Get length

```
class Node {
    String data;
    Node next;

    Node(String data) {
        this.data = data;
        this.next = null;
    }
}

class LinkedList
       {
       # Write Code Here
       }

    public void insertAtEnd(String data)
       {
        # Write Code Here
      }

    public void updateNode(String val, int index)
       {
       # Write Code Here
       }

    public void remove_first_node() {
        # Write Code Here
    }

    public void remove_last_node()
       {
```

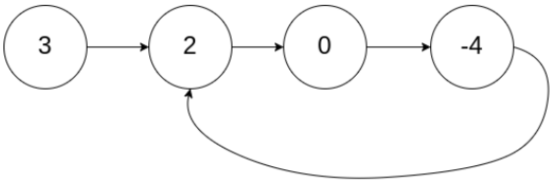
```
       # Write Code Here
       }

    public void remove_at_index(int index)

        {
      # Write Code Here
        }

    public void remove_node(String data)
       {
        # Write Code Here
        }

    public int sizeOfLL()
       {
     # Write Code Here
       }

    public void printLL()
       {
        # Write Code Here
       }

    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

7.2 Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

Note that pos is not passed as a parameter.

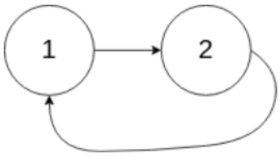
Return true if there is a cycle in the linked list. Otherwise, return false.



Input: head = [3, 2, 0, -4], pos = 1

Output: true

Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).



Input: head = [1, 2], pos = 0

Output: true

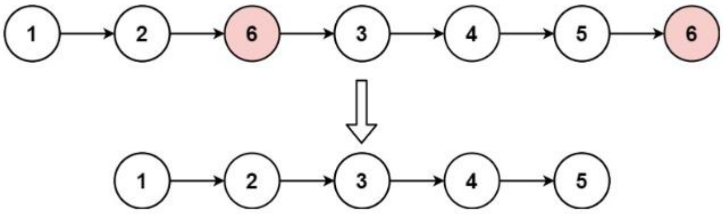
Explanation: There is a cycle in the linked list, where the tail connects to the 0th node.

Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.

```
class ListNode
       {
    # Write Code Here
       }

public class Solution
       {
        # Write Code Here
       }
}
```

7.3 Remove Linked List Elements

Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return the new head.



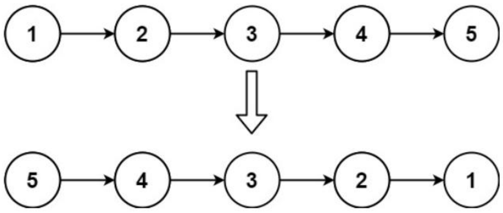
Input: head = [1, 2, 6, 3, 4, 5, 6], val = 6
Output: [1, 2, 3, 4, 5]
Input: head = [ ], val = 1
Output: [ ]
Input: head = [7, 7, 7, 7], val = 7
Output: [ ]

```
class ListNode {
    # Write Code Here
    }
}
public class Solution {
    public boolean hasCycle(ListNode head)
       {
        # Write Code Here
       }

    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

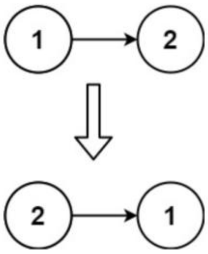
7.4 Reverse Linked List

Given the head of a singly linked list, reverse the list, and return the reversed list.

Input: head = [1, 2, 3, 4, 5]
Output: [5, 4, 3, 2, 1]



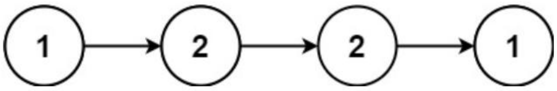
Input: head = [1, 2]
Output: [2, 1]



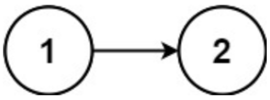
```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class Solution {
    public ListNode reverseList(ListNode head)
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
```

7.5 Palindrome Linked List

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.



Input: head = [1, 2, 2, 1]
Output: true

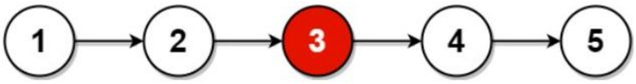


Input: head = [1, 2]
Output: false

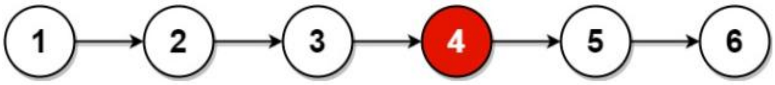
```
class ListNode
      {
      # Write Code Here
      }
public class Solution {
    public boolean isPalindrome(ListNode head)
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

7.6 Middle of the Linked List

Given the head of a singly linked list, return the middle node of the linked list. If there are two middle nodes, return the second middle node.



Input: head = [1, 2, 3, 4, 5]
Output: [3, 4, 5]
Explanation: The middle node of the list is node 3.



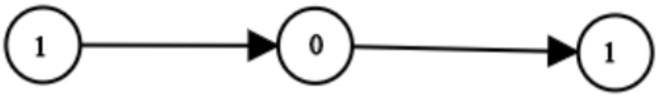
Input: head = [1, 2, 3, 4, 5, 6]
Output: [4, 5, 6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.

```
class ListNode
      {
      # Write Code Here
      }
}
public class Solution
    {
      # Write Code Here
    }

    public static void main(String[] args) {
       # Write Code Here
    }
}
```

7.7 Convert Binary Number in a Linked List to Integer

Given head which is a reference node to a singly-linked list. The value of each node in the linked list is either 0 or 1. The linked list holds the binary representation of a number.

Return the decimal value of the number in the linked list. The most significant bit is at the head of the linked list.



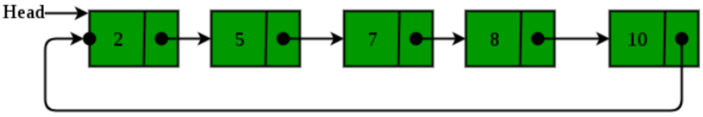
Input: head = [1, 0, 1]
Output: 5
Explanation: (101) in base 2 = (5) in base 10
Input: head = [0]
Output: 0

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int val) {
        this.val = val;
        this.next = null;
    }
}

public class Solution
       {
          # Write Code Here
       }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

8. Circular Single Linked List and Doubly Linked List

8.1 Circular Linked List

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end.



Operations on the circular linked list:

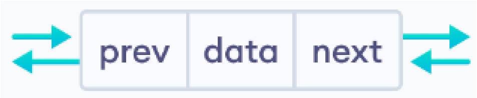
1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

```
import java.util.ArrayList;
public class Main{
        static class Node{
                int data;
                Node next;
                Node(int data){
                        this.data = data;
                        this.next = null;
                }
        }
        static class CircularLinkedList
        {
        # Write Code Here
        }
        Node addAfter(int data, int item)
        {
        # Write Code Here
        }
void deleteNode(Node last, int key)
        {
        # Write Code Here
        }
                        System.
```

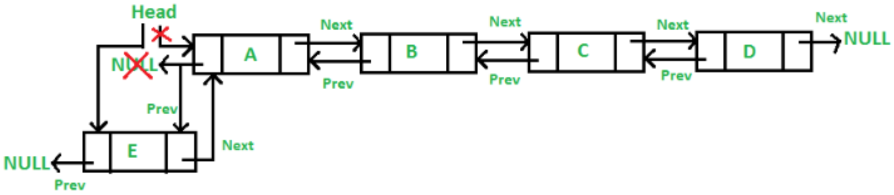
8.2 Doubly Linked List

The A doubly linked list is a type of linked list in which each node consists of 3 components:

1. *prev - address of the previous node
2. data - data item
3. *next - address of next node.



Double Linked List Node



Operations on the Double Linked List:

1. Insertion at the beginning
2. Insertion at the end
3. Insertion in between the nodes
4. Deletion at the beginning
5. Deletion at the end
6. Deletion in between the nodes
7. Traversal

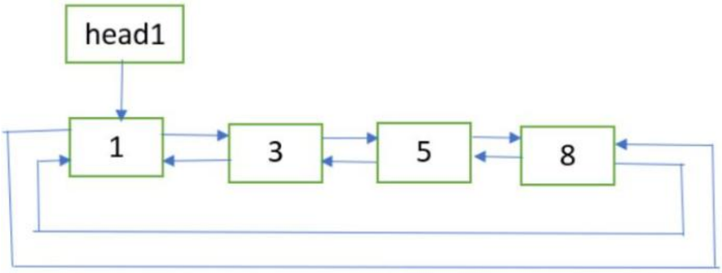
```
import java.util.Scanner;

class Node {
    int data;
    Node next;
    Node prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

class DLinkedList {
    Node head;
    int ctr;

    DLinkedList() {
        this.head = null;
        this.ctr = 0;
    }

    void insertBeg(int data)
       {
       # Write Code Here
       }
    void insertEnd(int data)
       {
       # Write Code Here
       }

    void deleteBeg()
       {
       # Write Code Here
       }
    void deleteEnd()
       {
        # Write Code Here
       }

    void insertPos(int pos, int data)
       {
       # Write Code Here
       }

    void deletePos(int pos)
       {
        # Write Code Here
       }

    void traverseF()
       {
        # Write Code Here
       }

    void traverseR()
```

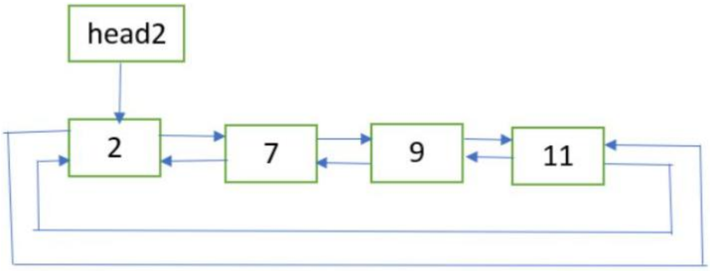
```
        {
         # Write Code Here
        }

public class Main {
    public static void main(String[] args)
        {
        # Write Code Here
        }
}
```

8.3 Sorted Merge of Two Sorted Doubly Circular Linked Lists

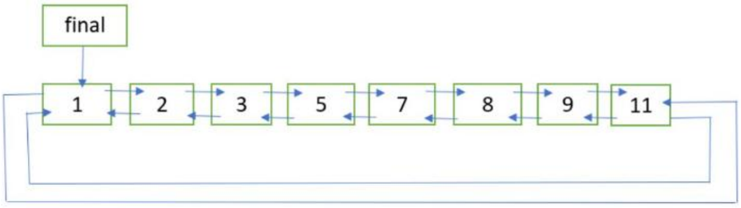
Given two sorted Doubly circular Linked List containing n1 and n2 nodes respectively. The problem is to merge the two lists such that resultant list is also in sorted order.

Input: List 1 and List 2





Output: Merged List



Procedure for Merging Doubly Linked List:

1. If head1 == NULL, return head2.
2. If head2 == NULL, return head1.
3. Let **last1** and **last2** be the last nodes of the two lists respectively. They can be obtained with the help of the previous links of the first nodes.
4. Get pointer to the node which will be the last node of the final list. If last1.data < last2.data, then **last_node** = last2, Else **last_node** = last1.
5. Update last1.next = last2.next = NULL.
6. Now merge the two lists as two sorted doubly linked list are being merged.
   Refer **merge** procedure of this post. Let the first node of the final list be **finalHead**.

7. Update finalHead.prev = last_node and last_node.next = finalHead.
8. Return **finalHead**.

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class SortedMergeDoublyCircularLinkedList
  {
# Write Code Here
         }

    static Node mergeUtil(Node head1, Node head2)
  {
# Write Code Here
  }

    static void printList(Node head)
  {
# Write Code Here
  }

    public static void main(String[] args)
  {
        # Write Code Here
         }
}
```

8.4 Delete all occurrences of a given key in a Doubly Linked List

Given a doubly linked list and a key x. The problem is to delete all occurrences of the given key x from the doubly linked list.

Input: 2 <-> 2 <-> 10 <-> 8 <-> 4 <-> 2 <-> 5 <-> 2
x = 2
Output: 10 <-> 8 <-> 4 <-> 5

Algorithm:

```
delAllOccurOfGivenKey (head_ref, x)
   if head_ref == NULL
      return
   Initialize current = head_ref
   Declare next
   while current != NULL
      if current->data == x
         next = current->next
         deleteNode(head_ref, current)
         current = next
      else
```

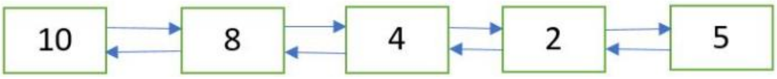
current = current->next

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class DeleteOccurrenceInDoublyLinkedList
      {
       # Write Code Here
      }

    static Node deleteAllOccurOfX(Node head, int x)
      {
      # Write Code Here
      }

    static void printList(Node head)
      {
       # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

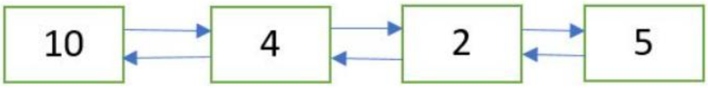
8.5 Delete a Doubly Linked List Node at a Given Position

Given a doubly linked list and a position n. The task is to delete the node at the given position n from the beginning.

Input: Initial doubly linked list



Output: Doubly Linked List after deletion of node at position n = 2



Procedure:

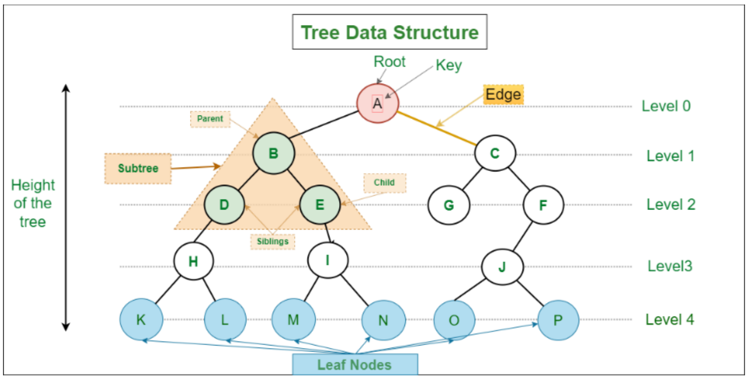
1. Get the pointer to the node at position n by traversing the doubly linked list up to the nth node from the beginning.
2. Delete the node using the pointer obtained in Step 1.

```
class Node {
    int data;
    Node next, prev;

    Node(int data) {
        this.data = data;
        this.next = null;
        this.prev = null;
    }
}

public class DeleteNodeAtGivenPosition
 {
          # Write Code Here
        }

    static Node deleteNode(Node head, Node del)
 {
        # Write Code Here
        }

    static Node deleteNodeAtGivenPos(Node head, int n)
 {
        # Write Code Here
        }
    static void printList(Node head)
 {
 # Write Code Here
        }
}
```

9. Trees

9.1 Tree Creation and Basic Tree Terminologies

A tree data structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are connected by edges and has a hierarchical relationship between the nodes.



Basic Terminologies in Tree:

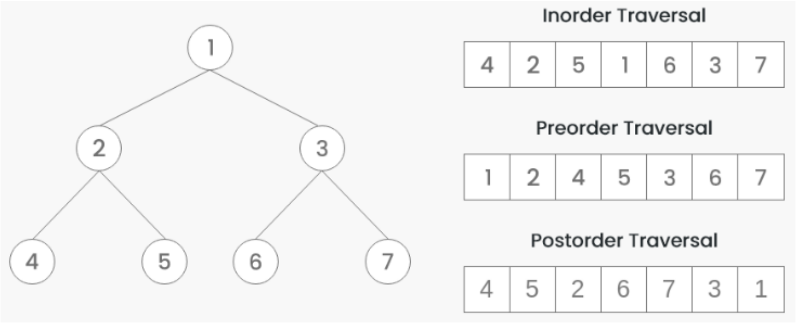
1. Parent Node: The node which is a predecessor of a node is called the parent node of that node. {B} is the parent node of {D, E}.
2. Child Node: The node which is the immediate successor of a node is called the child node of that node. Examples: {D, E} are the child nodes of {B}.
3. Root Node: The topmost node of a tree or the node which does not have any parent node is called the root node. {A} is the root node of the tree. A non-empty tree must contain exactly one root node and exactly one path from the root to all other nodes of the tree.
4. Leaf Node or External Node: The nodes which do not have any child nodes are called leaf nodes. {K, L, M, N, O, P} are the leaf nodes of the tree.
5. Ancestor of a Node: Any predecessor nodes on the path of the root to that node are called Ancestors of that node. {A, B} are the ancestor nodes of the node {E}
6. Descendant: Any successor node on the path from the leaf node to that node. {E, I} are the descendants of the node {B}.
7. Sibling: Children of the same parent node are called siblings. {D, E} are called siblings.
8. Level of a node: The count of edges on the path from the root node to that node. The root node has level 0.
9. Internal node: A node with at least one child is called Internal Node.
10. Neighbour of a Node: Parent or child nodes of that node are called neighbors of that node.
11. Subtree: Any node of the tree along with its descendant.

```
import java.util.ArrayList;
import java.util.List;

public class TreeBasicTerminologies
       {

        # Write Code Here
       }
static void printChildren(int root, List<List<Integer>> adj)
       {
        # Write Code Here
       }
    static void printLeafNodes(int root, List<List<Integer>> adj)
       {
        # Write Code Here
       }
    static void printDegrees(int root, List<List<Integer>> adj)
       {
        # Write Code Here
       }
    public static void main(String[] args)
       {
        # Write Code Here
       }
}
```

9.2 Binary Tree Traversal Techniques

A binary tree data structure can be traversed in following ways:

1. Inorder Traversal
2. Preorder Traversal
3. Postorder Traversal
4. Level Order Traversal



Inorder Traversal

4	2	5	1	6	3	7

Preorder Traversal

1	2	4	5	3	6	7

Postorder Traversal

4	5	2	6	7	3	1

Algorithm Inorder (tree)

1. Traverse the left subtree, i.e., call Inorder(left->subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right->subtree)

Algorithm Preorder (tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left->subtree)
3. Traverse the right subtree, i.e., call Preorder(right->subtree)

Algorithm Postorder (tree)

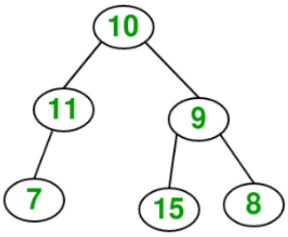
1. Traverse the left subtree, i.e., call Postorder(left->subtree)
2. Traverse the right subtree, i.e., call Postorder(right->subtree)
3. Visit the root.

```
import java.util.Scanner;
class Node
       {
         # Write Code Here
       }
class BT {
        Node root;
        BT() {
        this.root = null;
    }
    void insert(int data)
       {
       # Write Code Here
       }
    Node insertRec(Node root, int data)
       {
        # Write Code Here
       }
    void postorder(Node root)
       {
        # Write Code Here
       }
    void preorder(Node root)
       {
        # Write Code Here
      }
    }
    void inorder(Node root)
       {
       # Write Code Here
       }
}

public class BinaryTreeTraversal {
    public static void main(String[] args)
             {
        # Write Code Here
            }
        }
    }
}
```

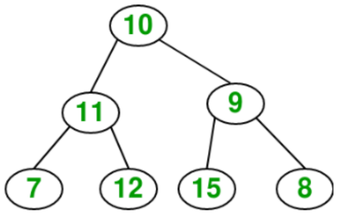
9.3 Insertion in a Binary Tree in Level Order

Given a binary tree and a key, insert the key into the binary tree at the first position available in level order.

Input: Consider the tree given below



Output:



After inserting 12

The idea is to do an iterative level order traversal of the given tree using queue. If we find a node whose left child is empty, we make a new key as the left child of the node. Else if we find a node whose right child is empty, we make the new key as the right child. We keep traversing the tree until we find a node whose either left or right child is empty.

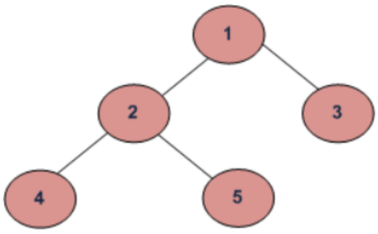
```
class Node
{
# Write Code Here
}

public class BinaryTreeInsertion
{
# Write Code Here
       }
    static Node insert(Node root, int key)
{
# Write Code Here
       }

    public static void main(String[] args)
{
        # Write Code Here
       }
}
```

9.4 Finding the Maximum Height or Depth of a Binary Tree

Given a binary tree, the task is to find the height of the tree. The height of the tree is the number of edges in the tree from the root to the deepest node.

Note: The height of an empty tree is 0.

Input: Consider the tree below



Recursively calculate the height of the left and the right subtrees of a node and assign height to the node as max of the heights of two children plus 1.

maxDepth('1') = max(maxDepth('2'), maxDepth('3')) + 1 = 2 + 1

because recursively
maxDepth('2') = max (maxDepth('4'), maxDepth('5')) + 1 = 1 + 1 and (as height of both '4' and '5' are 1)
maxDepth('3') = 1

Procedure:

- Recursively do a Depth-first search.
- If the tree is empty then return 0
- Otherwise, do the following
    - Get the max depth of the left subtree recursively i.e. call maxDepth(tree->left-subtree)
    - Get the max depth of the right subtree recursively i.e. call maxDepth(tree->right-subtree)
    - Get the max of max depths of left and right subtrees and add 1 to it for the current node.

      $max_depth = max(maxdeptofleftsubtree, maxdepthofrightsubtree) + 1$
- Return max_depth.

```
class Node
{
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class MaximumDepthOfTree
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
        # Write Code Here
      }
```

9.5 Deletion in a Binary Tree

Given a binary tree, delete a node from it by making sure that the tree shrinks from the bottom (i.e. the deleted node is replaced by the bottom-most and rightmost node).

Input: Delete 10 in below tree

```
    10
   /  \
 20    30
```

Output:

```
    30
   /
 20
```

Input: Delete 20 in below tree

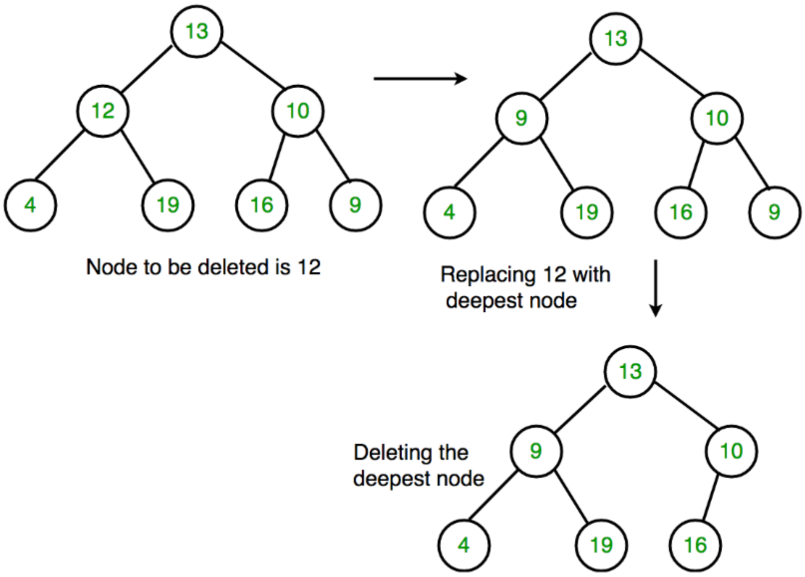
```
      10
    /   \
  20     30
           \
            40
```

Output:

```
      10
    /    \
  40      30
```

Algorithm:

1. Starting at the root, find the deepest and rightmost node in the binary tree and the node which we want to delete.
2. Replace the deepest rightmost node's data with the node to be deleted.
3. Then delete the deepest rightmost node.



```
class Node {
    int data;
    Node left, right;

    Node(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTreeDeletion
{
         # Write Code Here
       }
    static void deleteDeepest(Node root, Node dNode)
{
        # Write Code Here
}
    static Node deletion(Node root, int key)
{
        # Write Code Here
}


    public static void main(String[] args)
{
        # Write Code Here
}
}
```

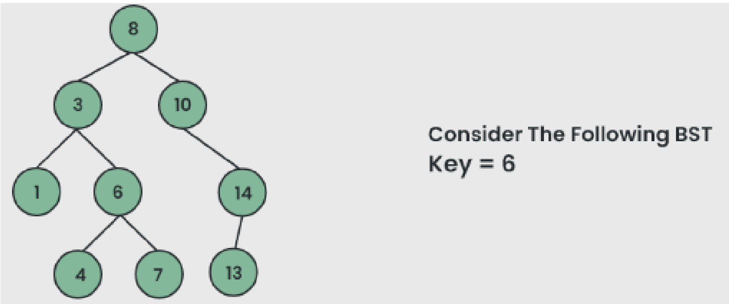
10. Binary Search Tree (BST)

10.1 Searching in Binary Search Tree

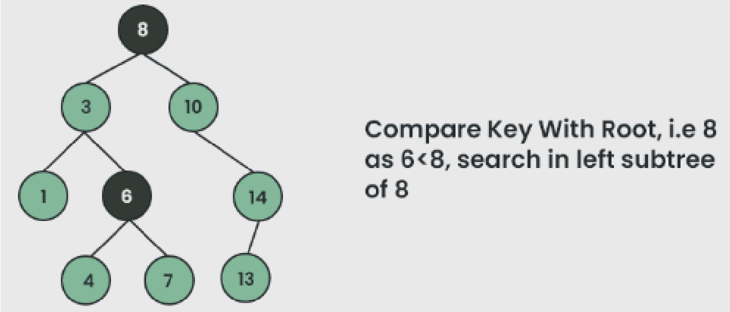
Given a BST, the task is to delete a node in this BST. For searching a value in BST, consider it as a sorted array. Perform search operation in BST using Binary Search Algorithm.

Algorithm to search for a key in a given Binary Search Tree:
Let's say we want to search for the number **X,** We start at the root. Then:

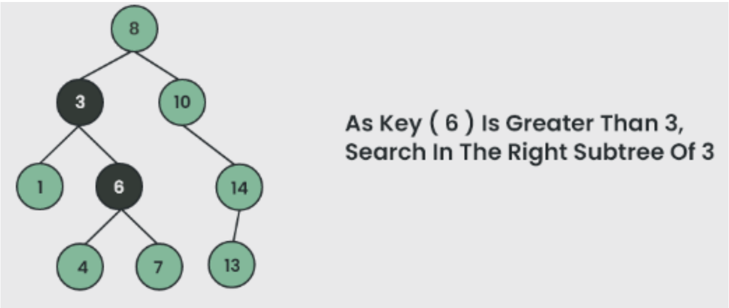
- We compare the value to be searched with the value of the root.
  - If it's equal we are done with the search if it's smaller we know that we need to go to the left subtree because in a binary search tree all the elements in the left subtree are smaller and all the elements in the right subtree are larger.
- Repeat the above step till no more traversal is possible
- If at any iteration, key is found, return True. Else False.



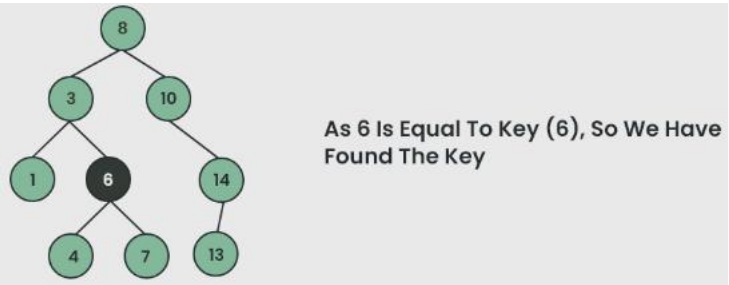
Consider The Following BST
Key = 6



Compare Key With Root, i.e 8 as 6<8, search in left subtree of 8



As Key (6) Is Greater Than 3, Search In The Right Subtree Of 3



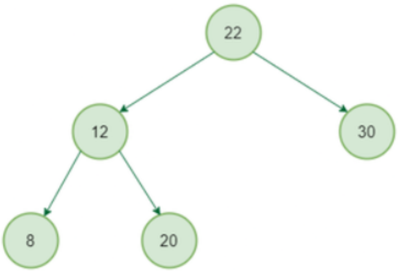
As 6 Is Equal To Key (6), So We Have Found The Key

```
// Node class to represent each node of the BST
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}
class BST
{
# Write Code Here
}
    Node search(int key) {
      return searchRec(root, key);
}
    Node searchRec(Node root, int key)
       {
       # Write Code Here
       }
    public static void main(String[] args)
{
        # Write Code Here
       }
}
```

10.2 Find the node with Minimum Value in a BST

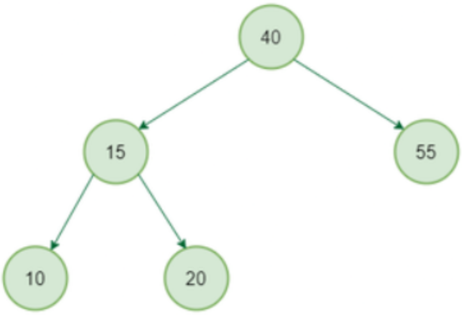
Write a function to find the node with minimum value in a Binary Search Tree.

Input: Consider the tree given below



Output: 8

Input: Consider the tree given below



Output: 10

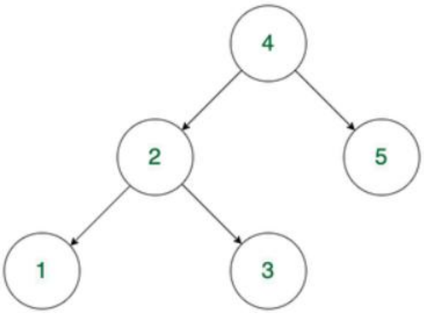
```
import java.util.ArrayList;
import java.util.List;

class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinarySearchTree
      {
        # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

10.3 Check if a Binary Tree is BST or not

A binary search tree (BST) is a node-based binary tree data structure that has the following properties.

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. Both the left and right subtrees must also be binary search trees.
4. Each node (item in the tree) has a distinct key.

Input: Consider the tree given below



Output: Check if max value in left subtree is smaller than the node and min value in right subtree greater than the node, then print it "Is BST" otherwise "Not a BST"

Procedure:
1. If the current node is null then return true
2. If the value of the left child of the node is greater than or equal to the current node then return false
3. If the value of the right child of the node is less than or equal to the current node then return false
4. If the left subtree or the right subtree is not a BST then return false
5. Else return true

```
class Node {
    int data;
    Node left, right;

    public Node(int item) {
        data = item;
        left = right = null;
    }
}

class BinaryTree
      {
      # Write Code Here
      }

    boolean isBST(Node node) {
        return isBSTUtil(node, Integer.MIN_VALUE, Integer.MAX_VALUE);
    }

    boolean isBSTUtil(Node node, int min, int max)
      {
      # Write Code Here
      }

    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

10.4 Second Largest Element in BST

Given a Binary search tree (BST), find the second largest element.

Input: Root of below BST

```
         10
        /
      5
```

Output: 5

Input: Root of below BST

```
        10

       /  \

     5     20

             \

              30
```

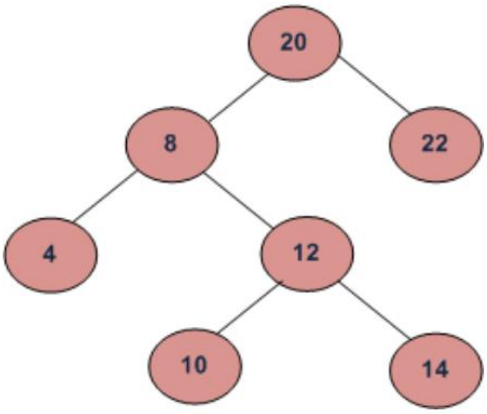
Output: 20

Procedure: The second largest element is second last element in inorder traversal and second element in reverse inorder traversal. We traverse given Binary Search Tree in reverse inorder and keep track of counts of nodes visited. Once the count becomes 2, we print the node.

```
class Node {
    int key;
    Node left, right;

    public Node(int item) {
        key = item;
        left = right = null;
    }
}

class BinarySearchTree
      {
      # Write Code Here
      }

        secondLargestUtil(node.right);
        count++;

        // If count is equal to 2 then this is the second largest
        if (count == 2) {
            System.out.println("The second largest element is " + node.key);
            return;
        }

        secondLargestUtil(node.left);
    }

    // Function to find the second largest element
    void secondLargest(Node node) {
        count = 0;
        secondLargestUtil(node);
    }

    // Driver code
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

Try:

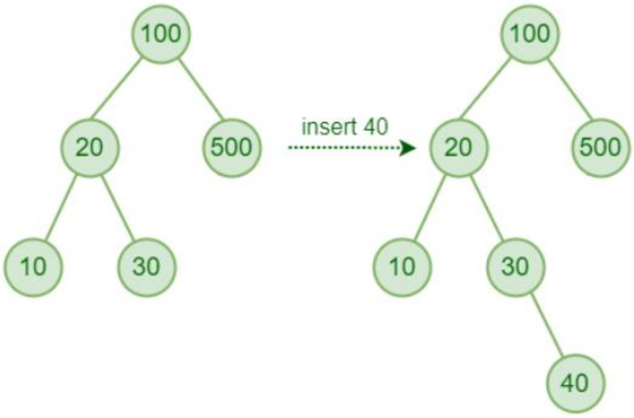
1. **Kth largest element in BST when modification to BST is not allowed:** Given a Binary Search Tree (BST) and a positive integer k, find the k'th largest element in the Binary Search Tree. For a given BST, if k = 3, then output should be 14, and if k = 5, then output should be 10.



10.5 Insertion in Binary Search Tree (BST)

Given a Binary search tree (BST), the task is to insert a new node in this BST.

Input: Consider a BST and insert the element 40 into it.



Procedure for inserting a value in a BST:

A new key is always inserted at the leaf by maintaining the property of the binary search tree. We start searching for a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node. The below steps are followed while we try to insert a node into a binary search tree:

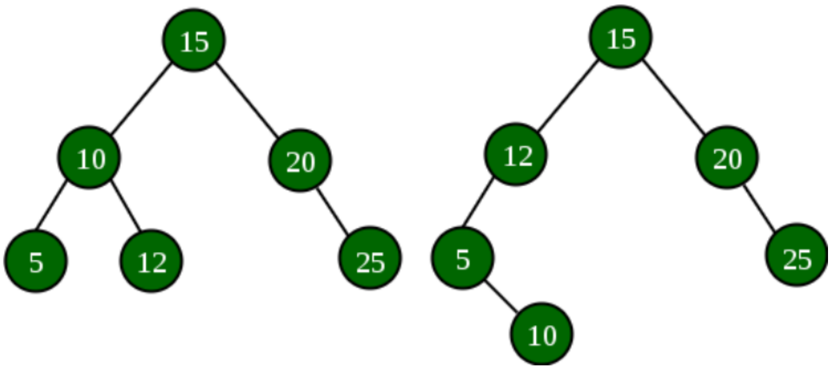
- Check the value to be inserted (say X) with the value of the current node (say val) we are in:
  - If X is less than val move to the left subtree.
  - Otherwise, move to the right subtree.
- Once the leaf node is reached, insert X to its right or left based on the relation between X and the leaf node's value.

```
// A utility class that represents an individual node in a BST
class Node {
    int val;
    Node left, right;

    public Node(int item) {
        val = item;
        left = right = null;
    }
}

class BinarySearchTree
       {

       # Write Code Here

       }

    void inorder()
       {
        inorderRec(root);
       }

    void inorderRec(Node root)
       {
         # Write Code Here

        }
    }

    // Driver code
    public static void main(String[] args)
       {
         # Write Code Here
       }
}
```

Try:

1. **Check if two BSTs contain same set of elements:** Given two Binary Search Trees consisting of unique positive elements, we have to check whether the two BSTs contain the same set of elements or not.

Input: Consider two BSTs which contains same set of elements {5, 10, 12, 15, 20, 25}, but the structure of the two given BSTs can be different.



11. AVL Tree

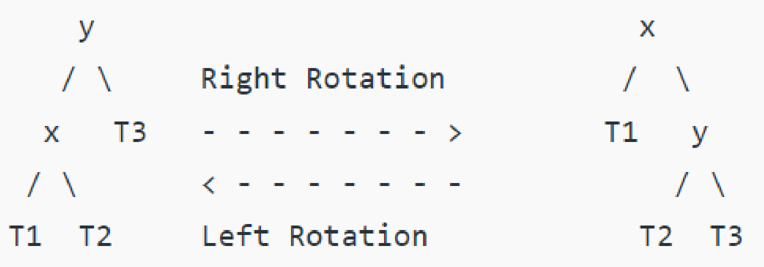
11.1 Insertion in an AVL Tree

AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees cannot be more than one for all nodes. To make sure that the given tree remains AVL after every insertion, we must augment the standard BST insert operation to perform some re-balancing.

Following are two basic operations that can be performed to balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

- Left Rotation
- Right Rotation

T1, T2 and T3 are subtrees of the tree, rooted with y (on the left side) or x (on the right side)



```
     y                               x
    / \     Right Rotation          /  \
   x   T3   - - - - - - - >        T1   y
  / \       < - - - - - - -            / \
T1   T2     Left Rotation            T2   T3
```

Keys in both of the above trees follow the following order

keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)

So BST property is not violated anywhere.

Procedure for inserting a node into an AVL tree

Let the newly inserted node be w

- Perform standard BST insert for w.
- Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the child of z that comes on the path from w to z and x be the grandchild of z that comes on the path from w to z.
- Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that need to be handled as x, y and z can be arranged in 4 ways.
- Following are the possible 4 arrangements:
    - y is the left child of z and x is the left child of y (Left Left Case)
    - y is the left child of z and x is the right child of y (Left Right Case)
    - y is the right child of z and x is the right child of y (Right Right Case)
    - y is the right child of z and x is the left child of y (Right Left Case)

```
class TreeNode {
    int val, height;
    TreeNode left, right;

    TreeNode(int d) {
        val = d;
        height = 1;
    }
}
class AVL_Tree {

    # write the code
    }

    TreeNode leftRotate(TreeNode x)
      {
      # write the code
      }
    public static void main(String[] args) {
        # write the code
      }
```

11.2 Deletion in an AVL Tree

Given an AVL tree, make sure that the given tree remains AVL after every deletion, we must augment the standard BST delete operation to perform some re-balancing. Following are two basic operations that can be performed to re-balance a BST without violating the BST property (keys(left) < key(root) < keys(right)).

1. Left Rotation
2. Right Rotation

T1, T2 and T3 are subtrees of the tree rooted with y (on left side)
or x (on right side)

```
    y                               x
   / \     Right Rotation          /   \
  x    T3  - - - - - - - >        T1    y
 / \       < - - - - - - -             / \
T1   T2    Left Rotation             T2   T3
```

Keys in both of the above trees follow the following order
   keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)
So BST property is not violated anywhere.

Procedure to delete a node from AVL tree:

Let w be the node to be deleted
1. Perform standard BST delete for w.

2. Starting from w, travel up and find the first unbalanced node. Let z be the first unbalanced node, y be the larger height child of z, and x be the larger height child of y. Note that the definitions of x and y are different from insertion here.
3. Re-balance the tree by performing appropriate rotations on the subtree rooted with z. There can be 4 possible cases that needs to be handled as x, y and z can be arranged in 4 ways. Following are the possible 4 arrangements:
    - i. y is left child of z and x is left child of y (Left Left Case)
    - ii. y is left child of z and x is right child of y (Left Right Case)
    - iii. y is right child of z and x is right child of y (Right Right Case)
    - iv. y is right child of z and x is left child of y (Right Left Case)

```
class TreeNode
       {
       int val, height;
       TreeNode left, right;

       TreeNode(int d) {
         val = d;
         height = 1;
    }
}

class AVL_Tree {

    TreeNode leftRotate(TreeNode z)
       {
       # Write code here
       }

    TreeNode rightRotate(TreeNode z)
       {
       # Write code here
       }

      TreeNode insert(TreeNode node, int key)
       {
       # Write code here
         }
```

11.3 Count Greater Nodes in AVL Tree

Given an AVL tree, calculate number of elements which are greater than given value in AVL tree.

Input: x = 5

Root of below AVL tree

```
      9
     / \
    1    10
   / \    \
  0   5    11
 /   / \
-1   2   6
```

Output: 4

Explanation: There are 4 values which are greater than 5 in AVL tree which are 6, 9, 10 and 11.

```
class TreeNode {
    int key, height, desc;
    TreeNode left, right;

    TreeNode(int d) {
        key = d;
        height = 1;
        desc = 0;
    }
}

class AVL_Tree
       {
       # Write code here
       }

    TreeNode insert(TreeNode node, int key)
       {
       # Write code here
       }
    TreeNode minValueNode(TreeNode node)
       {
       # Write code here
       }

    TreeNode deleteNode(TreeNode root, int key)
       {
        # Write code here
       }
void preOrder(TreeNode node)
       {
       # Write code here
       }

public class Main
       {
       # Write code here
       }
}
```

11.4 Minimum Number of Nodes in an AVL Tree with given Height

Given the height of an AVL tree 'h', the task is to find the minimum number of nodes the tree can have.

Input: H = 0
Output: N = 1
Only '1' node is possible if the height
of the tree is '0' which is the root node.

Input: H = 3
Output: N = 7

Recursive approach:
In an AVL tree, we have to maintain the height balance property, i.e. difference in the height of the left and the right subtrees cannot be other than -1, 0 or 1 for each node.
We will try to create a recurrence relation to find minimum number of nodes for a given height, n(h).

- For height = 0, we can only have a single node in an AVL tree, i.e. n(0) = 1
- For height = 1, we can have a minimum of two nodes in an AVL tree, i.e. n(1) = 2
- Now for any height 'h', root will have two subtrees (left and right). Out of which one has to be of height h-1 and other of h-2. [root node excluded]
- So, n(h) = 1 + n(h-1) + n(h-2) is the required recurrence relation for h>=2 [1 is added for the root node]

```
public class AVLTreeMinimumNodes {

    public static int AVLnodes(int height)
      {
      # Write code here
      }

    public static void main(String[] args) {
        int H = 3;
        System.out.println(AVLnodes(H)); // Output: 4
    }
}
```

12. Graph Traversal

12.1 Breadth First Search

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

For a given graph G, print BFS traversal from a given source vertex.

```
import java.util.*;

public class Graph {
    private Map<Integer, List<Integer>> graph;

    public Graph()
      {
        graph = new HashMap<>();
    }

    public void addEdge(int u, int v) {
        if (!graph.containsKey(u)) {
            graph.put(u, new ArrayList<>());
        }
        graph.get(u).add(v);
    }


          public void BFS(int s)
      {
      # Write code here
      }

    public static void main(String[] args)
      {
      # Write code here
      }
}
```

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

12.2 Depth First Search

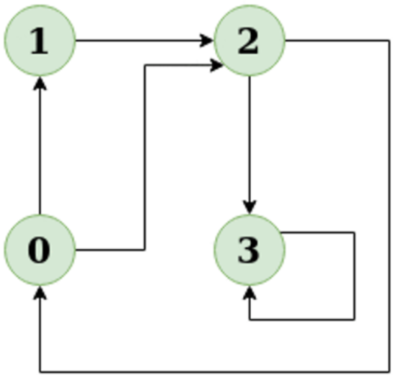
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

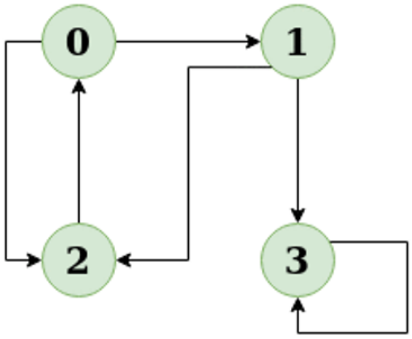
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:



```
import java.util.*;
class Graph {
    private Map<Integer, List<Integer>> graph;

    public Graph() {
        // Initialize the graph as a HashMap of ArrayLists
        graph = new HashMap<>();
    }
    public void addEdge(int u, int v)
       {
       # Write code here
      }

    public void DFS(int v) {

        DFSUtil(v, visited);
    }
    public static void main(String[] args)
       {
       # Write code here
       }
}
```

12.3 Best First Search (Informed Search)

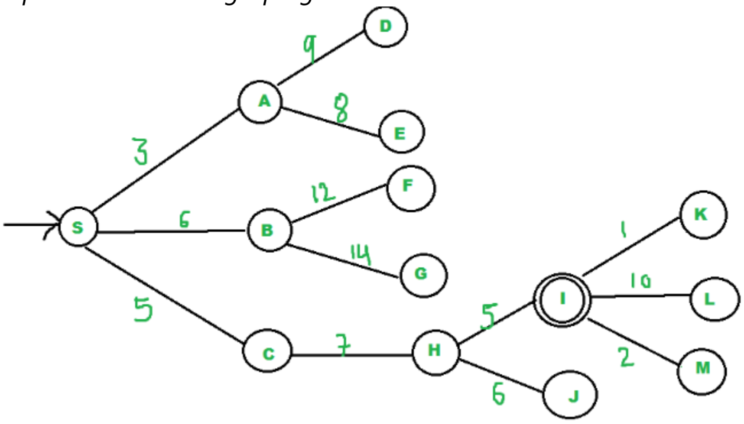
The idea of Best First Search is to use an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search.

Implementation of Best First Search:
We use a priority queue or heap to store the costs of nodes that have the lowest evaluation function value. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm:

```
Best-First-Search(Graph g, Node start)
   1) Create an empty PriorityQueue
      PriorityQueue pq;
   2) Insert "start" in pq.
      pq.insert(start)
   3) Until PriorityQueue is empty
        u = PriorityQueue.DeleteMin
        If u is the goal
          Exit
        Else
          Foreach neighbor v of u
            If v "Unvisited"
               Mark v "Visited"
               pq.insert(v)
          Mark u "Examined"
End procedure
```

Input: Consider the graph given below.



- We start from source “S” and search for goal “I” using given costs and Best First search.
- pq initially contains S
  - We remove S from pq and process unvisited neighbors of S to pq.
  - pq now contains {A, C, B} (C is put before B because C has lesser cost)
- We remove A from pq and process unvisited neighbors of A to pq.
  - pq now contains {C, B, E, D}
- We remove C from pq and process unvisited neighbors of C to pq.
  - pq now contains {B, H, E, D}
- We remove B from pq and process unvisited neighbors of B to pq.
  - pq now contains {H, E, D, F, G}

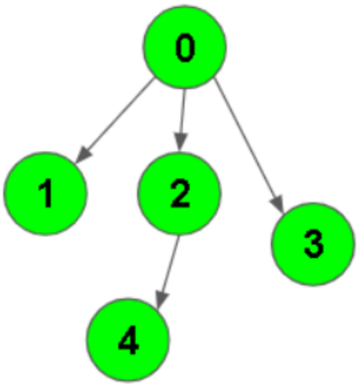
- We remove H from pq.
- Since our goal "I" is a neighbor of H, we return.

```
import java.util.*;

public class BestFirstSearch {
    static int v = 14;
    static List<List<Pair<Integer, Integer>>> graph = new ArrayList<>();
    static void addedge(int x, int y, int cost) {
        graph.get(x).add(new Pair<>(y, cost));
        graph.get(y).add(new Pair<>(x, cost));
    }

    static void best_first_search(int actual_Src, int target, int n)
       {
       # Write code here
       }

    public static void main(String[] args)
       {
       # Write code here
       }
}
```

12.4 Breadth First Traversal of a Graph

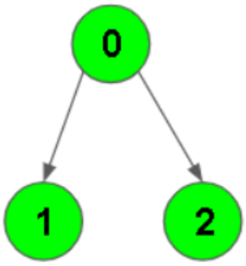
Given a directed graph. The task is to do Breadth First Traversal of this graph starting from 0.
One can move from node u to node v only if there's an edge from u to v. Find the BFS traversal of the graph starting from the 0th vertex, from left to right according to the input graph. Also, you should only take nodes directly or indirectly connected from Node 0 in consideration.

Input: Consider the graph given below where V = 5, E = 4, edges = {(0,1), (0,2), (0,3), (2,4)}



Output: 0 1 2 3 4
Explanation:
0 is connected to 1, 2, and 3.
2 is connected to 4.
So starting from 0, it will go to 1 then 2 then 3. After this 2 to 4, thus BFS will be 0 1 2 3 4.

Input: Consider the graph given below where V = 3, E = 2, edges = {(0, 1), (0, 2)}



Output: 0 1 2

Explanation:

0 is connected to 1, 2. So starting from 0, it will go to 1 then 2, thus BFS will be 0 1 2.

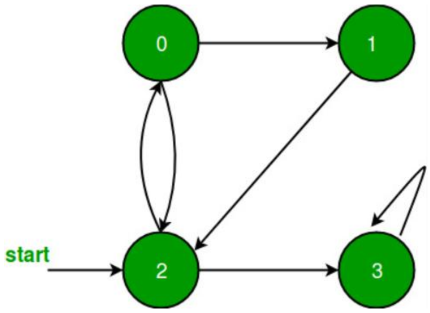
Your task is to complete the function **bfsOfGraph()** which takes the integer V denoting the number of vertices and adjacency list as input parameters and returns a list containing the BFS traversal of the graph starting from the 0th vertex from left to right.

```
import java.util.*;

class Graph {
    private int V;
    private LinkedList<Integer>[] adj;

    Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i = 0; i < v; ++i)
            adj[i] = new LinkedList();
    }

    void addEdge(int v, int w) {
        adj[v].add(w);
    }

    void BFS(int s)
      {
      # Write Code Here
        }
 }

    public static void main(String args[]) {
       # Write Code Here
    }
}
```

12.5 Depth First Search (DFS) for Disconnected Graph

Given a Disconnected Graph, the task is to implement DFS or Depth First Search Algorithm for this Disconnected Graph.

Input: Consider the graph given below.



Output: 0 1 2 3

Procedure for DFS on Disconnected Graph:

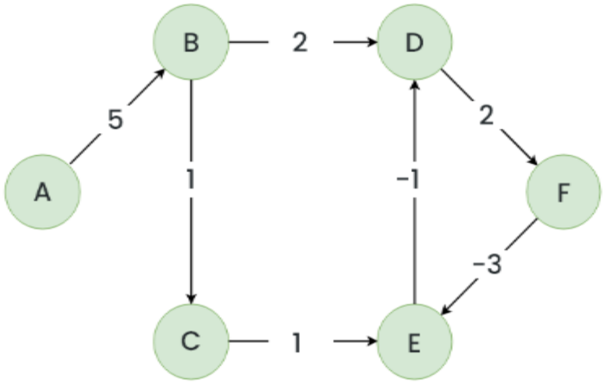
Iterate over all the vertices of the graph and for any unvisited vertex, run a DFS from that vertex.

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

// Class representing a directed graph using adjacency list representation
class Graph
       {
        # Write Code Here
       }

    public Graph()
       {
        graph = new HashMap<>();
       }

    public void addEdge(int u, int v)
       {
       # Write Code Here
       }
    private void DFSUtil(int v, boolean[] visited)
       {
       # Write Code Here
       }
    }

    public void DFS() {
        boolean[] visited = new boolean[graph.size()];
       # Write Code Here
       }

    public static void main(String[] args)
       {
       # Write Code Here
       }
}
```

Try:

1. Detect a negative cycle in a Graph (Bellman Ford): A Bellman-Ford algorithm is also guaranteed to find the shortest path in a graph, similar to Dijkstra's algorithm. Although Bellman-Ford is slower than Dijkstra's algorithm, it is capable of handling graphs with negative edge weights, which makes it more versatile. The shortest path cannot be found if there exists a negative cycle in the graph. If we continue to go around the negative cycle an infinite number of times, then the cost of the path will continue to decrease (even though the length of the path is increasing).

Consider a graph G and detect a negative cycle in the graph using Bellman Ford algorithm.



13. Minimum Spanning Tree (MST)

13.1 Kruskal's Algorithm

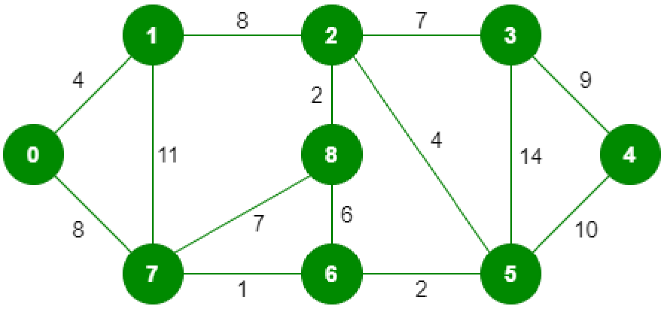
In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

MST using Kruskal's algorithm:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

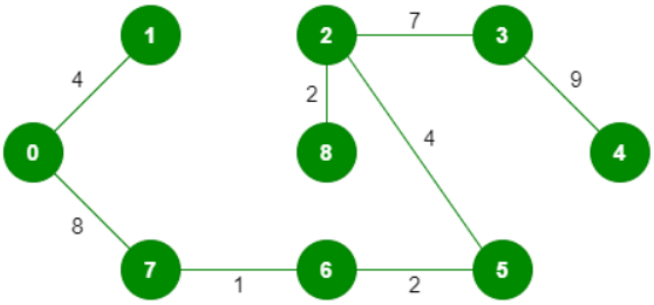
After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6

7	2	3
7	7	8
8	0	7
8	1	2
9	3	4
10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:



```
// Kruskal's algorithm to find minimum Spanning Tree of a given connected,
import java.util.*;

public class Graph {

    class Edge implements Comparable<Edge> {
        int src, dest, weight;

        // Comparator function used for sorting edges
        public int compareTo(Edge compareEdge) {
            return this.weight - compareEdge.weight;
        }
    }

    private int V; // Number of vertices
    private List<Edge> edges; // List of edges

    public Graph(int vertices) {
        this.V = vertices;
        this.edges = new ArrayList<>();
    }

    public void addEdge(int u, int v, int w)
      {
      # Write Code Here
      }
    private void union(int[] parent, int[] rank, int x, int y)
      {
      # Write Code Here
      }

    public void KruskalMST()
```

```
       {
       # Write Code Here
       }
 public static void main(String[] args) {
        Graph g = new Graph(4);
        g.addEdge(0, 1, 10);
        g.addEdge(0, 2, 6);
        g.addEdge(0, 3, 5);
        g.addEdge(1, 3, 15);
        g.addEdge(2, 3, 4);

        // Function call
        g.KruskalMST();
    }
}
```

Output: Following are the edges in the constructed MST

```
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost Spanning Tree: 19
```

13.2 Prim's Algorithm

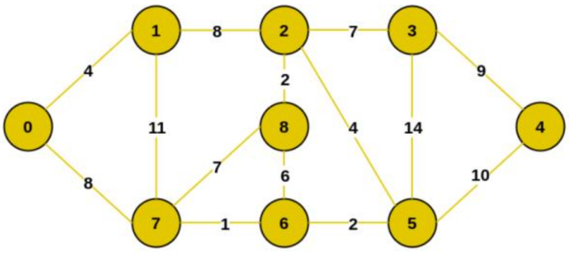
The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

Prim's Algorithm:

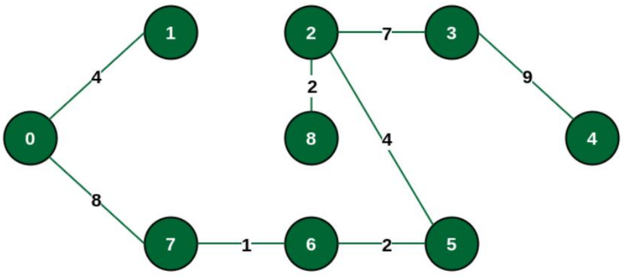
The working of Prim's algorithm can be described by using the following steps:

1. Determine an arbitrary vertex as the starting vertex of the MST.
2. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
3. Find edges connecting any tree vertex with the fringe vertices.
4. Find the minimum among these edges.
5. Add the chosen edge to the MST if it does not form any cycle.
6. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



```
import java.util.Arrays;

public class Graph {

    private int V; // Number of vertices
    private int[][] graph; // Adjacency matrix representation of graph

    public Graph(int vertices) {
        this.V = vertices;
        this.graph = new int[V][V];
    }
    public void printMST(int[] parent) {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++) {
            System.out.println(parent[i] + " - " + i + "\t" +
graph[i][parent[i]]);
        }
    }
    private int minKey(int[] key, boolean[] mstSet)
      {
               # Write Code Here
      }
    public void primMST()
      {
               # Write Code Here

      }
    public static void main(String[] args)
      {
               # Write Code Here
      }
}
```

Output:

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

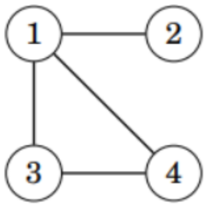
13.3 Total Number of Spanning Trees in a Graph

If a graph is a complete graph with n vertices, then total number of spanning trees is $n^{(n-2)}$ where n is the number of nodes in the graph. In complete graph, the task is equal to counting different labeled trees with n nodes for which have Cayley's formula.

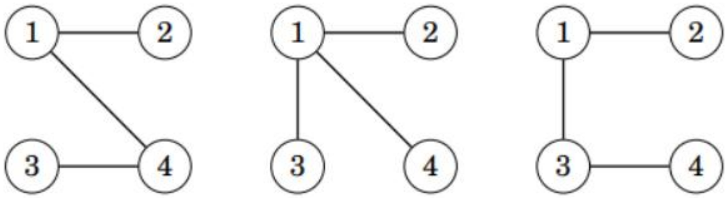
Laplacian matrix:

A Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.

Kirchhoff's theorem provides a way to calculate the number of spanning trees for a given graph as a determinant of a special matrix. Consider the following graph,



All possible spanning trees are as follows:



In order to calculate the number of spanning trees, construct a Laplacian matrix L, where L[i, i] is the degree of node i and L[i, j] = −1 if there is an edge between nodes i and j, and otherwise L[i, j] = 0.
for the above graph, The Laplacian matrix will look like this

$$L = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & -1 \\ -1 & 0 & -1 & 2 \end{bmatrix}$$

The number of spanning trees equals the determinant of a matrix.
The Determinant of a matrix that can be obtained when we remove any row and any column from L.
For example, if we remove the first row and column, the result will be,

$$\det(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}) = 3.$$

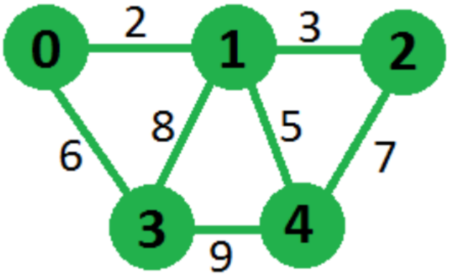
The determinant is always the same, regardless of which row and column we remove from L.

```
import java.util.Arrays;

public class NumberOfSpanningTrees {

    static final int MAX = 100;
    static final int MOD = 1000000007;
    void multiply(long[][] A, long[][] B, long[][] C, int size) {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                C[i][j] = 0;
                for (int k = 0; k < size; k++) {
                    C[i][j] = (C[i][j] + A[i][k] * B[k][j]) % MOD;
                }
            }
        }
    }
 void power(long[][] A, int N, long[][] result, int size)
       {
       # Write Code Here
       }

 long numOfSpanningTree(int[][] graph, int V)
       {
       # Write Code Here
       }

    public static void main(String[] args) {
        int V = 4; // Number of vertices in graph
        int E = 5; // Number of edges in graph
        int[][] graph = { { 0, 1, 1, 1 }, { 1, 0, 1, 1 }, { 1, 1, 0, 1 }, { 1, 1,
1, 0 } };

        NumberOfSpanningTrees obj = new NumberOfSpanningTrees();
        System.out.println(obj.numOfSpanningTree(graph, V));
    }
}
```

13.4 Minimum Product Spanning Tree

A minimum product spanning tree for a weighted, connected, and undirected graph is a spanning tree with a weight product less than or equal to the weight product of every other spanning tree. The weight product of a spanning tree is the product of weights corresponding to each edge of the spanning tree. All weights of the given graph will be positive for simplicity.

Input:



Output: Minimum Product that we can obtain is 180 for above graph by choosing edges 0-1, 1-2, 0-3 and 1-4

This problem can be solved using standard minimum spanning tree algorithms like Kruskal and prim's algorithm, but we need to modify our graph to use these algorithms. Minimum spanning tree algorithms tries to minimize the total sum of weights, here we need to minimize the total product of weights. We can use the property of logarithms to overcome this problem.
log(w1* w2 * w3 * .... * wN) = log(w1) + log(w2) + log(w3) ..... + log(wN)
We can replace each weight of the graph by its log value, then we apply any minimum spanning tree algorithm which will try to minimize the sum of log(wi) which in turn minimizes the weight product.

```
import java.util.Arrays;

public class MinimumProductMST {
    // Number of vertices in the graph
    static final int V = 5;

    // A utility function to find the vertex with minimum key value, from the set
of
    // vertices not yet included in MST
    int minKey(int key[], boolean mstSet[]) {
        int min = Integer.MAX_VALUE, min_index = -1;

        for (int v = 0; v < V; v++) {
            if (mstSet[v] == false && key[v] < min) {
                min = key[v];
                min_index = v;
            }
        }

        return min_index;
    }

    void printMST(int parent[], int n, int graph[][])
      {
      # Write Code Here
      }
    void primMST(int inputGraph[][], int logGraph[][])
      {
      # Write Code Here
      }
    void minimumProductMST(int graph[][])
      {
      # Write Code Here
      }
    public static void main(String[] args)
      {
      # Write Code Here
    }
}
```

13.5 Reverse Delete Algorithm for Minimum Spanning Tree

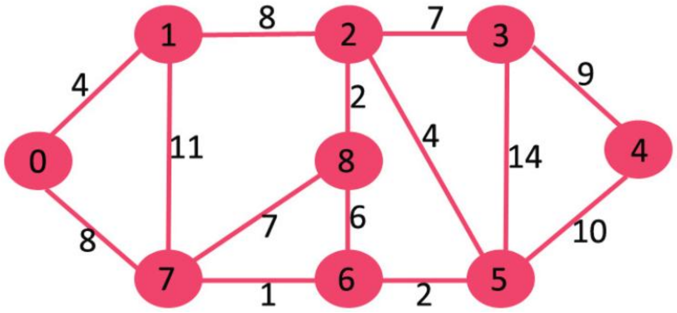
In Reverse Delete algorithm, we sort all edges in decreasing order of their weights. After sorting, we one by one pick edges in decreasing order. We include current picked edge if excluding current edge causes disconnection in current graph. The main idea is delete edge if its deletion does not lead to disconnection of graph.

Algorithm:

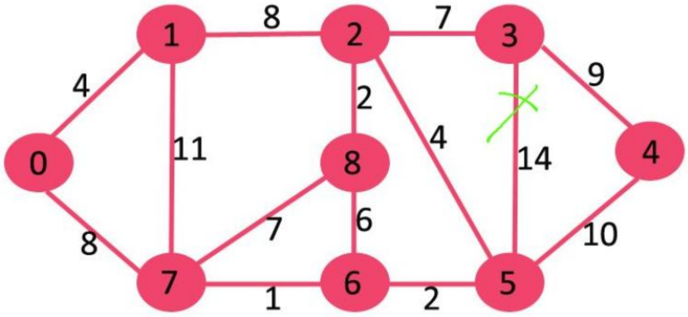
1. Sort all edges of graph in non-increasing order of edge weights.
2. Initialize MST as original graph and remove extra edges using step 3.

3. Pick highest weight edge from remaining edges and check if deleting the edge disconnects the graph or not.
   If disconnects, then we don't delete the edge.
   Else we delete the edge and continue.

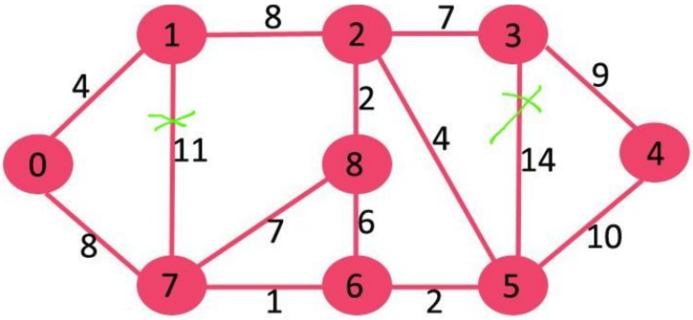
Input: Consider the graph below



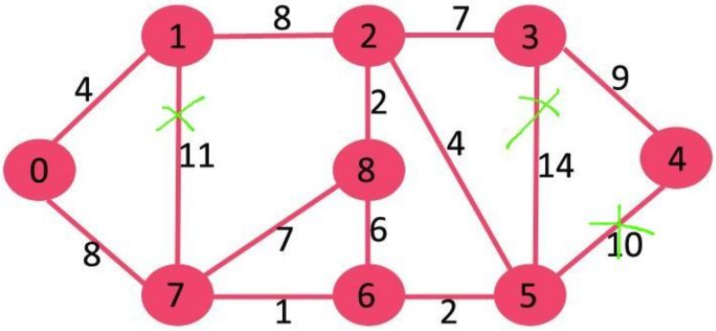
If we delete highest weight edge of weight 14, graph doesn't become disconnected, so we remove it.



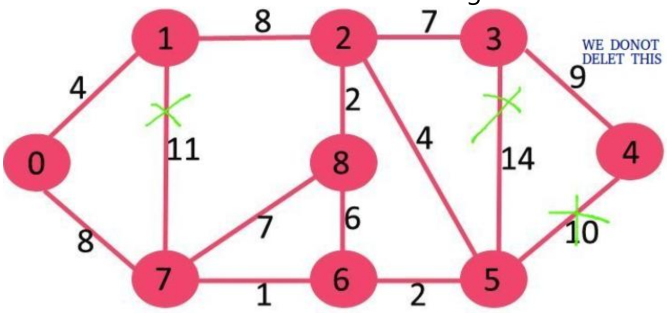
Next we delete 11 as deleting it doesn't disconnect the graph.



Next we delete 10 as deleting it doesn't disconnect the graph.



Next is 9. We cannot delete 9 as deleting it causes disconnection.



We continue this way and following edges remain in final MST.

Edges in MST

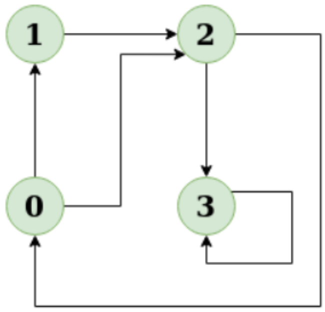
(3, 4)
(0, 7)
(2, 3)
(2, 5)
(0, 1)
(5, 6)
(2, 8)
(6, 7)

```
import java.util.ArrayList;
import java.util.Collections;

// Edge class to represent edges in the graph
class Edge {
    int src, dest, weight;

    Edge(int src, int dest, int weight) {
        this.src = src;
        this.dest = dest;
        this.weight = weight;
    }
}
class Graph
      {
      # Write Code Here
      }

    // Function to add an edge to the graph
    void addEdge(int u, int v, int w) {
      # Write Code Here
      }
    void dfs(int v, boolean[] visited)
      {
       # Write Code Here
      }

    // Function to check if the graph is connected
    boolean connected()
      {
      # Write Code Here
      }
void reverseDeleteMST()
      {
       # Write Code Here
      }
}

public class ReverseDeleteMST {
    public static void main(String[] args)
      {
      # Write Code Here
      }
}
```

Try:

1. **Detect Cycle in a Directed Graph:** Given the root of a Directed graph, The task is to check whether the graph contains a cycle or not.

Input: N = 4, E = 6



Output: Yes
Explanation: The diagram clearly shows a cycle 0 -> 2 -> 0

14. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
• CodeChef	20	200
• Leetcode	20	200
• GeeksforGeeks	20	200
• SPOJ	5	50
• InterviewBit	10	1000
• Hackerrank	25	250
• Codeforces	10	100
• BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certification:

1. HackerRank - Problem Solving Skills Certification (Basic and Intermediate)

2. GeeksforGeeks – Data Structures and Algorithms Certification
3. CodeChef - Learn Data Structures and Algorithms Certification
4. Interviewbit – DSA pro / Python pro
5. Edx – Data Structures and Algorithms
5. NPTEL – Programming, Data Structures and Algorithms
6. NPTEL – Introduction to Data Structures and Algorithms
7. NPTEL – Data Structures and Algorithms
8. NPTEL – Programming and Data Structure

V. TEXT BOOKS:

1. Rance D. Necaise, "Data Structures and Algorithms using Python", Wiley Student Edition.
2. Benjamin Baka, David Julian, "Python Data Structures and Algorithms", Packt Publishers, 2017.

VI. REFERENCE BOOKS:

1. S. Lipschutz, "Data Structures", Tata McGraw Hill Education, 1st Edition, 2008.
2. D. Samanta, "Classic Data Structures", PHI Learning, 2nd Edition, 2004.

VII. ELECTRONICS RESOURCES:

1. https://www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm
2. https://www.codechef.com/certification/data-structures-and-algorithms/prepare
3. https://www.cs.auckland.ac.nz/software/AlgAnim/dsToC.html
4. https://online-learning.harvard.edu/course/data-structures-and-algorithms

VIII. MATERIALS ONLINE

5. Syllabus
6. Lab manual

COURSE CONTENT

ENGLISH LANGUAGE AND COMMUNICATION SKILLS LABORATORY								
I Semester: AE / ME / CE / ECE / EEE / CSE (AI &ML) / CSE (DS) II Semester: CSE / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AHSE07	Foundation	L	T	P	C	CIA	SEE	Total
		-	-	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 45				Total Classes: 45		
Prerequisite: Nil								

I. COURSE OVERVIEW:

This laboratory course is designed to introduce students to create a wide exposure on language learning techniques of the basic elements of listening skills, speaking skills, reading skills and writing skills. In this laboratory, students are trained in communicative English language skills, phonetics, word accent, word stress, rhythm, intonation, oral presentations and extempore speeches. Students are also taught in terms of seminars, group-discussions, presenting techniques of writing, participating in role plays, telephonic etiquettes, asking and giving directions, information transfer, debates, description of persons, places and objects etc. The laboratory encourages students to work in a group, engage in peer-reviews and inculcate team spirit through various exercises on grammar, vocabulary, and pronunciation games etc. Students will make use of all these language skills in academic, professional and real time situations.

II. COURSES OBJECTIVES

The students will try to learn:

I The computer-assisted multi-media instructions to make possible individualized and independent language learning.

II The critical aspect of speaking and reading for interpreting in-depth meaning of the sentences.

III The language techniques for social interactions such as public speaking, group discussions and interviews.

IV English speech sounds, word accent, intonation and rhythm patterns for effective pronunciation.

III. COURSE OUTCOMES:

At the end of the course, students will be able to:

CO1	Discuss the prime necessities of listening skills for improving pronunciation in academic and non-academic purposes.
CO2	Summarize the significance of speaking skills by using phonetics knowledge and intonation patterns.
CO3	Express the usage of strong forms and weak forms in the connected speech.
CO4	Explain how writing skills fulfil the academic and non-academic requirements of various written communicative functions.
CO5	Generalize the activities of Interactive Communication Skills to overcome the day-to-day challenges.
CO6	Classify the roles of collaboration, risk-taking, multi-disciplinary awareness, and the imagination in achieving creative responses to problems.

Dos

1. Turn up in a neat and formal dress code regularly and maintain punctuality.
2. Bring observation books and worksheets for every laboratory session without fail.
3. Keep lab record book up to date.
4. Students must adhere to the acceptable use of ICT resources policy.
5. CD ROM ‘s, USB and other multimedia equipment are for college use only.
6. Replace headsets onto the monitor and rearrange your chairs back into their position as you leave laboratory.
7. Get your lab worksheets evaluated and upload online within the stipulated time for online evaluation by the faculty concerned.
8. Conduct yourself at the best to be a good learner.

Don’ts

1. Do not use the on/off switch to reboot the system.
2. Do not breach copyright regulations.
3. The not install or download any software or modify or delete any system files on any laboratory computer.
4. Do not read or modify other users’ files.
5. Do not damage, remove, or disconnect any labels, parts, cables or equipment.
6. The not make undue noise in the laboratories. Be considerate of the other lab users- this is study area.
7. No food and the beverages are allowed into computer laboratories.

IV. COURSE CONTENT

Exercises for professional communication laboratory

Exercises –1 CALL LAB: Speech Sounds with Active Listening.
ICS LAB: introducing self and introducing others and feedback.

a. Common mispronunciations.
b. Errors committed in self-introduction and introducing others.

Exercises –2 CALL LAB: Listening to Distinguish Speech Sounds (minimal pairs) – Testing Exercises.
ICS LAB: Ice Breaking Activity.

a. Difficulty in familiarizing with the sounds of English language, errors in using different kinds of sounds, vowels and consonants.
b. supports a positive social climate; decreases in off task behaviors; improved social skills; improves student enjoyment; and raises participation levels.

Exercises –3 CALL LAB: Listening for General Information Followed by Multiple Questions.
ICS LAB: Role Play Activity.

a. Listening actively to understand the information to respond.
b. Take on different roles and act out situations like ordering food in a restaurant, asking for directions, or even having a phone conversation.

Exercises –4 CALL LAB: Listening Comprehension Activity.
ICS LAB: Social Etiquettes.

a. Enhancing their language skills, academic performance, and social interactions.
b. Positive social interactions, enhancing communication skills, and contributing to overall personal and academic success.

Exercises –5 CALL LAB: Neutralization of Mother Tongue Influence (MTI).
ICS LAB: Describing Objects, Situations, Places, People and Events.

a. Influence of Mother tongue in spoken communication.
b. Strengthen the art of writing and spoken language.

Exercises – 6 CALL LAB: Techniques for Effective Listening.
ICS LAB: Story Telling.

a. actively engaging with the speaker to understand their message fully.
b. enhancing their language skills, creativity, and overall learning experience.

Exercises –7 CALL LAB: Identifying the Literal and Implied Meaning.
ICS LAB: Non-Verbal Communication

a. Listening for evaluation – Write summary – Listening for evaluation – Listening comprehension exercises.
b. Attention of non-verbal ques.

Exercises – 8 CALL LAB: Structure of syllables.
ICS LAB: JAM Sessions using public address system.

a. Practicing consonant clusters
b. Practicing different methods of dividing the syllables
c. Participating in just a minute session

Exercises –9 CALL LAB: Past tense and plural markers.
ICS LAB: Oral Presentations.

a. Addition of suffixes to verbs.
b. Confidence and fluency in delivering different oral presentations.

Exercises –10 CALL LAB: Minimal pairs.
ICS LAB: Debates.

a. Difficulties in understanding and remembering various homonyms, homophones and homographs.
b. Problems in understanding the difference between debates and discussions, participating and contributing.

Exercises –11 CALL LAB: Intonation.
ICS LAB: Group discussion.

a. Inability in focused listening, understanding the accent, vocabulary and discourse markers in connected speech.
b. Lack of confidence in participating and contributing to Group discussions.

Exercises –12 CALL LAB: Demonstration on how to write leaflets, messages and notices.
ICS LAB: Techniques and methods to write summaries and reviews of videos.

a. Inadequacy and inappropriacy in writing leaflets, messages and notices.
b. Lack of proficiency in writing summaries and reviews of videos.

Exercises –13 CALL LAB: Pronunciation practice.
ICS LAB: Information transfer.

a. Influence of mother tongue in using English language.
b. Problems in interpreting data from diagram to text and text to diagram.

Exercises –14 CALL LAB; Open Ended Experiments-Phonetics Practice.
ICS LAB: Picture Extempore.

a. Persistent problems in identifying the phonetic symbols, remembering and using them.
b. Execution while describing picture.

Exercises –15 CALL LAB: Open Ended experiments-Text to Speech.
ICS LAB: Writing slogan related to the image.

a. Difficulties in writing text to Speech.
b. Lack of fluency in writing slogans related to the images.

V. TEXT BOOKS:

1. Professional Communication laboratory manual

VI. REFERENCE BOOK

1. Meenakshi Raman, Sangeetha Sharma, *Technical Communication Principles and Practices*, Oxford University Press, New Delhi, 3rd Edition, 2015.
2. Rhirdion, Daniel, *Technical Communication*, Cengage Learning, New Delhi, 1st Edition, 2009.

VII. ELECTRONICS RESOURCES

1. Cambridge online pronunciation dictionary https://dictionary.cambridge.org/
2. Fluentu website https://www.fluentu.com/
3. Repeat after us https://brycs.org/clearinghouse/3018/
4. Language lab https://brycs.org/clearinghouse/3018/
5. Oxford online videos

VIII. MATERIALS ONLINE

1. Course template
2. Lab manual

COURSE CONTENT

<table>
<tr><th colspan="9">PROGRAMMING FOR PROBLEM SOLVING LABORATORY</th></tr>
<tr><td colspan="9">II Semester: AE / ME / CE / ECE / EEE / CSE / IT / CSE (AI&ML) / CSE (DS)</td></tr>
<tr><td rowspan="2">Course Code</td><td rowspan="2">Category</td><td colspan="3">Hours / Week</td><td>Credits</td><td colspan="3">Maximum Marks</td></tr>
<tr><td>L</td><td>T</td><td>P</td><td>C</td><td>CIA</td><td>SEE</td><td>Total</td></tr>
<tr><td>ACSE07</td><td>Foundation</td><td>0</td><td>0</td><td>2</td><td>1</td><td>40</td><td>60</td><td>100</td></tr>
<tr><td>Contact Classes: Nil</td><td>Tutorial Classes: Nil</td><td colspan="4">Practical Classes: 45</td><td colspan="3">Total Classes: 45</td></tr>
<tr><td colspan="9">Prerequisite: There is no prerequisite to take this course</td></tr>
</table>

I. COURSE OVERVIEW:

The course is designed with the fundamental programming skills and problem-solving strategies necessary to tackle a wide range of computational challenges. Through hands-on programming exercises and projects, students will learn how to write code, analyze problems and develop solutions using various programming languages and tools. The course will cover fundamental programming concepts and gradually progress to more advanced topics.

II. COURSES OBJECTIVES:

The students will try to learn

I The fundamental programming constructs and use of collection data types in Python.

II The ability to develop programs using object-oriented features.

III Basic data structures and algorithms for efficient problem-solving.

IV Principles of graph theory and be able to apply their knowledge to a wide range of practical problems across various disciplines.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO1	Adapt programming concepts, syntax, and data structures through hands on coding exercises
CO2	Develop the ability to solve a variety of programming problems and algorithms using python
CO3	Implement complex and custom data structures to solve real-world problems.
CO4	Demonstrate proficiency in implementing graph algorithms to solve variety of problems and scenarios.
CO5	Develop critical thinking skills to solve the various real-world applications using graph theory.
CO6	Learn the importance of numerical methods and apply them to tackle a wide range of computational problems.

IV. COURSE CONTENT:

EXERCISES FOR PROGRAMMING FOR PROBLEM SOLVING LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

1.1 Two Sum

Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order.

Input: nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

Explanation: Because nums[0] + nums[1] == 9, so return [0, 1].

Input: nums = [3, 2, 4], target = 6

Output: [1, 2]

Input: nums = [3, 3], target = 6

Output: [0, 1]

Hints:

```
def twoSum(self, nums: List[int], target: int) -> List[int]:
     a=[]
     # Write code here
     …

     return a
```

1.2 Contains Duplicate

Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Input: nums = [1, 2, 3, 1]

Output: true

Input: nums = [1, 2, 3, 4]

Output: false

Input: nums = [1, 1, 1, 3, 3, 4, 3, 2, 4, 2]

Output: true

Hints:

```
def containsDuplicate(self, nums):
        a = set() # set can have only distinct elements
        # Write code here
        …
        return False
```

1.3 Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:
I can be placed before V (5) and X (10) to make 4 and 9.
X can be placed before L (50) and C (100) to make 40 and 90.
C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Input: s = "III"

Output: 3

Input: s = "LVIII"

Output: 58

Hints:

```
def romanToInt(self, s: str) -> int:
        # Write code here
        …

        return number
```

1.4 Plus One

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's. Increment the large integer by one and return the resulting array of digits.

Input: digits = [1, 2, 3]

Output: [1, 2, 4]

Explanation: The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1, 2, 4].

Hints:

```
def plusOne(self, digits: List[int]) -> List[int]:
        n = len(digits)
        # Write code here
         …

        return digits
```

1.5 Majority Element

Given an array nums of size n, return the majority element. The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

Input: nums = [3, 2, 3]

Output: 3

Input: nums = [2, 2, 1, 1, 1, 2, 2]

Output: 2

Hints:

```
def majorityElement(self, nums):
        # write code here
         …
```

1.6 Richest Customer Wealth

You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the i^{th} customer has in the j^{th} bank. Return the wealth that the richest customer has. A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth.

Input: accounts = [[1, 2, 3], [3,2,1]]

Output: 6

Explanation:

1st customer has wealth = 1 + 2 + 3 = 6

2nd customer has wealth = 3 + 2 + 1 = 6

Both customers are considered the richest with a wealth of 6 each, so return 6.

Input: accounts = [[1, 5], [7,3],[3,5]]

Output: 10

Explanation:

1st customer has wealth = 6

2nd customer has wealth = 10

3rd customer has wealth = 8

The 2nd customer is the richest with a wealth of 10.

Input: accounts = [[2,8,7],[7,1,3],[1,9,5]]

Output: 17

Hints:

```
def maximumWealth(self, accounts: List[List[int]]) -> int:
        # write code here
         …
```

1.7 Fizz Buzz

Given an integer n, return a string array answer (1-indexed) where:

answer[i] == "FizzBuzz" if i is divisible by 3 and 5.

answer[i] == "Fizz" if i is divisible by 3.

answer[i] == "Buzz" if i is divisible by 5.

answer[i] == i (as a string) if none of the above conditions are true.

Input: n = 3

Output: ["1","2","Fizz"]

Input: n = 5

Output: ["1","2","Fizz","4","Buzz"]

Input: n = 15

Output: ["1","2","Fizz","4","Buzz","Fizz","7","8","Fizz","Buzz","11","Fizz","13","14","FizzBuzz"]

Hints:

```
def fizzBuzz(self, n: int) -> List[str]:
        # write code here
         …
```

1.8 Number of Steps to Reduce a Number to Zero

Given an integer num, return the number of steps to reduce it to zero. In one step, if the current number is even, you have to divide it by 2, otherwise, you have to subtract 1 from it.

Input: num = 14

Output: 6

Explanation:

- 14 is even; divide by 2 and obtain 7.
- 7 is odd; subtract 1 and obtain 6.
- 6 is even; divide by 2 and obtain 3.
- 3 is odd; subtract 1 and obtain 2.
- 2 is even; divide by 2 and obtain 1.
- 1 is odd; subtract 1 and obtain 0.

Input: num = 8

Output: 4

Explanation:

- 8 is even; divide by 2 and obtain 4.
- 4 is even; divide by 2 and obtain 2.
- 2 is even; divide by 2 and obtain 1.

- 1 is odd; subtract 1 and obtain 0.

Input: num = 123

Output: 12

Hints:

```
def numberOfSteps(self, n: int) -> int:
        # write code here
         …
```

1.9 Running Sum of 1D Array

Given an array nums. We define a running sum of an array as runningSum[i] = sum (nums[0]…nums[i]).

Return the running sum of nums.

Input: nums = [1, 2, 3, 4]

Output: [1, 3, 6, 10]

Explanation: Running sum is obtained as follows: [1, 1+2, 1+2+3, 1+2+3+4].

Input: nums = [1, 1, 1, 1, 1]

Output: [1, 2, 3, 4, 5]

Explanation: Running sum is obtained as follows: [1, 1+1, 1+1+1, 1+1+1+1, 1+1+1+1+1].

Input: nums = [3, 1, 2, 10, 1]

Output: [3, 4, 6, 16, 17]

Hints:

```
def runningSum(self, nums: List[int]) -> List[int]:
      # write code here
        …
      return answer
```

1.10 Remove Element

Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val. Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:

- Change the array nums such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.
- Return k.

Input: nums = [3, 2, 2, 3], val = 3

Output: 2, nums = [2, 2, _, _]

Explanation: Your function should return k = 2, with the first two elements of nums being 2.

It does not matter what you leave beyond the returned k (hence they are underscores).

Input: nums = [0,1,2,2,3,0,4,2], val = 2

Output: 5, nums = [0,1,4,0,3,_,_,_]

Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3, and 4.

Note that the five elements can be returned in any order.

It does not matter what you leave beyond the returned k (hence they are underscores).

Hints:

```
def removeElement(self, nums: List[int], val: int) -> int:
      # write code here
        …
      return len(nums)
```

2. Matrix Operations

2.1 Add Two Matrices

Given two matrices X and Y, the task is to compute the sum of two matrices and then print it in Python.

Input:

X= [[1, 2, 3],
 [4, 5, 6],
 [7, 8, 9]]

Y = [[9, 8, 7],
 [6, 5, 4],
 [3, 2, 1]]

Output:

Result = [[10, 10, 10],
 [10, 10, 10],
 [10, 10, 10]]

Hints:

```
# Program to add two matrices using nested loop

X = [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]]

Y = [[9,8,7],
    [6,5,4],
    [3,2,1]]


result = [[0,0,0],
         [0,0,0],
         [0,0,0]]

# iterate through rows
for i in range(len(X)):
    # write code here
    …

for r in result:
    print(r)
```

TRY

1. Take input as X = [[10, 20, 30],[41, 52, 63], [47, 58, 69]] Y = [[19,18,17],[66,35,49], [13,21,11]] and verify the results.

2.2 Multiply Two Matrices

Given two matrices X and Y, the task is to compute the multiplication of two matrices and then print it.

Input:

X= [[1, 7, 3],
  [3, 5, 6],
  [6, 8, 9]]

Y = [[1, 1, 1, 2],
  [6, 7, 3, 0],
  [4, 5, 9, 1]]

Output:

Result = [[55, 65, 49, 5],
  [57, 68, 72, 12],
  [90, 107, 111, 21]]

Hints:

```
# Program to multiply two matrices using list comprehension

# take a 3x3 matrix
A = [[12, 7, 3],
    [4, 5, 6],
    [7, 8, 9]]

# take a 3x4 matrix
B = [[5, 8, 1, 2],
    [6, 7, 3, 0],
    [4, 5, 9, 1]]

# result will be 3x4
# write code here
    …
for r in result:
    print(r)
```

TRY

1. Take input as X = [[11, 0, 30],[-41, -2, 63], [41, -5, -9]] Y = [[19,-48,17],[-6,35,19], [13,1,-9]] and verify the results.

2.3 Transpose of a Matrix

A matrix can be implemented using a nested list. Each element is treated as a row of the matrix. Find the transpose of a matrix in multiple ways.

Input: [[1, 2], [3, 4], [5, 6]]
Output: [[1, 3, 5], [2, 4, 6]]

Explanation: Suppose we are given a matrix

```
[[1, 2],
 [3, 4],
 [5, 6]]
```

Then the transpose of the given matrix will be,

```
[[1, 3, 5],
 [2, 4, 6]]
```

Hints:

```
# Program to multiply two matrices using list comprehension

# take a 3x3 matrix
A = [[12, 7, 3],
    [4, 5, 6],
    [7, 8, 9]]

# result will be 3x4
# write code here
    …
for r in result:
    print(r)
```

TRY

1. Take input as X = [[11, 0, 30],[-41, -2, 63], [41, -5, -9]] and verify the results.

2.4 Matrix Product

Matrix product problem we can solve using list comprehension as a potential shorthand to the conventional loops. Iterate and find the product of the nested list and at the end return the cumulative product using function.

Input: The original list: [[1, 4, 5], [7, 3], [4], [46, 7, 3]]
Output: The total element product in lists is: 1622880

Hints:

```
# Matrix Product using list comprehension + loop

def prod(val):
    # write code here
    …

# initializing list
test_list = [[1, 4, 5], [7, 3], [4], [46, 7, 3]]
```

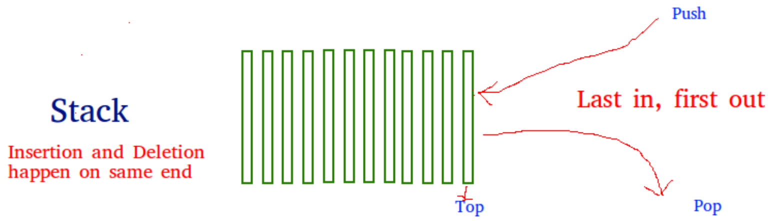
TRY

1. Take input list: [[1, 4, 5], [7, 3], [4], [46, 7, 3]] and verify the result.

3. Stack

3.1 Stack implementation using List

A stack is a linear data structure that stores items in a Last-In/First-Out (LIFO) or First-In/Last-Out (FILO) manner. In stack, a new element is added at one end and an element is removed from that end only. The insert and delete operations are often called push and pop.



The functions associated with stack are:

- **empty()** – Returns whether the stack is empty
- **size()** – Returns the size of the stack
- **top() / peek()** – Returns a reference to the topmost element of the stack
- **push(a)** – Inserts the element 'a' at the top of the stack
- **pop()** – Deletes the topmost element of the stack

Hints:

```
# Stack implementation using list
top=0
mymax=5
def createStack():
    stack=[]
    return stack
def isEmpty(stack):
    # write code here
    …
def Push(stack,item):
    # write code here
    …
def Pop(stack):
    # write code here
    …
# create a stack object
stack = createStack()
while True:
    print("1.Push")
    print("2.Pop")
    print("3.Display")
    print("4.Quit")
    # write code here
    …
```

TRY

1. Take input operations as [PUSH(A),PUSH(B),PUSH(C),POP,POP,POP,POP,PUSH(D)] and verify the result.

2. Take input operations as [POP, POP, PUSH (A), PUSH (B), POP, and PUSH(C)] and verify the result.

3.2 Balanced Parenthesis Checking

Given an expression string, write a python program to find whether a given string has balanced parentheses or not.

Input: {[]{()}}

Output: Balanced

Input: [{}{}(]

Output: Unbalanced

Using stack One approach to check balanced parentheses is to use stack. Each time, when an open parentheses is encountered push it in the stack, and when closed parenthesis is encountered, match it with the top of stack and pop it. If stack is empty at the end, return Balanced otherwise, Unbalanced.

Hints:

```
# Check for balanced parentheses in an expression
open_list = ["[","{","("]
close_list = ["]","}",")"]

# Function to check parentheses
def check(myStr):
    # write code here
    …
```

TRY

1. Take input as {[]{()}[][]} and verify the result.

2. Take input as {[]{()}[]{}} and verify the result.

3.3 Evaluation of Postfix Expression

Given a postfix expression, the task is to evaluate the postfix expression. Postfix expression: The expression of the form "a b operator" (ab+) i.e., when a pair of operands is followed by an operator.

Input: str = "2 3 1 * + 9 -"

Output: -4

Explanation: If the expression is converted into an infix expression, it will be 2 + (3 * 1) – 9 = 5 – 9 = -4.

Input: str = "100 200 + 2 / 5 * 7 +"

Output: 757

Procedure for evaluation postfix expression using stack:

- Create a stack to store operands (or values).
- Scan the given expression from left to right and do the following for every scanned element.
    - If the element is a number, push it into the stack.
    - If the element is an operator, pop operands for the operator from the stack. Evaluate the operator and push the result back to the stack.
- When the expression is ended, the number in the stack is the final answer.

Hints:

```
# Evaluate value of a postfix expression

# Class to convert the expression
class Evaluate:

    # Constructor to initialize the class variables
    def __init__(self, capacity):
        self.top = -1
        self.capacity = capacity

        # This array is used a stack
        self.array = []

    # Check if the stack is empty
    def isEmpty(self):
        # write code here
        …
    def peek(self):
        # write code here
        …
    def pop(self):
        # write code here
        …
    def push(self, op):
        # write code here
        …
    def evaluatePostfix(self, exp):
        # write code here
        …

# Driver code
exp = "231*+9-"
obj = Evaluate(len(exp))

# Function call
print("postfix evaluation: %d" % (obj.evaluatePostfix(exp)))
```

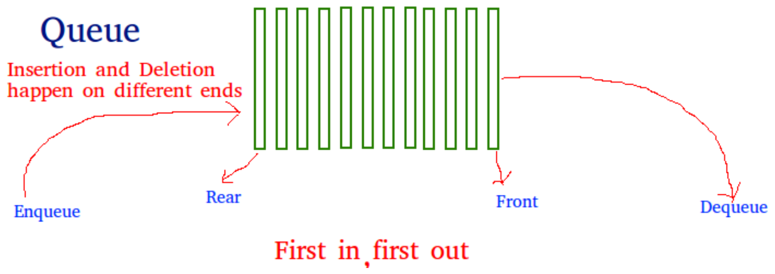
TRY

1. Take input str = "A B + C* D / E +" and verify the result.
2. Take input str = "XYZ- + W+ R / S -" and verify the result.

4. Queue

4.1 Linear Queue using List

Linear queue is a linear data structure that stores items in First in First out (FIFO) manner. With a queue the least recently added item is removed first. A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.



Hints:

```
# Static implementation of linear queue
front=0
rear=0
mymax=5
def createQueue():
    queue=[]   #empty list
    return queue

def isEmpty(queue):
    # write code here
    …
def enqueue(queue,item):  # insert an element into the queue
    # write code here
    …
def dequeue(queue):  #remove an element from the queue
    # write code here
    …
# Driver code
queue = createQueue()
while True:
    print("1.Enqueue")
    print("2.Dequeue")
    print("3.Display")
    print("4.Quit")
    # write code here
    …
```

TRY

1. Take input operations as [ENQUEUE(A),DEQUEUE(),ENQUEUE(B),DEQUEUE(),ENQUEUE(C),DEQUEUE(),] and verify the result.

2. Take input operations as [ENQUEUE(A), ENQUEUE(B),DEQUEUE(),ENQUEUE(C), DEQUEUE(),ENQUEUE(D), DEQUEUE(), ENQUEUE(C),DEQUEUE(),] and verify the result.

4.2 Stack using Queues

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (push, top, pop, and empty).

- void push(int x) Pushes element x to the top of the stack.
- int pop() Removes the element on the top of the stack and returns it.
- int top() Returns the element on the top of the stack.
- boolean empty() Returns true if the stack is empty, false otherwise.

Input:

["MyStack", "push", "push", "top", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 2, 2, false]

Hints:

```
class MyStack:

    def __init__(self):
       # write code here
       …
    def push(self, x: int) -> None:
       # write code here
       …
    def pop(self) -> int:
       # write code here
       …
    def top(self) -> int:
       # write code here
       …
    def empty(self) -> bool:
       # write code here
       …
# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

4.3 Implement Queue using Stacks

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (push, peek, pop, and empty).

- void push(int x) Pushes element x to the back of the queue.
- int pop() Removes the element from the front of the queue and returns it.
- int peek() Returns the element at the front of the queue.
- boolean empty() Returns true if the queue is empty, false otherwise.

Input:

["MyQueue", "push", "push", "peek", "pop", "empty"]

[[], [1], [2], [], [], []]

Output:

[null, null, null, 1, 1, false]

Hints:

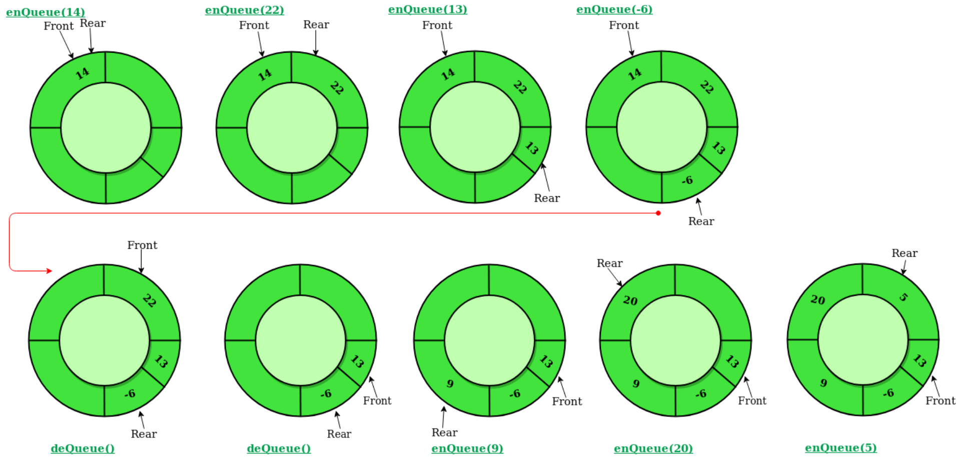
```
class MyQueue:

    def __init__(self):
       # write code here
       …

    def push(self, x: int) -> None:
       # write code here
       …

    def pop(self) -> int:
       # write code here
       …

    def peek(self) -> int:
       # write code here
       …


    def empty(self) -> bool:
       # write code here
       …
# Your MyQueue object will be instantiated and called as such:
# obj = MyQueue()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.peek()
# param_4 = obj.empty()
```

4.4 Circular Queue

A Circular Queue is an extended version of a normal queue where the last element of the queue is connected to the first element of the queue forming a circle. The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer'.

Operations on Circular Queue:

- **Front:** Get the front item from the queue.
- **Rear:** Get the last item from the queue.
- **enQueue(value)** This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at the rear position.
    - Check whether the queue is full – [i.e., the rear end is in just before the front end in a circular manner].
    - If it is full then display Queue is full.
        - If the queue is not full then, insert an element at the end of the queue.
- **deQueue()** This function is used to delete an element from the circular queue. In a circular queue, the element is always deleted from the front position.
    - Check whether the queue is Empty.
    - If it is empty then display Queue is empty.
        - If the queue is not empty, then get the last element and remove it from the queue.



Implement Circular Queue using Array:

5. Initialize an array queue of size **n**, where n is the maximum number of elements that the queue can hold.
6. Initialize two variables front and rear to -1.
7. **Enqueue:** To enqueue an element **x** into the queue, do the following:
   - Increment rear by 1.
     - If **rear** is equal to n, set **rear** to 0.
   - If **front** is -1, set **front** to 0.
   - Set queue[rear] to x.
8. **Dequeue:** To dequeue an element from the queue, do the following:
   - Check if the queue is empty by checking if **front** is -1.
     - If it is, return an error message indicating that the queue is empty.
   - Set **x** to queue [front].
   - If **front** is equal to **rear**, set **front** and **rear** to -1.
   - Otherwise, increment **front** by 1 and if **front** is equal to n, set **front** to 0.
   - Return x.

Hints:

```
class CircularQueue():

    # constructor
    def __init__(self, size): # initializing the class
        self.size = size

        # initializing queue with none
        self.queue = [None for i in range(size)]
        self.front = self.rear = -1

    def enqueue(self, data):
         # Write code here
         …

    def dequeue(self):
         # Write code here
         …

    def display(self):
        # Write code here
        …
```

```
# Driver Code
ob = CircularQueue(5)
ob.enqueue(14)
ob.enqueue(22)
ob.enqueue(13)
ob.enqueue(-6)
ob.display()
print ("Deleted value = ", ob.dequeue())
print ("Deleted value = ", ob.dequeue())
ob.display()
ob.enqueue(9)
ob.enqueue(20)
ob.enqueue(5)
ob.display()
```

TRY

1. Take input operations as [ENQUEUE(A,B,C,D,E,F),DEQUEUE(),DEQUEUE(),DEQUEUE(),ENQUEUE(G,H,I] and verify the result.

2. Take input operations as [DEQUEUE(),ENQUEUE(A,B,C,D,E,F),DEQUEUE(),ENQUEUE(G,H,I] and verify the result.

5. Graph Representation

5.1 Build a graph

You are given an integer n. Determine if there is an unconnected graph with n vertices that contains at least two connected components and contains the number of edges that is equal to the number of vertices. Each vertex must follow one of these conditions:

- Its degree is less than or equal to 1.
- It's a cut-vertex.

Note:

- The graph must be simple.
- Loops and multiple edges are not allowed.

Input: First line: n

Output: Print Yes if it is an unconnected graph. Otherwise, print No.

Sample Input	Sample Output
3	No

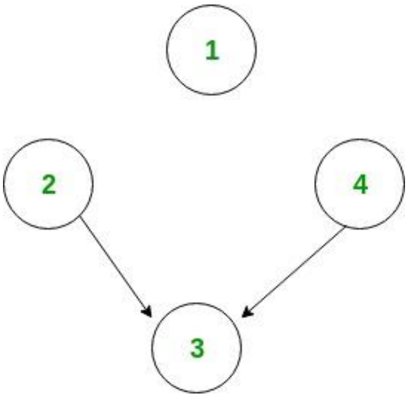
Constraints: 1 ≤ n ≤ 100

Explanation: There is only one graph with the number of edges equal to the number of vertices (triangle) which is connected.

5.2 Number of Sink Nodes in a Graph

Given a Directed Acyclic Graph of n nodes (numbered from 1 to n) and m edges. The task is to find the number of sink nodes. A sink node is a node such that no edge emerges out of it.

Input: n = 4, m = 2, edges[] = {{2, 3}, {4, 3}}



Only node 1 and node 3 are sink nodes.

Input: n = 4, m = 2, edges[] = {{3, 2}, {3, 4}}

Output: 3

The idea is to iterate through all the edges. And for each edge, mark the source node from which the edge emerged out. Now, for each node check if it is marked or not. And count the unmarked nodes.

Algorithm:

1. Make any array A[] of size equal to the number of nodes and initialize to 1.

2. Traverse all the edges one by one, say, u -> v.

   (i) Mark A[u] as 1.

3. Now traverse whole array A[] and count number of unmarked nodes.

Hints:

```
# Program to count number if sink nodes

# Return the number of Sink Nodes.
def countSink(n, m, edgeFrom, edgeTo):
     # Write code here
       …

    return count

# Driver Code
n = 4
m = 2
edgeFrom = [2, 4]
edgeTo = [3, 3]

print(countSink(n, m, edgeFrom, edgeTo))
```

5.3 Connected Components in a Graph

Given n, i.e. total number of nodes in an undirected graph numbered from 1 to n and an integer e, i.e. total number of edges in the graph. Calculate the total number of connected components in the graph. A connected component is a set of vertices in a graph that are linked to each other by paths.

Input: First line of input line contains two integers' n and e. Next e line will contain two integers u and v meaning that node u and node v are connected to each other in undirected fashion.

Output: For each input graph print an integer x denoting total number of connected components.

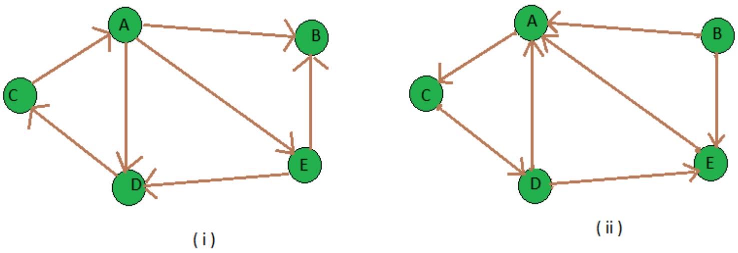
Sample Input	Sample Output

8 5	3
1 2	
2 3	
2 4	
3 5	
6 7	

Constraints: All the input values are well within the integer range.

5.4 Transpose Graph

Transpose of a directed graph G is another directed graph on the same set of vertices with all of the edges reversed compared to the orientation of the corresponding edges in G. That is, if G contains an edge (u, v) then the converse/transpose/reverse of G contains an edge (v, u) and vice versa. Given a graph (represented as adjacency list), we need to find another graph which is the transpose of the given graph.



Input: figure (i) is the input graph.

Output: figure (ii) is the transpose graph of the given graph.

```
0--> 2
1--> 0  4
2--> 3
3--> 0  4
4--> 0
```

Explanation: We traverse the adjacency list and as we find a vertex v in the adjacency list of vertex u which indicates an edge from u to v in main graph, we just add an edge from v to u in the transpose graph i.e. add u in the adjacency list of vertex v of the new graph. Thus traversing lists of all vertices of main graph we can get the transpose graph.

Hints:

```
# find transpose of a graph.
# function to add an edge from vertex source to vertex dest
def addEdge(adj, src, dest):
    adj[src].append(dest)

# function to print adjacency list of a graph
def displayGraph(adj, v):
     …
        print()

# function to get Transpose of a graph taking adjacency list of given graph and
that of Transpose graph
def transposeGraph(adj, transpose, v):
    # traverse the adjacency list of given graph and for each edge (u, v) add
    # an edge (v, u) in the transpose graph's adjacency list
    …

# Driver Code
v = 5
adj = [[] for i in range(v)]
addEdge(adj, 0, 1)
addEdge(adj, 0, 4)
addEdge(adj, 0, 3)
addEdge(adj, 2, 0)
addEdge(adj, 3, 2)
addEdge(adj, 4, 1)
addEdge(adj, 4, 3)

# Finding transpose of graph represented by adjacency list adj[]
transpose = [[]for i in range(v)]
transposeGraph(adj, transpose, v)

# Displaying adjacency list of transpose graph i.e. b
displayGraph(transpose, v)
```

TRY

1. Take input operations as addEdge(A, B), addEdge(A, D), addEdge(A, C), addEdge(C, A),addEdge(A, D), addEdge(C, B), addEdge(B, C) and verify the result.

5.5 Counting Triplets

You are given an undirected, complete graph G that contains N vertices. Each edge is colored in either white or black. You are required to determine the number of triplets (i, j, k) $(1 \le i < j < k \le N)$ of vertices such that the edges (i, j), (j, k), (i, k) are of the same color.

There are M white edges and (N (N-1)/2) – M black edges.

Input:

First line: Two integers – N and M $(3 \le N \le 10^5, 1 \le M \le 3 * 10^5)$

(i+1)th line: Two integers – u_i and v_i $(1 \le u_i, v_i \le N)$ denoting that the edge (u_i, v_i) is white in color.

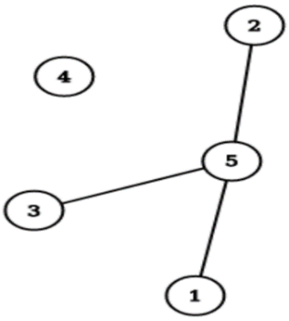
Note: The conditions $(u_i, v_i) \neq (u_j, v_j)$ and $(u_i, v_i) \neq (v_j, u_j)$ are satisfied for all $1 \le i < j \le M$.

.**Output:** Print an integer that denotes the number of triples that satisfy the mentioned condition.

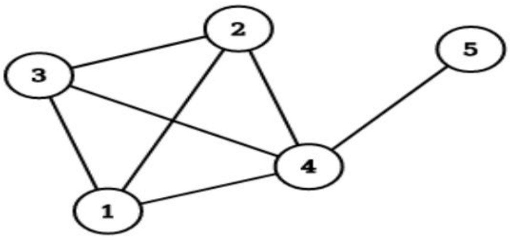
Sample Input	Sample Output
5 3	4
1 5	
2 5	
3 5	

Explanation: The triplets are: {(1, 2, 3), (1, 2, 4), (2, 3, 4), (1, 3, 4)}

The graph consisting of only white edges:



The graph consisting of only black edges:

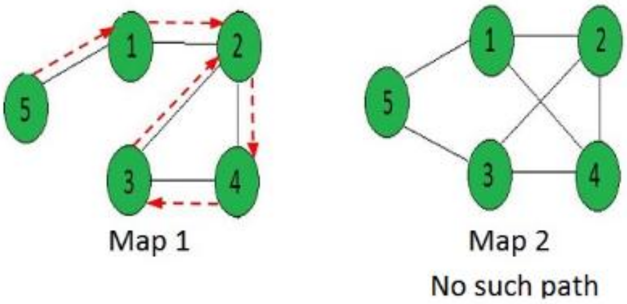


6. Graph Routing Algorithms

6.1 Seven Bridges of Konigsberg

There was 7 bridges connecting 4 lands around the city of Königsberg in Prussia. Was there any way to start from any of the land and go through each of the bridges once and only once? Euler first introduced graph theory to solve this problem. He considered each of the lands as a node of a graph and each bridge in between as an edge in between. Now he calculated if there is any Eulerian Path in that graph. If there is an Eulerian path then there is a solution otherwise not.

There are n nodes and m bridges in between these nodes. Print the possible path through each node using each edges (if possible), traveling through each edges only once.



Input: [[0, 1, 0, 0, 1],

[1, 0, 1, 1, 0],

[0, 1, 0, 1, 0],

[0, 1, 1, 0, 0],

[1, 0, 0, 0, 0]]

Output: 5 -> 1 -> 2 -> 4 -> 3 -> 2

Input: [[0, 1, 0, 1, 1],
[1, 0, 1, 0, 1],
[0, 1, 0, 1, 1],
[1, 1, 1, 0, 0],
[1, 0, 1, 0, 0]]

Output: "No Solution"

Hints:

```
# A Python program to print Eulerian trail in a
# given Eulerian or Semi-Eulerian Graph
from collections import defaultdict

class Graph:
# Constructor and destructor
def __init__(self, V):
self.V = V
 self.adj = defaultdict(list)

# functions to add and remove edge
def addEdge(self, u, v):
def rmvEdge(self, u, v):
   …
# Methods to print Eulerian tour
def printEulerTour(self):
 # Find a vertex with odd degree
  …
 # Print tour starting from oddv
self.printEulerUtil(u)
 print()
def printEulerUtil(self, u):
 # Recur for all the vertices adjacent to this vertex
for v in self.adj[u]:
 # If edge u-v is not removed and it's a valid next edge
# The function to check if edge u-v can be considered as next edge in Euler Tout
 def isValidNextEdge(self, u, v):
    # The edge u-v is valid in one of the following two cases:

    # 1) If v is the only adjacent vertex of u
        …

    # 2) If there are multiple adjacents, then u-v is not a bridge
    # Do following steps to check if u-v is a bridge
    # 2.a) count of vertices reachable from u
       …

    # 2.b) Remove edge (u, v) and after removing
    # the edge, count vertices reachable from u
        …

    # 2.c) Add the edge back to the graph
        self.addEdge(u, v)

    # 2.d) If count1 is greater, then edge (u, v) is a bridge
        return False if count1 > count2 else True

    # A DFS based function to count reachable vertices from v

    def DFSCount(self, v, visited):
        # Mark the current node as visited
        …
        # Recur for all the vertices adjacent to this vertex
        …
    # utility function to form edge between two vertices source and dest
    def makeEdge(src, dest):
        graph.addEdge(src, dest)

# Driver code
# Let us first create and test graphs shown in above figure
g1 = Graph(4)
g1.addEdge(0, 1)
g1.addEdge(0, 2)
g1.addEdge(1, 2)
g1.addEdge(2, 3)
g1.printEulerTour()

g3 = Graph(4)
g3.addEdge(0, 1)
g3.addEdge(1, 0)
g3.addEdge(0, 2)
g3.addEdge(2, 0)
g3.addEdge(2, 3)
g3.addEdge(3, 1)
g3.printEulerTour()
```

TRY

1. Take input: [[1, 0, 1, 0, 1], [1, 0, 1, 0, 0], [1, 1, 0, 1, 0], [0, 0, 1, 0, 0], [1, 0, 1, 0, 0]] and verify the result.

2. Take input: [[0, 0, 1, 0, 1], [0, 0, 1, 0, 0], [1, 0, 0, 1, 0], [1, 0, 1, 0, 0], [1, 1, 1, 0, 0]] and verify the result.

6.2 Hamiltonian Cycle

The Hamiltonian cycle of undirected graph G = <V, E> is the cycle containing each vertex in V. If graph contains a Hamiltonian cycle, it is called Hamiltonian graph otherwise it is non-Hamiltonian.

Hamiltonian Path in an undirected graph is a path that visits each vertex exactly once. A Hamiltonian cycle (or Hamiltonian circuit) is a Hamiltonian path such that there is an edge (in the graph) from the last vertex to the first vertex of the Hamiltonian path. Consider a graph G, and determine whether a given graph contains Hamiltonian cycle or not. If it contains, then prints the path. Following are the input and output of the required function.

Input: A 2D array graph [V][V] where V is the number of vertices in graph and graph [V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

Output: An array path [V] that should contain the Hamiltonian Path. Path [i] should represent the i^{th} vertex in the Hamiltonian Path. The code should also return false if there is no Hamiltonian Cycle in the graph.

For example, a Hamiltonian Cycle in the following graph is {0, 1, 2, 4, 3, 0}.

```
(0)--(1)--(2)
 |   /\   |
 |  /  \  |
 | /    \ |
(3)-------(4)
```

And the following graph doesn't contain any Hamiltonian Cycle.

```
(0)--(1)--(2)
 |    /\   |
 |   /  \  |
 | /    \  |
(3)      (4)
```

Backtracking Algorithm: Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex then we return false.

Hints:

```
# Python program for solution of Hamiltonian cycle problem

class Graph():
    def __init__(self, vertices):
        self.graph = [[0 for column in range(vertices)]
                              for row in range(vertices)]
        self.V = vertices

    def isSafe(self, v, pos, path):
        # Check if current vertex and last vertex in path are adjacent
      …

    # A recursive utility function to solve Hamiltonian cycle problem
    def hamCycleUtil(self, path, pos):
        …

    def hamCycle(self):
        …
    def printSolution(self, path):
        print ("Solution Exists: Following","is one Hamiltonian Cycle")
        for vertex in path:
            print (vertex, end = " ")
        print (path[0], "\n")

# Driver Code

''' Let us create the following graph
    (0)--(1)--(2)
    |    / \   |
    |   /   \  |
    |  /     \ |
    (3)-------(4) '''
g1 = Graph(5)
g1.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
            [0, 1, 0, 0, 1,],[1, 1, 0, 0, 1],
            [0, 1, 1, 1, 0], ]

# Print the solution
g1.hamCycle();

''' Let us create the following graph
    (0)--(1)--(2)
    |    / \   |
    |   /   \  |
    |  /     \ |
    (3)     (4) '''
g2 = Graph(5)
g2.graph = [ [0, 1, 0, 1, 0], [1, 0, 1, 1, 1],
        [0, 1, 0, 0, 1,], [1, 1, 0, 0, 0],
        [0, 1, 1, 0, 0], ]

# Print the solution
g2.hamCycle();
```

TRY

1. Take a graph = [[1, 1, 0, 1, 0], [1, 1, 1, 1, 1], [0, 1, 0, 1, 1,], [1, 1, 0, 1, 0], [0, 1, 1, 1, 0],] and verify the results.

6.3 Number of Hamiltonian Cycle

Given an undirected complete graph of N vertices where N > 2. The task is to find the number of different Hamiltonian cycle of the graph.

Complete Graph: A graph is said to be complete if each possible vertices is connected through an Edge.

Hamiltonian Cycle: It is a closed walk such that each vertex is visited at most once except the initial vertex. and it is not necessary to visit all the edges.

Formula: (N – 1)! / 2

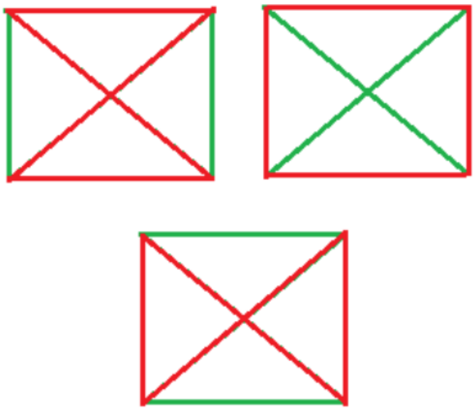
Input: N = 6

Output: Hamiltonian cycles = 60

Input: N = 4

Output: Hamiltonian cycles = 3

Explanation: Let us take the example of N = 4 complete undirected graph, The 3 different Hamiltonian cycle is as shown below:



Hints:

```
# Number of Hamiltonian cycles
import math as mt

# Function that calculates number of Hamiltonian cycle
def Cycles(N):
    …

# Driver code
N = 5
Number = Cycles(N)
print("Hamiltonian cycles = ", Number)
```

TRY

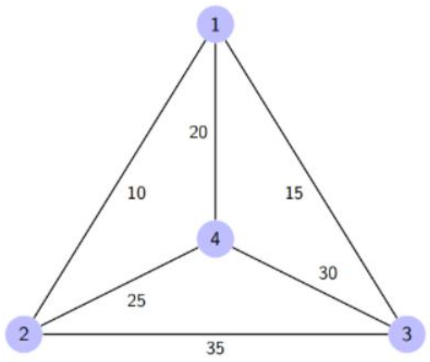
1. Take an input N=7 and verify the results.
2. Take an input N=10 and verify the results.

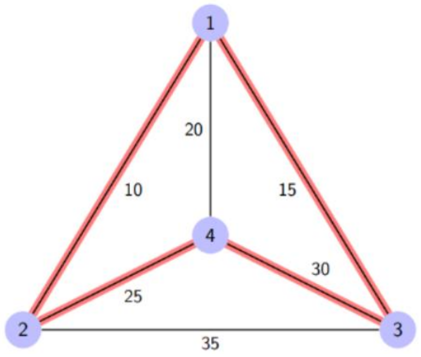
7. Shortest Path Algorithms

7.1 Travelling Salesman Problem

Given a set of cities and the distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point. The problem statement gives a list of cities along with the distances between each city.

Objective: To start from the origin city, visit other cities only once, and return to the original city again. Our target is to find the shortest possible path to complete the round-trip route.


1
20
10
15
4
30
25
2
35
3


1
20
10
15
4
30
25
2
35
3

Here a graph is given where 1, 2, 3, and 4 represent the cities, and the weight associated with every edge represents the distance between those cities. The goal is to find the shortest possible path for the tour that starts from the origin city, traverses the graph while only visiting the other cities or nodes once, and returns to the origin city.

For the above graph, the optimal route is to follow the minimum cost path: 1 – 2 – 4 – 3 - 1. And this shortest route would cost 10 + 25 + 30 + 15 =80

Algorithm for Traveling Salesman Problem: We will use the dynamic programming approach to solve the Travelling Salesman Problem (TSP).

- *A graph G=(V, E), which is a set of vertices and edges.*
- *V is the set of vertices.*
- *E is the set of edges.*
- *Vertices are connected through edges.*
- *Dist(i,j) denotes the non-negative distance between two vertices, i and j.*

Let's assume S is the subset of cities and belongs to {1, 2, 3, ..., n} where 1, 2, 3…n are the cities and i, j are two cities in that subset. Now cost(i, S, j) is defined in such a way as the length of the shortest path visiting node in S, which is exactly once having the starting and ending point as i and j respectively.

For example, cost (1, {2, 3, 4}, 1) denotes the length of the shortest path where:

- *Starting city is 1*
- *Cities 2, 3, and 4 are visited only once*
- *The ending point is 1*

The dynamic programming algorithm would be:

- *Set cost(i,, i) = 0, which means we start and end at i, and the cost is 0.*
- *When |S| > 1, we define cost(i, S, 1) = ∝ where i !=1 . Because initially, we do not know the exact cost to reach city i to city 1 through other cities.*
- *Now, we need to start at 1 and complete the tour. We need to select the next city in such a way-*
- *cost(i, S, j) = min cost (i, S−{i}, j) + dist(i, j) where i ∈S and i ≠ j*

For the given figure above, the adjacency matrix would be the following:

dist(i, j)	1	2	3	4
1	0	10	15	20
2	10	0	35	25
3	15	35	0	30
4	20	25	30	0

Now S = {1, 2, 3, 4}. There are four elements. Hence the number of subsets will be 2^4 or 16. Those subsets are-

1) |S| = Null:

{Φ}

2) |S| = 1:

{{1}, {2}, {3}, {4}}

3) |S| = 2:

{{1, 2}, {1, 3}, {1, 4}, {2, 3}, {2, 4}, {3, 4}}

4) |S| = 3:
{{1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 3, 4}}

5) |S| = 4:
{{1, 2, 3, 4}}

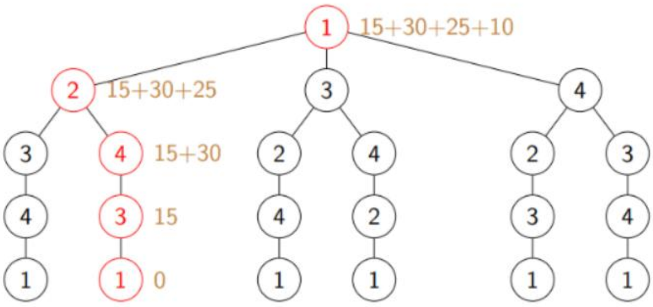
As we are starting at 1, we could discard the subsets containing city 1. The algorithm calculation steps:
1) |S| = Φ:
cost (2, Φ, 1) = dist(2, 1) = 10
cost (3, Φ, 1) = dist(3, 1) = 15
cost (4, Φ, 1) = dist(4, 1) = 20

2) |S| = 1:
cost (2, {3}, 1) = dist(2, 3) + cost (3, Φ, 1) = 35+15 = 50
cost (2, {4}, 1) = dist(2, 4) + cost (4, Φ, 1) = 25+20 = 45
cost (3, {2}, 1) = dist(3, 2) + cost (2, Φ, 1) = 35+10 = 45
cost (3, {4}, 1) = dist(3, 4) + cost (4, Φ, 1) = 30+20 = 50
cost (4, {2}, 1) = dist(4, 2) + cost (2, Φ, 1) = 25+10 = 35
cost (4, {3}, 1) = dist(4, 3) + cost (3, Φ, 1) = 30+15 = 45

3) |S| = 2:
cost (2, {3, 4}, 1) = min [dist[2,3] + Cost(3,{4},1) = 35+50 = 85,
dist[2,4]+Cost(4,{3},1) = 25+45 = 70] = 70
cost (3, {2, 4}, 1) = min [dist[3,2] + Cost(2,{4},1) = 35+45 = 80,
dist[3,4]+Cost(4,{2},1) = 30+35 = 65] = 65
cost (4, {2, 3}, 1) = min [dist[4,2] + Cost(2,{3},1) = 25+50 = 75
dist[4,3] + Cost(3,{2},1) = 30+45 = 75] = 75

4) |S| = 3:
cost (1, {2, 3, 4}, 1) = min [dist[1,2]+Cost(2,{3,4},1) = 10+70 = 80
dist[1,3]+Cost(3,{2,4},1) = 15+65 = 80
dist[1,4]+Cost(4,{2,3},1) = 20+75 = 95] = 80

So the optimal solution would be 1-2-4-3-1



Hints:

```
from sys import maxsize
from itertools, import permutations
V = 4
def tsp(graph, s):
        …
```

```
# Driver code
graph = [[0, 10, 15, 20], [10, 0, 35, 25], [15, 35, 0, 30], [20, 25, 30, 0]]
s = 0
print(tsp(graph, s))
```

TRY

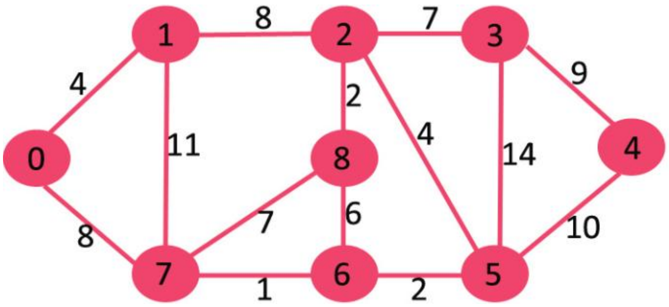
1. Take a below table values and verify the results.

dist(i, j)	*1*	*2*	*3*	4
1	*0*	*40*	*25*	40
2	*20*	*0*	*35*	25
3	*25*	*35*	*0*	60
4	*40*	*25*	*30*	0

7.2 Shortest Paths from Source to all Vertices (Dijkstra's Algorithm)

Given a graph and a source vertex in the graph, find the shortest paths from the source to all vertices in the given graph.

Input: src = 0, the graph is shown below.



Output: 0 4 12 19 21 11 9 8 14

Explanation: The distance from 0 to 1 = 4.
The minimum distance from 0 to 2 = 12. 0->1->2
The minimum distance from 0 to 3 = 19. 0->1->2->3
The minimum distance from 0 to 4 = 21. 0->7->6->5->4
The minimum distance from 0 to 5 = 11. 0->7->6->5
The minimum distance from 0 to 6 = 9. 0->7->6
The minimum distance from 0 to 7 = 8. 0->7
The minimum distance from 0 to 8 = 14. 0->1->2->8

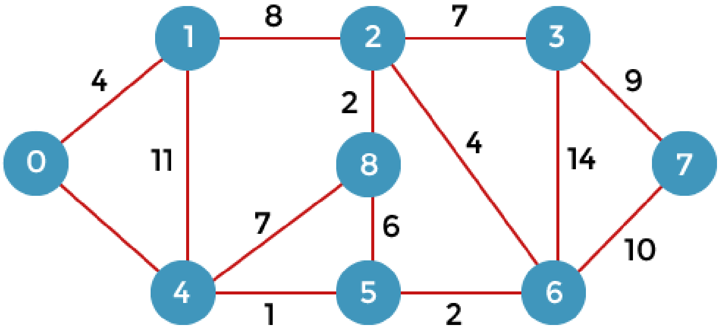
Hints:

```
# Dijkstra's single source shortest path algorithm. The program is for adjacency
matrix representation of the graph

# Library for INT_MAX
import sys
```

```
class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    def printSolution(self, dist):
        print("Vertex \tDistance from Source")
        for node in range(self.V):
            print(node, "\t", dist[node])

    # A utility function to find the vertex with minimum distance value,
    # from the set of vertices not yet included in shortest path tree
    def minDistance(self, dist, sptSet):
        …

    # Function that implements Dijkstra's single source shortest path
    # algorithm for a graph represented using adjacency matrix representation
    def dijkstra(self, src):
        …


# Driver's code
g = Graph(9)
g.graph = [[0, 4, 0, 0, 0, 0, 0, 8, 0],
           [4, 0, 8, 0, 0, 0, 0, 11, 0],
           [0, 8, 0, 7, 0, 4, 0, 0, 2],
           [0, 0, 7, 0, 9, 14, 0, 0, 0],
           [0, 0, 0, 9, 0, 10, 0, 0, 0],
           [0, 0, 4, 14, 10, 0, 2, 0, 0],
           [0, 0, 0, 0, 0, 2, 0, 1, 6],
           [8, 11, 0, 0, 0, 0, 1, 0, 7],
           [0, 0, 2, 0, 0, 0, 6, 7, 0]
          ]

g.dijkstra(0)
```

TRY

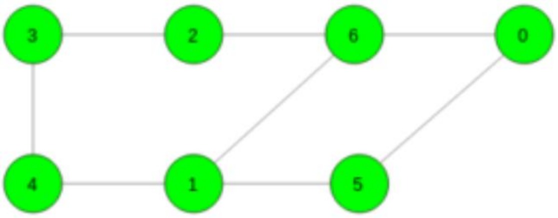
1. Take a below graph and verify the results.



7.3 Shortest Cycle in an Undirected Unweighted Graph

Given an undirected unweighted graph. The task is to find the length of the shortest cycle in the given graph. If no cycle exists print -1.

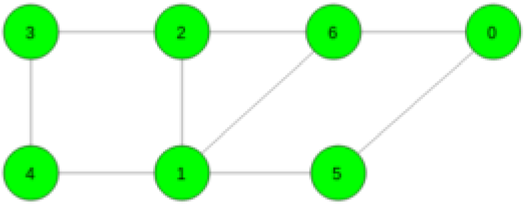
Input: Consider the graph given below



Output: 4

Cycle 6 -> 1 -> 5 -> 0 -> 6

Input: Consider the graph given below



Output: 3

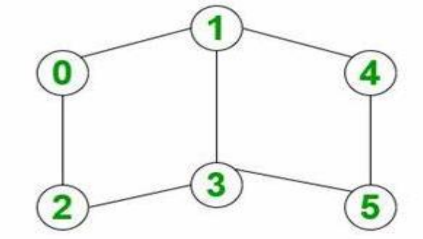
Cycle 6 -> 1 -> 2 -> 6

Hints:

```
from sys import maxsize as INT_MAX
from collections import deque

N = 100200

gr = [0] * N
for i in range(N):
    gr[i] = []

# Function to add edge
def add_edge(x: int, y: int) -> None:
    global gr
    gr[x].append(y)
    gr[y].append(x)

# Function to find the length of the shortest cycle in the graph
def shortest_cycle(n: int) -> int:

    # To store length of the shortest cycle
    ans = INT_MAX

    # For all vertices
    # write code here
    …

    # If graph contains no cycle
    if ans == INT_MAX:
        return -1

    # If graph contains cycle
    else:
        return ans

# Driver Code
# Number of vertices
n = 7
# Add edges
add_edge(0, 6)
add_edge(0, 5)
add_edge(5, 1)
add_edge(1, 6)
add_edge(2, 6)
add_edge(2, 3)
add_edge(3, 4)
add_edge(4, 1)

# Function call
print(shortest_cycle(n))
```

TRY

1. Take a below graph and verify the results.



7.4 Count Unique and all Possible Paths in a M x N Matrix

Count unique paths: The problem is to count all unique possible paths from the top left to the bottom right of a M X N matrix with the constraints that from each cell you can either move only to the right or down.

Input: M = 2, N = 2

Output: 2

Explanation: There are two paths
(0, 0) -> (0, 1) -> (1, 1)
(0, 0) -> (1, 0) -> (1, 1)

Input: M = 2, N = 3

Output: 3

Explanation: There are three paths
(0, 0) -> (0, 1) -> (0, 2) -> (1, 2)

(0, 0) -> (0, 1) -> (1, 1) -> (1, 2)
(0, 0) -> (1, 0) -> (1, 1) -> (1, 2)

Count all possible paths: We can recursively move to right and down from the start until we reach the destination and then add up all valid paths to get the answer.

Procedure:

- Create a recursive function with parameters as row and column index
- Call this recursive function for N-1 and M-1
- In the recursive function
    - If N == 1 or M == 1 then return 1
    - else call the recursive function with (N-1, M) and (N, M-1) and return the sum of this
- Print the answer

Hints:

```
# Python program to count all possible paths from top left to bottom right

# Function to return count of possible paths to reach cell at row number m and
column number n from the topmost leftmost cell (cell at 1, 1)

def numberOfPaths(m, n):
    …

# Driver program to test above function
m = 3
n = 3
print(numberOfPaths(m, n))
```

TRY

1. Take input : M = 3, N = 2 and verify the results.
2. Take input : M = 2, N = 1 and verify the results.

8. Graph Coloring

8.1 Graph Coloring using Greedy Algorithm

Greedy algorithm is used to assign colors to the vertices of a graph. It doesn't guarantee to use minimum colors, but it guarantees an upper bound on the number of colors. The basic algorithm never uses more than d+1 colors where d is the maximum degree of a vertex in the given graph.

Basic Greedy Coloring Algorithm:

1. Color first vertex with first color.
2. Do following for remaining V-1 vertices.
    a) Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

Hints:

```
# Implement greedy algorithm for graph coloring

def addEdge(adj, v, w):
    adj[v].append(w)
    # Note: the graph is undirected
    adj[w].append(v)
    return adj

# Assigns colors (starting from 0) to all
# vertices and prints the assignment of colors
def greedyColoring(adj, V):
    …

# Driver Code
g1 = [[] for i in range(5)]
g1 = addEdge(g1, 0, 1)
g1 = addEdge(g1, 0, 2)
g1 = addEdge(g1, 1, 2)
g1 = addEdge(g1, 1, 3)
g1 = addEdge(g1, 2, 3)
g1 = addEdge(g1, 3, 4)
print("Coloring of graph 1 ")
greedyColoring(g1, 5)

g2 = [[] for i in range(5)]
g2 = addEdge(g2, 0, 1)
g2 = addEdge(g2, 0, 2)
g2 = addEdge(g2, 1, 2)
g2 = addEdge(g2, 1, 4)
g2 = addEdge(g2, 2, 4)
g2 = addEdge(g2, 4, 3)
print("\nColoring of graph 2")
greedyColoring(g2, 5)
```

Output:

```
Coloring of graph 1
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 1

Coloring of graph 2
Vertex 0 --->  Color 0
Vertex 1 --->  Color 1
Vertex 2 --->  Color 2
Vertex 3 --->  Color 0
Vertex 4 --->  Color 3
```

8.2 Coloring a Cycle Graph

Given the number of vertices in a Cyclic Graph. The task is to determine the Number of colors required to color the graph so that no two adjacent vertices have the same color.

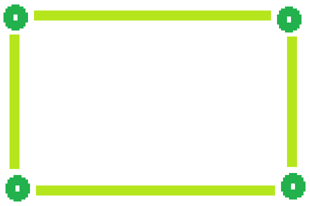
Approach:

- If the no. of vertices is Even then it is Even Cycle and to color such graph we require 2 colors.
- If the no. of vertices is Odd then it is Odd Cycle and to color such graph we require 3 colors.

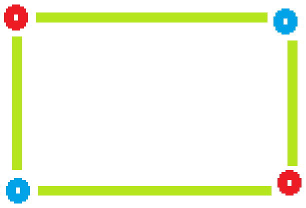
Input: Vertices = 3
Output: No. of colors require is: 3

Input: vertices = 4
Output: No. of colors require is: 2

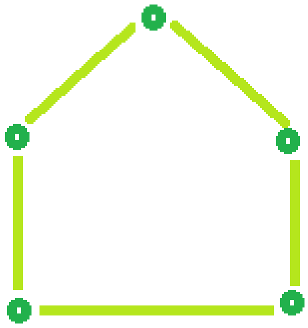
Example 1: Even Cycle: Number of vertices = 4



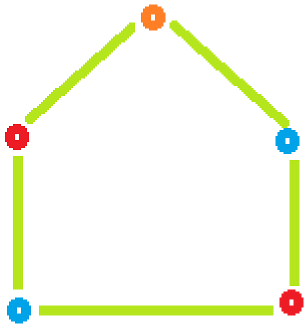
Color required = 2



Example 2: Odd Cycle: Number of vertices = 5



Color required = 3



Hints:

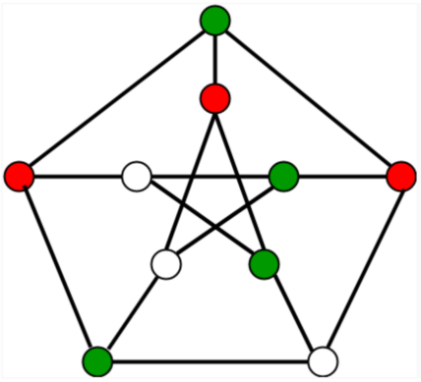
```
# Find the number of colors required to color a cycle graph

# Function to find Color required.
def Color(vertices):
    …
```

```
# Driver Code
vertices = 3
print ("No. of colors require is:", Color(vertices))
```

8.3 m Coloring Problem

Given an undirected graph and a number m, determine if the graph can be colored with at most m colors such that no two adjacent vertices of the graph are colored with the same color.

Note: Here coloring of a graph means the assignment of colors to all vertices
Following is an example of a graph that can be colored with 3 different colors:



Input: graph = {0, 1, 1, 1},
{1, 0, 1, 0},
{1, 1, 0, 1},
{1, 0, 1, 0}
Output: Solution Exists: Following are the assigned colors: 1 2 3 2
Explanation: By coloring the vertices with following colors, adjacent vertices does not have same colors

Input: graph = {1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1},
{1, 1, 1, 1}
Output: Solution does not exist
Explanation: No solution exits

Generate all possible configurations of colors. Since each node can be colored using any of the m available colors, the total number of color configurations possible is mV. After generating a configuration of color, check if the adjacent vertices have the same color or not. If the conditions are met, print the combination and break the loop
Follow the given steps to solve the problem:

- Create a recursive function that takes the current index, number of vertices and output color array
- If the current index is equal to number of vertices. Check if the output color configuration is safe, i.e check if the adjacent vertices do not have same color. If the conditions are met, print the configuration and break
- Assign a color to a vertex (1 to m)
- For every assigned color recursively call the function with next index and number of vertices
- If any recursive function returns true break the loop and returns true.

Hints:

```
# Number of vertices in the graph
# define 4 4

# check if the colored graph is safe or not


def isSafe(graph, color):
    # check for every edge
    for i in range(4):
        for j in range(i + 1, 4):
            if (graph[i][j] and color[j] == color[i]):
                return False
    return True


def graphColoring(graph, m, i, color):
    # write your code here
    …

# /* A utility function to print solution */

def printSolution(color):
    print("Solution Exists:" " Following are the assigned colors ")
    for i in range(4):
        print(color[i], end=" ")


# Driver code
# /* Create following graph and test whether it is 3 colorable
# (3)---(2)
# |   / |
# |  /  |
# | /   |
# (0)---(1)
# */
graph = [
        [0, 1, 1, 1],
        [1, 0, 1, 0],
        [1, 1, 0, 1],
        [1, 0, 1, 0],
        ]
m = 3  # Number of colors

# Initialize all color values as 0.
# This initialization is needed
# correct functioning of isSafe()
color = [0 for i in range(4)]

# Function call
if (not graphColoring(graph, m, 0, color)):
        print("Solution does not exist")
```

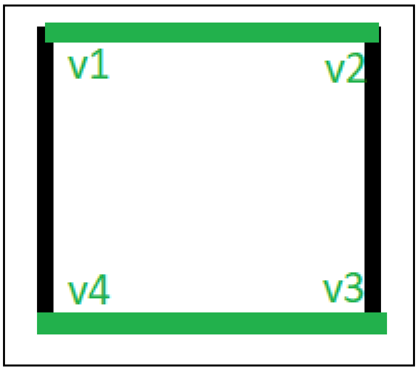
8.4 Edge Coloring of a Graph

Edge coloring of a graph is an assignment of “colors” to the edges of the graph so that no two adjacent edges have the same color with an optimal number of colors. Two edges are said to be adjacent if they are connected to the same vertex. There is no known polynomial time algorithm for edge-coloring every graph with an optimal number of colors.

Input: u1 = 1, v1 = 4
u2 = 1, v2 = 2
u3 = 2, v3 = 3
u4 = 3, v4 = 4

Output: Edge 1 is of color 1
Edge 2 is of color 2
Edge 3 is of color 1
Edge 4 is of color 2

The above input shows the pair of vertices (u_i, v_i) who have an edge between them. The output shows the color assigned to the respective edges.



Edge colorings are one of several different types of graph coloring problems. The above figure of a Graph shows an edge coloring of a graph by the colors green and black, in which no adjacent edge have the same color.

Algorithm:

1. Use BFS traversal to start traversing the graph.
2. Pick any vertex and give different colors to all of the edges connected to it, and mark those edges as colored.
3. Traverse one of it's edges.
4. Repeat step to with a new vertex until all edges are colored.

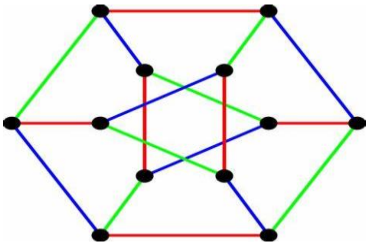
Hints:

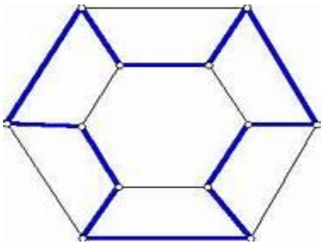
```
# Edge Coloring

from queue import Queue

# function to determine the edge colors
def colorEdges(ptr, gra, edgeColors, isVisited):
      # Write your code here
      …


# Driver Function
empty=set()
# declaring vector of vector of pairs, to define Graph
gra=[]
edgeColors=[]
isVisited=[False]*100000
```

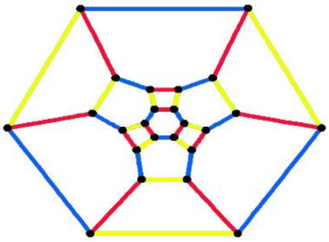
```
ver = 4
edge = 4
gra=[[] for _ in range(ver)]
edgeColors=[-1]*edge
gra[0].append((1, 0))
gra[1].append((0, 0))
gra[1].append((2, 1))
gra[2].append((1, 1))
gra[2].append((3, 2))
gra[3].append((2, 2))
gra[0].append((3, 3))
gra[3].append((0, 3))
colorEdges(0, gra, edgeColors, isVisited)

# printing all the edge colors
for i in range(edge):
    print("Edge {} is of color {}".format(i + 1,edgeColors[i] + 1))
```

TRY

1. Write a program to implement graph coloring and edge coloring by consider the below graph and verify the results.







9. Graph Traversal

9.1 Breadth First Search

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.
For a given graph G, print BFS traversal from a given source vertex.

Hints:

```
# BFS traversal from a given source vertex.

from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:

    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)

    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):
      # Write your code here
      …


# Create a graph given in the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print("Following is Breadth First Traversal" " (starting from vertex 2)")
g.BFS(2)
```

Output: Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1

9.2 Depth First Search

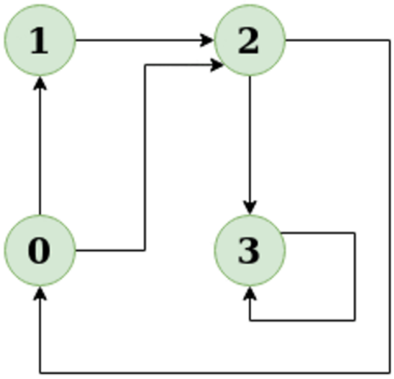
Depth First Traversal (or DFS) for a graph is similar to Depth First Traversal of a tree. The only catch here is, that, unlike trees, graphs may contain cycles (a node may be visited twice). To avoid processing a node more than once, use a boolean visited array. A graph can have more than one DFS traversal.

For a given graph G, print DFS traversal from a given source vertex.

Input: n = 4, e = 6
0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

Output: DFS from vertex 1: 1 2 0 3

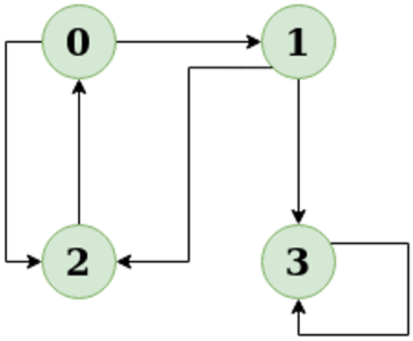
Explanation:
DFS Diagram:



Input: n = 4, e = 6
2 -> 0, 0 -> 2, 1 -> 2, 0 -> 1, 3 -> 3, 1 -> 3

Output: DFS from vertex 2: 2 0 1 3

Explanation:
DFS Diagram:

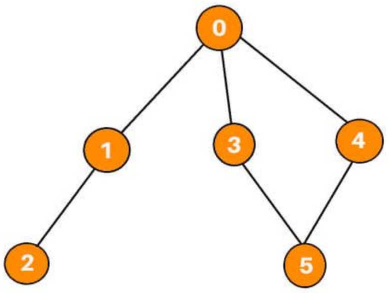


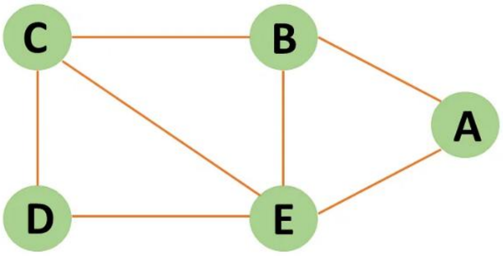
Hints:

```
# DFS traversal from a given graph
from collections import defaultdict

# This class represents a directed graph using adjacency list representation
class Graph:
    # Constructor
    def __init__(self):
        # Default dictionary to store graph
        self.graph = defaultdict(list)


    # Function to add an edge to graph
    def addEdge(self, u, v):
        self.graph[u].append(v)


    # A function used by DFS
    def DFSUtil(self, v, visited):
        …


    # The function to do DFS traversal. It uses recursive DFSUtil()

    def DFS(self, v):
      # Write your code here
        …

# Driver's code
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
print("Following is Depth First Traversal (starting from vertex 2)")
# Function call
g.DFS(2)
```

TRY

1. Write a program to implement breadth first search and depth first search by consider the below graph and verify the results.





10. Minimum Spanning Tree (MST)

10.1 Kruskal's Algorithm

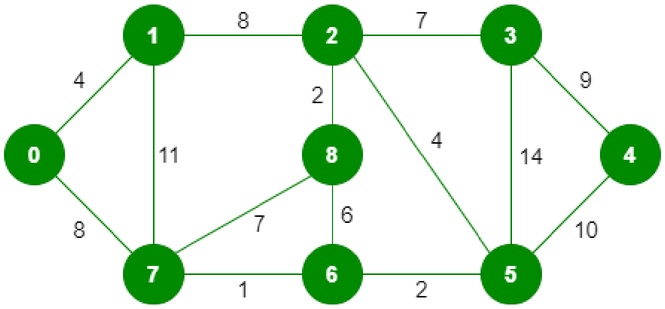
In Kruskal's algorithm, sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last.

MST using Kruskal's algorithm:

4. Sort all the edges in non-decreasing order of their weight.
5. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
6. Repeat step#2 until there are (V-1) edges in the spanning tree.

Kruskal's algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far.

Input: For the given graph G find the minimum cost spanning tree.



The graph contains 9 vertices and 14 edges. So, the minimum spanning tree formed will be having (9 – 1) = 8 edges.

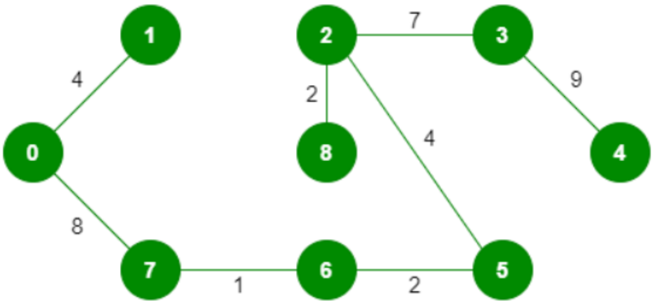
After sorting:

Weight	Source	Destination
1	7	6
2	8	2
2	6	5
4	0	1
4	2	5
6	8	6
7	2	3
7	7	8
8	0	7
8	1	2
9	3	4

10	5	4
11	1	7
14	3	5

Now pick all edges one by one from the sorted list of edges.

Output:



Hints:

```
# Kruskal's algorithm to find minimum Spanning Tree of a given connected,
# undirected and weighted graph

# Class to represent a graph
class Graph:
    def __init__(self, vertices):
        self.V = vertices
        self.graph = []

    # Function to add an edge to graph
    def addEdge(self, u, v, w):
        self.graph.append([u, v, w])

    def find(self, parent, i):
        …

    def union(self, parent, rank, x, y):
        …

    def KruskalMST(self):
      # write your code here
      …

# Driver code
g = Graph(4)
g.addEdge(0, 1, 10)
g.addEdge(0, 2, 6)
g.addEdge(0, 3, 5)
g.addEdge(1, 3, 15)
g.addEdge(2, 3, 4)

# Function call
g.KruskalMST()
```

Output: Following are the edges in the constructed MST
2 -- 3 == 4

0 -- 3 == 5

0 -- 1 == 10

Minimum Cost Spanning Tree: 19

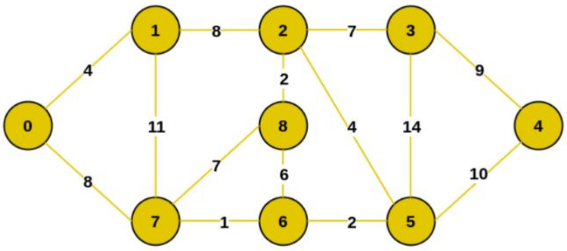
10.2 Prim's Algorithm

The Prim's algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

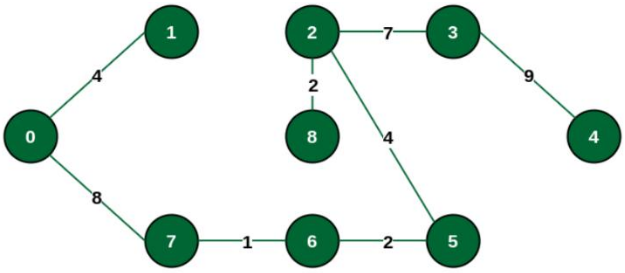
Prim's Algorithm:
The working of Prim's algorithm can be described by using the following steps:

7. Determine an arbitrary vertex as the starting vertex of the MST.
8. Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
9. Find edges connecting any tree vertex with the fringe vertices.
10. Find the minimum among these edges.
11. Add the chosen edge to the MST if it does not form any cycle.
12. Return the MST and exit

Input: For the given graph G find the minimum cost spanning tree.



Output: The final structure of the MST is as follows and the weight of the edges of the MST is (4 + 8 + 1 + 2 + 4 + 2 + 7 + 9) = 37.



Hints:

```
# Prim's Minimum Spanning Tree (MST) algorithm.
# The program is for adjacency matrix representation of the graph

# Library for INT_MAX
```

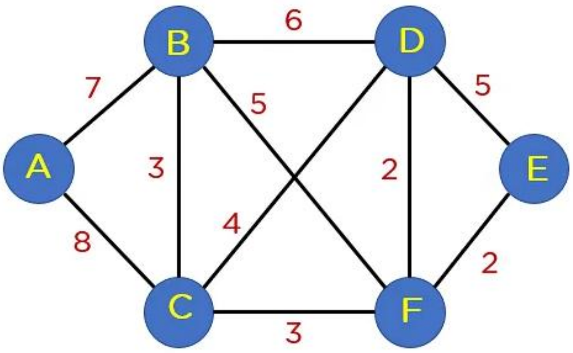
```
import sys

class Graph():
    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # A utility function to print
    # the constructed MST stored in parent[]
    def printMST(self, parent):
        print("Edge \tWeight")
        for i in range(1, self.V):
            print(parent[i], "-", i, "\t", self.graph[i][parent[i]])

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        …
    def primMST(self):
        …


# Driver's code
g = Graph(5)
g.graph = [[0, 2, 0, 6, 0],
           [2, 0, 3, 8, 5],
           [0, 3, 0, 0, 7],
           [6, 8, 0, 0, 9],
           [0, 5, 7, 9, 0]]

g.primMST()
```

Output:

```
Edge   Weight

0 - 1    2

1 - 2    3

0 - 3    6

1 - 4    5
```

TRY

1. Write a program to implement kruskal’s algorithm and prim’s algorithm by consider the below graph and verify the results.



11. Roots of Equations

11.1 Bisection Method

The Bisection method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which f(x) = 0. The method is based on **The Intermediate Value Theorem** which states that if f(x) is a continuous function and there are two real numbers a and b such that f(a) * f(b) 0 and f(b) < 0), then it is guaranteed that it has at least one root between them.

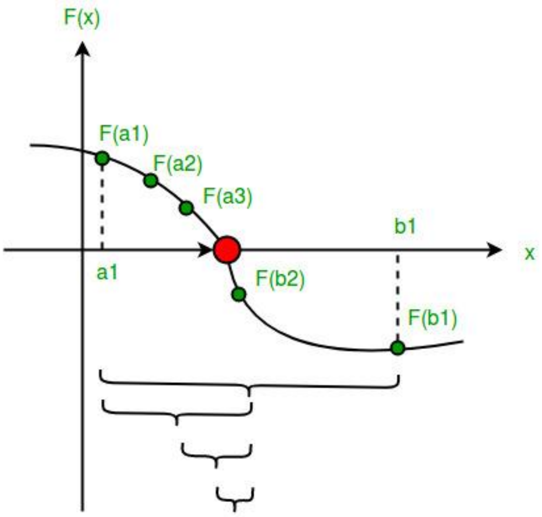
Assumptions:

1. f(x) is a continuous function in interval [a, b]
2. f(a) * f(b) < 0

Steps:

1. Find middle point c= (a + b)/2.
2. If f(c) == 0, then c is the root of the solution.
3. Else f(c) != 0
   - If value f(a)*f(c) < 0 then root lies between a and c. So we recur for a and c
   - Else If f(b)*f(c) < 0 then root lies between b and c. So we recur b and c.
   - Else given function doesn't follow one of assumptions.

Since root may be a floating point number, we repeat above steps while difference between a and b is greater than and equal to a value? (A very small value).



Hints:

```
# An example function whose solution is determined using Bisection Method.
# The function is x^3 - x^2  + 2
def func(x):
    return x*x*x - x*x + 2

# Prints root of func(x) with error of EPSILON
def bisection(a,b):
    # Write your code here
    …

# Driver code
# Initial values assumed
a =-200
b = 300
bisection (a, b)
```

Output: The value of root is : -1.0025

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.2 Method of False Position

Given a function f(x) on floating number x and two numbers 'a' and 'b' such that f(a)*f(b) < 0 and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).

Input: A function of x, for example x3 – x2 + 2.
And two values: a = -200 and b = 300 such that
f(a)*f(b) < 0, i.e., f(a) and f(b) have opposite signs.

Output: The value of root is : -1.00
OR any other value close to root.

Hints:

```
MAX_ITER = 1000000

# An example function whose solution is determined using Regular Falsi Method.
# The function is x^3 - x^2 + 2
def func( x ):
    return (x * x * x - x * x + 2)

# Prints root of func(x) in interval [a, b]
def regulaFalsi( a , b):
    # Write your code here
    …

# Driver code to test above function
# Initial values assumed
a =-200
b = 300
regulaFalsi(a, b)
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.3 Newton Raphson Method

Given a function f(x) on floating number x and an initial guess for root, find root of function in interval. Here f(x) represents algebraic or transcendental equation.

Input: A function of x (for example $x^3 – x^2 + 2$), derivative function of x ($3x^2 – 2x$ for above example) and an initial guess x0 = -20

Output: The value of root is: -1.00 or any other value close to root.

Algorithm:
Input: initial x, func(x), derivFunc(x)
Output: Root of Func()

```
2.      Compute values of func(x) and derivFunc(x) for given initial x
3.      Compute h: h = func(x) / derivFunc(x)
4.      While h is greater than allowed error ε
    •       h = func(x) / derivFunc(x)
    •       x = x – h
```

Hints:

```
# Implementation of Newton Raphson Method for solving equations
# An example function whose solution is determined using Bisection Method.
# The function is x^3 - x^2 + 2
def func( x ):
    return x * x * x - x * x + 2

# Derivative of the above function which is 3*x^x - 2*x
def derivFunc( x ):
    return 3 * x * x - 2 * x

# Function to find the root
def newtonRaphson( x ):
    # Write your code here
    …

# Driver program
x0 = -20
newtonRaphson(x0)
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

11.4 Secant Method

The secant method is used to find the root of an equation f(x) = 0. It is started from two distinct estimates x1 and x2 for the root. It is an iterative procedure involving linear interpolation to a root. The iteration stops if the difference between two intermediate values is less than the convergence factor.

Input: Equation = x3 + x – 1
x1 = 0, x2 = 1, E = 0.0001

Output: Root of the given equation = 0.682326
No. of iteration=5

Algorithm

```
Initialize: x1, x2, E, n         // E = convergence indicator
calculate f(x1),f(x2)

if(f(x1) * f(x2) = E); //repeat the loop until the convergence
    print 'x0' //value of the root
    print 'n' //number of iteration
}
else
    print "cannot found a root in the given interval"
```

Hints:

```
# Find root of an equations using secant method
# Function takes value of x and returns f(x)
def f(x):
    # we are taking equation as x^3+x-1
    f = pow(x, 3) + x - 1;
    return f;

def secant(x1, x2, E):
   # Write your code here
    …

# Driver code
# initializing the values
x1 = 0;
x2 = 1;
E = 0.0001;
secant(x1, x2, E);
```

TRY

1. Take an input function x^2 - x + 2 and verify the results.
2. Take an input function x^3 - x^2 + 4 and verify the results.

11.5 Muller Method

Given a function f(x) on floating number x and three initial distinct guesses for root of the function, find the root of function. Here, f(x) can be an algebraic or transcendental function.

Input: A function f(x) = x   + 2x   + 10x - 20 and three initial guesses - 0, 1 and 2 .

Output: The value of the root is 1.3688 or any other value within permittable deviation from the root.

Input: A function f(x) = x   - 5x + 2 and three initial guesses - 0, 1 and 2.

Output: The value of the root is 0.4021 or any other value within permittable deviation from the root.

Hints:

```
# Find root of a function, f(x)
import math;
```

```
MAX_ITERATIONS = 10000;

# Function to calculate f(x)
def f(x):
    # Taking f(x) = x ^ 3 + 2x ^ 2 + 10x - 20
    return (1 * pow(x, 3) + 2 * x * x +
                           10 * x - 20);

def Muller(a, b, c):
    # Write your code here
    …


# Driver Code
a = 0;
b = 1;
c = 2;
Muller(a, b, c);
```

TRY

1. Take an input function x^2 - x^3 + 2 and verify the results.
2. Take an input function x^3 - x^3 + 4 and verify the results.

12. Numerical Integration

12.1 Trapezoidal Rule for Approximate Value of Definite Integral

Trapezoidal rule is used to find the approximation of a definite integral. The basic idea in Trapezoidal rule is to assume the region under the graph of the given function to be a trapezoid and calculate its area.

$$\int_a^b f(x)\,dx \approx (b-a)\left[\frac{f(a)+f(b)}{2}\right]$$

Hints:

```
# Implement Trapezoidal rule

# A sample function whose definite integral's approximate value is
# computed using Trapezoidal rule
def y( x ):

    # Declaring the function
    # f(x) = 1/(1+x*x)
    return (1 / (1 + x * x))

# Function to evaluate the value of integral
def trapezoidal (a, b, n):
    # Write your code here
    …

# Driver code
# Range of definite integral
x0 = 0
xn = 1

# Number of grids. Higher value means more accuracy
n = 6
print ("Value of integral is ",
     "%.4f"%trapezoidal(x0, xn, n))
```

12.2 Simpson's 1/3 Rule

Simpson's 1/3 rule is a method for numerical approximation of definite integrals. Specifically, it is the following approximation:

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6}\left(f(a)+4f\frac{(a+b)}{2}+f(b)\right)$$

Procedure:

In order to integrate any function f(x) in the interval (a, b), follow the steps given below:

1. Select a value for n, which is the number of parts the interval is divided into.
2. Calculate the width, h = (b-a)/n
3. Calculate the values of x0 to xn as x0 = a, x1 = x0 + h, ...xn-1 = xn-2 + h, xn = b. Consider y = f(x). Now find the values of y (y0 to yn) for the corresponding x (x0 to xn) values.
4. Substitute all the above found values in the Simpson's Rule Formula to calculate the integral value.

Approximate value of the integral can be given by **Simpson's Rule:**

$$\int_a^b f(x)dx \approx \frac{h}{3}\left(f_0+f_n+4*\sum_{i=1,3,5}^{n-1} f_i+2*\sum_{i=2,4,6}^{n-2} f_i\right)$$

Input: Evaluate logx dx within limit 4 to 5.2.

First we will divide interval into six equal parts as number of interval should be even.

x :	4	4.2	4.4	4.6	4.8	5.0	5.2
log$_x$:	1.38	1.43	1.48	1.52	1.56	1.60	1.64

Output: Now we can calculate approximate value of integral using above formula:

= h/3[(1.38 + 1.64) + 4 * (1.43 + 1.52 + 1.60) +2 *(1.48 + 1.56)]

= 1.84

Hence the approximation of above integral is

1.827 using Simpson's 1/3 rule.

Hints:

```
# Simpson's 1 / 3 rule
import math

# Function to calculate f(x)
def func(x):
    return math.log(x)

# Function for approximate integral
def simpsons_(ll, ul, n):
     # Write your code here
    …

# Driver code
lower_limit = 4       # Lower limit
upper_limit = 5.2    # Upper limit
n = 6                # Number of interval
print("%.6f"% simpsons_(lower_limit, upper_limit, n))
```

12.3 Simpson's 3/8 Rule

The Simpson's 3/8 rule was developed by Thomas Simpson. This method is used for performing numerical integrations. This method is generally used for numerical approximation of definite integrals. Here, parabolas are used to approximate each part of curve.

Input: lower_limit = 1, upper_limit = 10, interval_limit = 10

Output: integration_result = 0.687927

Input: lower_limit = 1, upper_limit = 5, interval_limit = 3

Output: integration_result = 0.605835

Hints:

```
# Implement Simpson's 3/8 rule

# Given function to be integrated
def func(x):
    return (float(1) / ( 1 + x * x ))

# Function to perform calculations
def calculate(lower_limit, upper_limit, interval_limit ):
     # Write your code here
     …



# driver function
interval_limit = 10
lower_limit = 1
upper_limit = 10

integral_res = calculate(lower_limit, upper_limit, interval_limit)

# rounding the final answer to 6 decimal places
print (round(integral_res, 6))
```

13. Ordinary Differential Equations

13.1 The Euler Method

Given a differential equation dy/dx = f(x, y) with initial condition y(x0) = y0. Find its approximate solution using Euler method.

Euler Method:

In mathematics and computational science, the Euler method (also called forward Euler method) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value.

Consider a differential equation dy/dx = f(x, y) with initial condition y(x0) = y0
then a successive approximation of this equation can be given by:

y(n+1) = y(n) + h * f(x(n), y(n))
where h = (x(n) – x(0)) / n, h indicates step size. Choosing smaller values of h leads to more accurate results and more computation time.

Example:
Consider below differential equation dy/dx = (x + y + xy) with initial condition y(0) = 1 and step size h = 0.025. Find y(0.1).

Solution:
f(x, y) = (x + y + xy)
x0 = 0, y0 = 1, h = 0.025
Now we can calculate y1 using Euler formula
y1 = y0 + h * f(x0, y0)
y1 = 1 + 0.025 *(0 + 1 + 0 * 1)
y1 = 1.025
y(0.025) = 1.025.
Similarly we can calculate y(0.050), y(0.075), ...y(0.1).
y(0.1) = 1.11167

Hints:

```
# Find approximation of an ordinary differential equation using Euler method.

# Consider a differential equation
# dy / dx =(x + y + xy)
def func( x, y ):
    return (x + y + x * y)

# Function for Euler formula
def euler( x0, y, h, x ):
    # write code here
    …

# Driver Code
# Initial Values
x0 = 0
y0 = 1
h = 0.025

# Value of x at which we need approximation
x = 0.1
euler(x0, y0, h, x)
```

13.2 Runge-Kutta Second Order Method

Given the following inputs:

1. An ordinary differential equation that defines the value of **dy/dx** in the form **x** and **y**.

$$\frac{dy}{dx} = f(x, y)$$

2. Initial value of y, i.e., **y(0)**.

$$y(0) = y_o$$

The task is to find the value of unknown function y at a given point x, i.e. **y(x)**.
Input: x0 = 0, y0 = 1, x = 2, h = 0.2

Output: y(x) = 0.645590

Input: x0 = 2, y0 = 1, x = 4, h = 0.4;

Output: y(x) = 4.122991

Approach:
The Runge-Kutta method finds an approximate value of y for a given x. Only first-order ordinary differential equations can be solved by using the Runge-Kutta 2nd-order method.
Below is the formula used to compute the next value yn+1 from the previous value yn. Therefore:
y_{n+1} = value of y at (x = n + 1)
y_n = value of y at (x = n)
where 0 ? n ? (x - x_0)/h, h is step height
x_{n+1} = x_0 + h
The essential formula to compute the value of y(n+1):

K1 = h * f(x, y)

K2 = h * f(x/2, y/2) or K1/2

y_{n+1} = y_n + K^2 + (h^3)

The formula basically computes the next value $\mathbf{y_{n+1}}$ using current $\mathbf{y_n}$ plus the weighted average of two increments:

- **K1** is the increment based on the slope at the beginning of the interval, using y.
- **K2** is the increment based on the slope at the midpoint of the interval, using **(y + h*K1/2)**.

Hints:

```
# Implement Runge-Kutta method

# A sample differential equation
# "dy/dx = (x - y)/2"

def dydx(x, y):
    return (x + y - 2)

# Finds value of y for a given x using step size h and initial value y0 at x0.
def rungeKutta(x0, y0, x, h):
    # write code here
    …

# Driver Code
x0 = 0
y = 1
x = 2
h = 0.2
print("y(x) =", rungeKutta(x0, y, x, h))
```

14. Final Notes

The only way to learn programming is program, program and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests (such as International Collegiate Programming Contest (ICPC), International Olympiad in Informatics (IOI)). Check out these sites:

- The ACM - ICPC International collegiate programming contest (https://icpc.global/)
- The Topcoder Open (TCO) annual programming and design contest (https://www.topcoder.com/)
- Universidad de Valladolid's online judge (https://uva.onlinejudge.org/).
- Peking University's online judge (http://poj.org/).
- USA Computing Olympiad (USACO) Training Program @ http://train.usaco.org/usacogate.
- Google's coding competitions (https://codingcompetitions.withgoogle.com/codejam, https://codingcompetitions.withgoogle.com/hashcode)
- The ICFP programming contest (https://www.icfpconference.org/)
- BME International 24-hours programming contest (https://www.challenge24.org/)
- The International Obfuscated C Code Contest (https://www0.us.ioccc.org/main.html)
- Internet Problem Solving Contest (https://ipsc.ksp.sk/)
- Microsoft Imagine Cup (https://imaginecup.microsoft.com/en-us)
- Hewlett Packard Enterprise (HPE) Codewars (https://hpecodewars.org/)
- OpenChallenge (https://www.openchallenge.org/)

Coding Contests Scores

Students must solve problems and attain scores in the following coding contests:

Name of the contest	Minimum number of problems to solve	Required score
CodeChef	20	200
Leetcode	20	200
GeeksforGeeks	20	200
SPOJ	5	50
InterviewBit	10	1000
Hackerrank	25	250
Codeforces	10	100
BuildIT	50	500
	Total score need to obtain	2500

Student must have any one of the following certification:

1. HackerRank - Problem Solving Skills Certification (Basic and Intermediate)
2. GeeksforGeeks – Data Structures and Algorithms Certification
3. CodeChef - Learn Python Certification
4. Interviewbit – DSA pro / Python pro
5. NPTEL – Programming, Data Structures and Algorithms
6. NPTEL – The Joy of Computing using Python

V. TEXT BOOKS:

1. Eric Matthes, "Python Crash Course: A Hands-On, Project-based Introduction to Programming", No Starch Press, 3rd Edition, 2023.
2. John M Zelle, "Python Programming: An Introduction to Computer Science", Ingram short title, 3rd Edition, 2016.

VI. REFERENCE BOOKS:

1. Yashavant Kanetkar, Aditya Kanetkar, "Let Us Python", BPB Publications, 2nd Edition, 2019.
2. Martin C. Brown, "Python: The Complete Reference", Mc. Graw Hill, Indian Edition, 2018.
3. Paul Barry, "Head First Python: A Brain-Friendly Guide", O'Reilly, 2nd Edition, 2016
4. Taneja Sheetal, Kumar Naveen, "Python Programming – A Modular Approach", Pearson, 1st Edition, 2017.
5. R Nageswar Rao, "Core Python Programming", Dreamtech Press, 2018.

VII. ELECTRONICS RESOURCES

8. https://realPython.com/Python3-object-oriented-programming/
9. https://Python.swaroopch.com/oop.html

10. https://Python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
11. https://www.programiz.com/Python-programming/
12. https://www.geeksforgeeks.org/python-programming-language/

VIII. MATERIALS ONLINE

1. Course template
2. Lab Manual

COURSE CONTENT

ENGINEERING WORKSHOP

I Semester: AE / CE / ME / ECE / EEE / CSE (AI&ML) / CSE (DS)
II Semester: CSE / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
AMEE02	Foundation	0	0	2	1	40	60	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 45				Total Classes:45		
Prerequisite: There is no prerequisite for this course.								

I. COURSE OVERVIEW:

This course provides the opportunity to become confident with new tools, equipment, and techniques for creating physical objects and mechanisms with a variety of materials. The students will learn principles of contemporary trends in manufacturing processes, such as CNC machining and 3D printing, as well as gain practical experience in carpentry, fitting, and welding. Skills learned in the course enable the students to learn about the design process in digital manufacturing used in various industrial applications.

II. COURSES OBJECTIVES:

The students will try to learn

I. The basics and hands-on practice of carpentry, fitting, and welding
II. The impart knowledge and skill to use tools, equipment, measuring instruments, and modern techniques.
III. The concepts apply to the manufacturing processes of casting, moulding and forging.
IV. The basic machining operations by CNC lathe, CNC milling, and 3D printing machine.

III. COURSE OUTCOMES:

At the end of the course students should be able to:

CO 1 Select appropriate tools, work material and measuring instruments useful for carpentry, fitting, and welding.
CO 2 Use flat sheets for sheet metal and intricate shapes made from mild steel for Black smithy.
CO 3 Choose appropriate components and tools to prepare pipe fitting and joints of specific shapes and sizes.
CO 4 Experiment with the moulding techniques for producing cast components in complex shapes using different patterns.
CO 5 Execute hard soldering techniques to join similar and dissimilar materials used in industries.
CO 6 Demonstrate appropriate equipment and methods for various machining processes used in CNC machines and 3D printing for manufacturing industries.

IV. COURSE CONTENT:

EXERCISES IN ENGINEERING WORKSHOP

Note: All dimensions are in mm in experiments.

Getting started experiments

Introduction

Engineering workshop provides both tools and equipments (or machinery) that are required for the manufacture of the goods. Students are familiarized with basic workshop practice like Wood working, Sheet metal, metal joining processes, manufacturing processes etc. and required to identify, operate and control various machines, tools and equipments.

Safety

Safety is a vital issue in all workplaces. Before using any equipment and machines or attempt practical work in a workshop everyone must understand basic safety rules. These rules will help keep all safe in the workshop.

Safety Rules:

- Always listen carefully to the teacher and follow instructions.
- When learning how to use a machine, listen very carefully to all the instructions given by the faculty / instructor. Ask questions, especially if you do not fully understand.
- Always wear an apron as it will protect your clothes and holds lose clothing such as ties in place.
- Bags should not be brought into a workshop as people can trip over them.
- Do not use a machine if you have not been shown how to operate it safely by the faculty / instructors
- Know where the emergency stop buttons are positioned in the workshop. If you see an accident at the other side of the workshop you can use the emergency stop button to turn off all electrical power to machines.
- Wherever required, wear protective equipment, such as goggles, safety glasses, masks, gloves, hair nets, etc.
- Always be patient, never rush in the workshop.
- Always use a guard when working on a machine.
- Keep hands away from moving/rotating machinery.
- Use hand tools carefully, keeping both hands behind the cutting edge.
- Report any UNSAFE condition or acts to instructor.
- Report any damage to machines/equipment as this could cause an accident.
- Keep your work area clean.

DO’s

- Students must always wear uniform and shoes before entering the lab.
- Proper code of conduct and ethics must be followed in the lab.
- Note down the specifications/drawings before working on the preparation of models.

- Receive the tools and materials required for preparation of models with signing in register.
- Properly fix hacksaw blade in frame with help of instructor.
- Use of safety goggles / face shield during welding.
- Do the models under the supervision/guidance of a faculty/ lab instructor only.
- Keep the sufficient distance from other students while preparing models.
- In case of fire use fire extinguisher/throw the sand provided in the lab.
- In case of any physical injuries or emergencies use first aid box provided.

DONT's

- Do not touch electrical circuits of welding machine.
- Be cautious while fixing hacksaw blade in frame, that may cause injuries to hand.
- Don't touch /operate power tools without aid from instructors.
- Don't gather while preparing models, that may hurt other with tools.
- Don't unlock snip/sheet metal cutter lock, without use.

1. Introduction to carpentry

Carpentry is the process of shaping wood, using hand tools. The products produced are used in building construction, such as doors and windows, furniture manufacturing, patterns for mouldin g infoundries, etc. Carpentry work mainly involves the joining together of wooden pieces and finishing the surfaces after shaping them. Hence, the term joining is also used commonly for carpentry. A student studying the fundamentals of wood working has to know about timber and other carpentry materials, wood working tools, carpentry operations and the method of making common types of joints.

1.1. Experiments on carpentry

1. Preparation of the cross-half lap joint as shown in Fig. 1.1

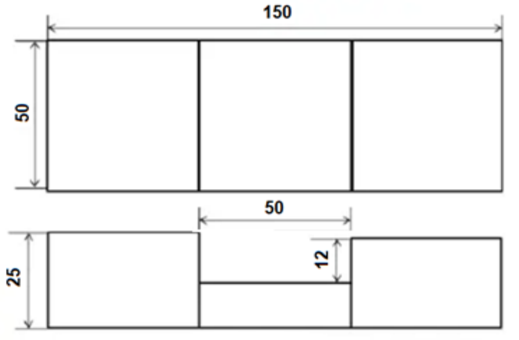


Fig.1.1 Dimensions of pieces

2. Preparation of the dove tail joint as depicted in Fig.1.2

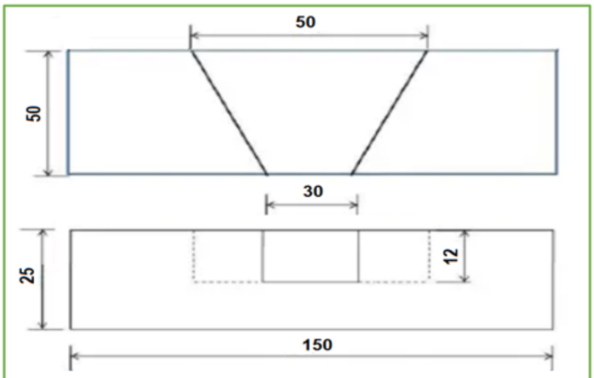


Fig.1.2 Dimensions of pieces

Try

1. Mortise and tenon joint preparation as illustrated in Fig.1.3 with dimensions of width = 50 mm and tenon thickness = 10 mm.
2. End lap joint preparation as illustrated in Fig. 1.4. The end lap projection dimensions to be taken into consideration are width = 50 mm and thickness = 15 mm.

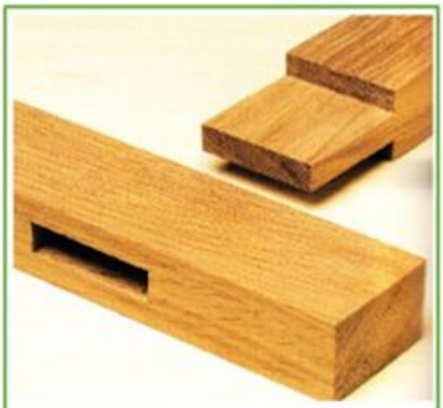


Fig.1.3 Mortise and tenon joint

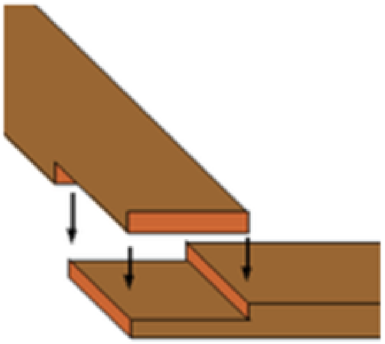


Fig.1.4 End lap joint

2. Introduction to fitting

The term fitting, is related to assembly of parts, after bringing the dimension or shape to the required size or form, in order to secure the necessary fit. The operations required for the same are usually carried out on a work bench, hence the term bench work is also added with the name fitting. The bench work and fitting play an important role in engineering. Although in today's industries most of the work is done by automatic machines which produces the jobs with good accuracy but still it(job) requires some hand operations called fitting operations.

2.1. Experiments on fitting

1. Making of a square fitting using mild steel plates of the specified size, as shown in Fig. 2.1
2. Making of a V-fit according to the size of the provided mild steel plates, as shown in Fig. 2.2

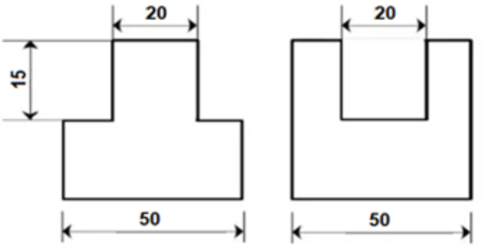


Fig.2.1 Dimensions of mild steel plates

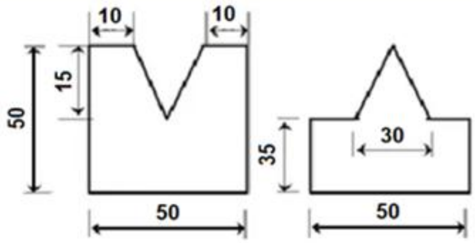


Fig.2.2 Dimensions of mild steel plates

Try

1. Straight fitting of mild steel plates to the specified sizes, as shown in Fig. 2.3
2. Making of semicircular fit with mild steel plates of the specified size, as depicted in Fig. 2.4

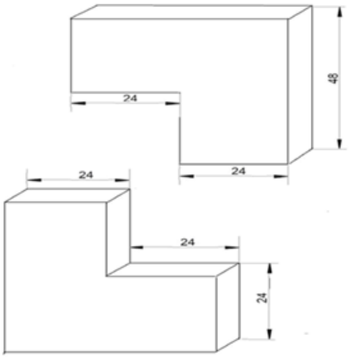


Fig.2.3 Dimensions of mild steel plates

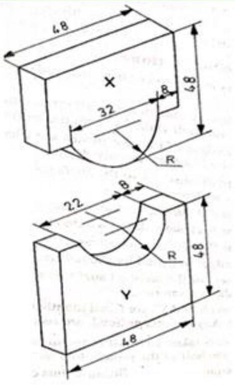


Fig.2.4 Dimensions of mild steel plates

3. Introduction to welding

Welding is a process for joining two similar or dissimilar metals by fusion. It joins different metals/alloys, with or without the application of pressure and with or without the use of filler metal. The fusion of metal takes place by means of heat. The heat may be generated either from combustion of gases, electric arc, electric resistance or by chemical reaction. Welding provides a permanent joint but it normally affects the metallurgy of the components. It is therefore usually accompanied by post weld heat treatment for most of the critical components. The welding is widely used as a fabrication and repairing process in industries. Some of the typical applications of welding include the fabrication of ships, pressure vessels, automobile bodies, off-shore platform, bridges, welded pipes, sealing of nuclear fuel and explosives, etc.

3.1. Experiments and demonstration on different welding techniques

1. Creating the lap joint in accordance with the mild steel plates given, as shown in Fig .3.1
2. Making the butt joint as depicted in Fig. 3.2 using the mild steel plates as are offered.

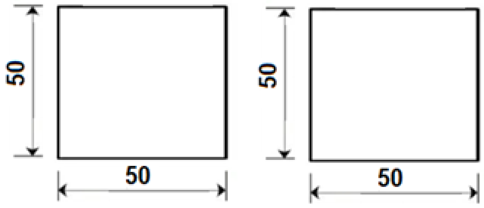


Fig.3.1 Dimensions of mild steel plates

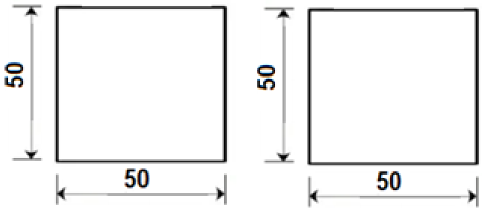


Fig.3.2 Dimensions of mild steel plates

Try

1. Construction of the tee joint using the mild steel plates provided, as shown in Fig. 3.3
2. As illustrated in Fig. 3.4, creating the corner (L) joint using the provided mild steel plates.

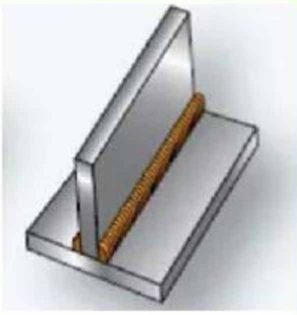


Fig.3.3 Tee joint

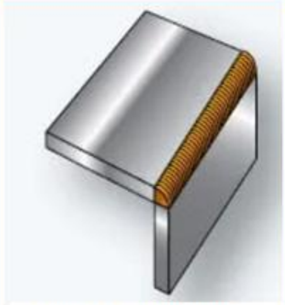


Fig.3.4 Corner joint

4. Introduction to sheet metal

Sheet metal work has its own significance in the engineering work. Many products, which fulfill the household needs, decoration work and various engineering articles, are produced from sheet metals. Common examples of sheet metal work are hoopers, canisters, guards, covers, pipes, hoods, funnels, bends, boxes etc. Such articles are found less expensive, lighter in weight and in some cases sheet metal products replace the use of castings or forgings.

4.1. Experiments on sheet metal forming

1. Create the rectangular tray as depicted in Fig. 4.1.

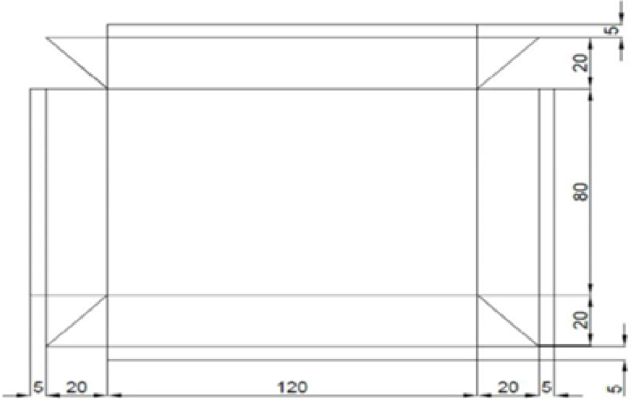


Fig.4.1 Dimensions of GI sheet

2. As illustrated in Fig.4.2, prepare the developing surface and create cylindrical tin.

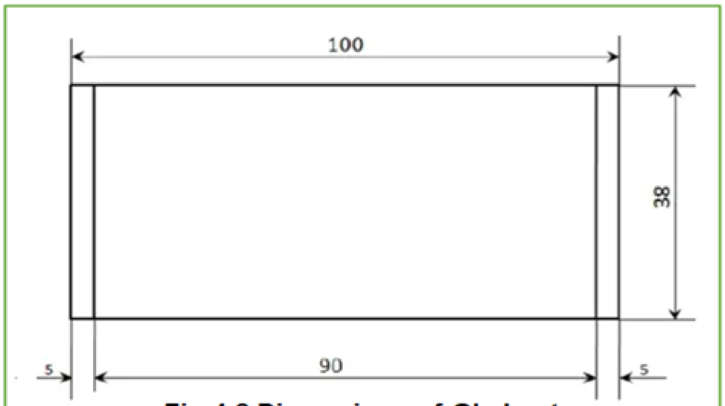


Fig.4.2 Dimensions of GI sheet

Try

1. Construct the open scoop as depicted in Fig. 4.3

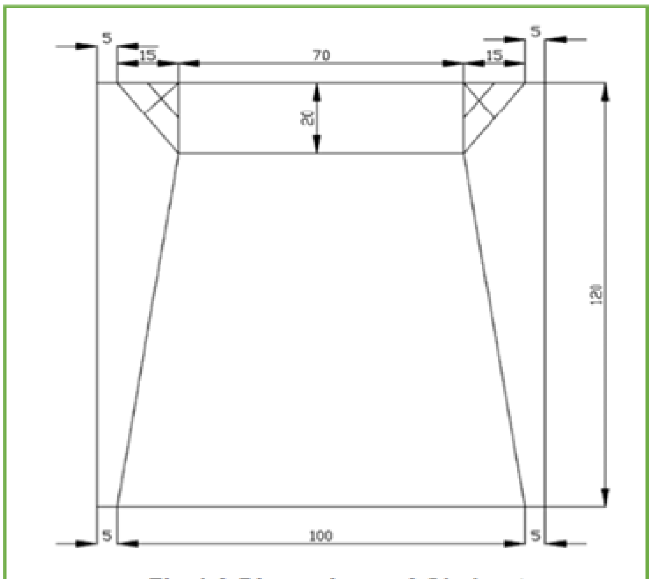


Fig.4.3 Dimensions of GI sheet

2. create the hexagonal prism as shown in Fig.4.4

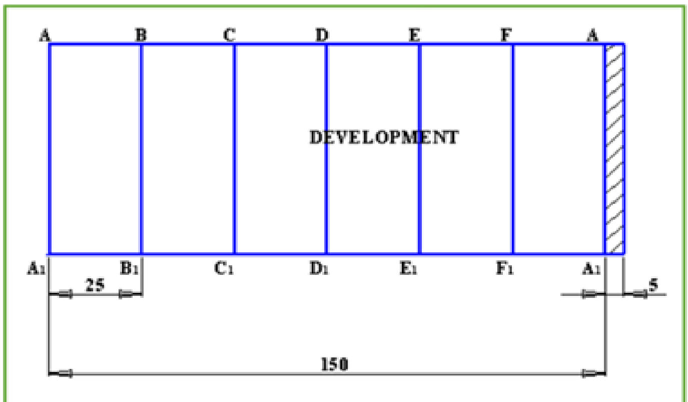


Fig.4.4 Surface development of hexagonal prism

5. Introduction to black smithy

Black smithy or Forging is an oldest shaping process used for the producing small articles for which accuracy in size is not so important. The parts are shaped by heating them in an open fire or hearth by the blacksmith and shaping them through applying compressive forces using hammer. Thus forging is defined as the plastic deformation of metals at elevated temperatures into a predetermined size or shape using compressive forces exerted through some means of hand hammers, small power hammers, die, press or upsetting machine. It consists essentially of changing or altering the shape and section of metal by hammering at a temperature of about 980°C, at which the metal is entirely plastic and can be easily deformed or shaped under pressure. The shop in which the various forging operations are carried out is known as the smithy or smith's shop.

5.1. Experiments on black smithy

1. Make the s-hook as depicted in Fig. 5.1 using the mild steel rod provided.
2. Construct the J-hook using the given mild steel rod as indicated in Fig. 5.2.

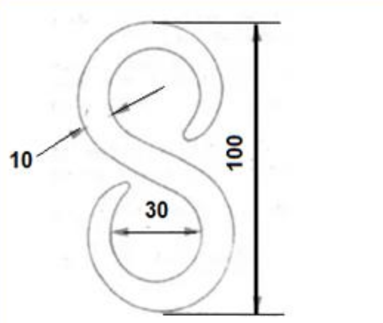


Fig.5.1 Dimensions of S-hook

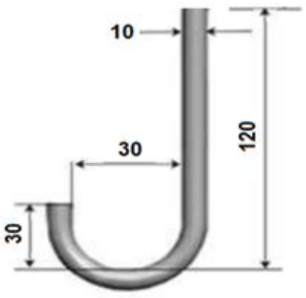


Fig.5.2 Dimensions of J-hook

Try

1 Create the C - hook with the given mild steel rod as shown in Fig. 5.3
2 Prepare the U - bend with the given mild steel rod as shown in Fig. 5.4

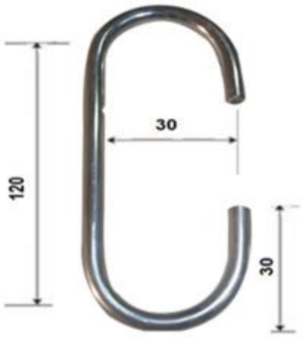


Fig.5.3 Dimensions of C-hook

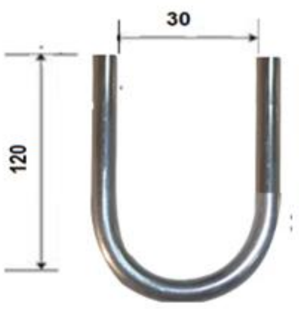


Fig.5.4 Dimensions of U-hook

6. Introduction to plumbing

Plumbing is a skilled trade of working with pipes or tubes and plumbing fixtures. The process is mainly used for the supply of drinking water and the drainage of waste water, sometimes mixed with waste floating materials in a living or working place. A plumber is someone who installs or repairs piping systems, plumbing fixtures and equipment such as valves, washbasins, water heaters, wat erclosests, etc. Thus it usually refers to a system of pipes and fixtures installed in a building for the distribution of water and the removal of waterborne wastes.

6.1. Experiments and demonstration on plumbing

1.Form of PVC pipe fitting through various components as shown in Fig. 6.1

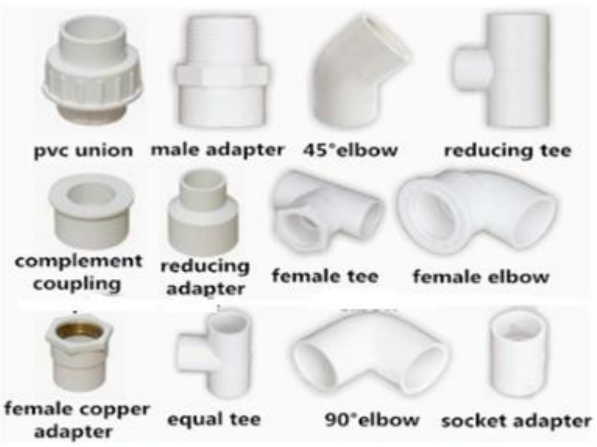


Fig.6.1 PVC components for pipe fittings

2. Form of GI pipe fitting with various components, as shown in Fig. 6.2

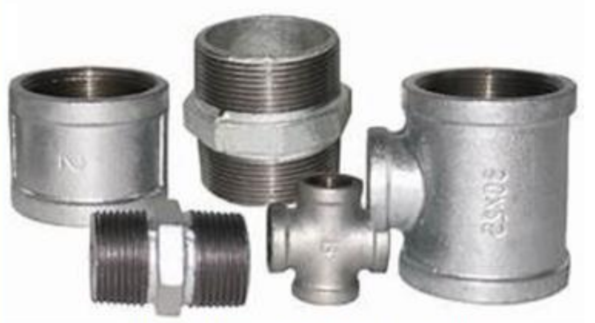


Fig.6.2 GI componets for pipe fittings

Try

1. Form of PVC pipe fitting with reducer for water tap with different components as shown in Fig. 6.1
2. Form of GI pipe fitting with different components as shown in Fig. 6.2 for different fluids.

7. Introduction to moulding

Moulding is the process of manufacturing by shaping liquid or pliable raw material using a rigid frame called a mold or matrix. This itself may have been made using a pattern or model of the final object.

A mould is a hollowed-out block that is filled with a liquid or pliable material such as plastic, glass, metal, or ceramic raw material. The liquid hardens or sets inside the mould, adopting its shape. A mould is a counterpart to a cast. The very common bi-valve moulding process uses two moulds, one for each half of the object.

7.1. Experiments on mechanical components moulding (casting process)

1. Making of flange mould using a given pattern as shown in Fig.7.1

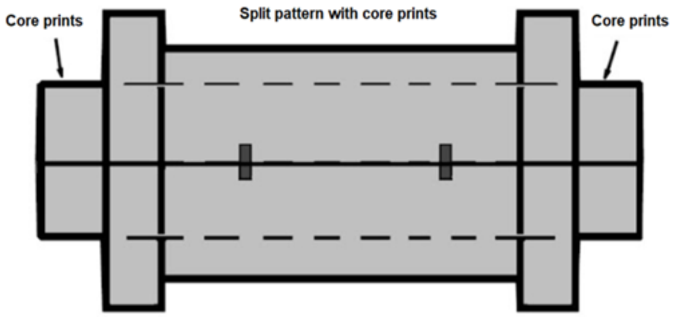


Fig.7.1 Flange coupling pattern

2. Utilizing the provided pattern, create the bearing housing mould as shown in Fig. 7.2.

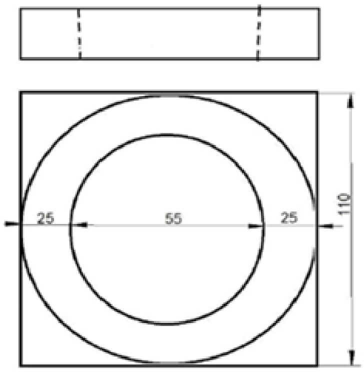


Fig.7.2 Bearing housing

Try

1. Making of dumble using a given pattern as shown in Fig.7.3
2. Using a single piece pattern, create a one-stepped shaft as shown in Fig. 7.4.

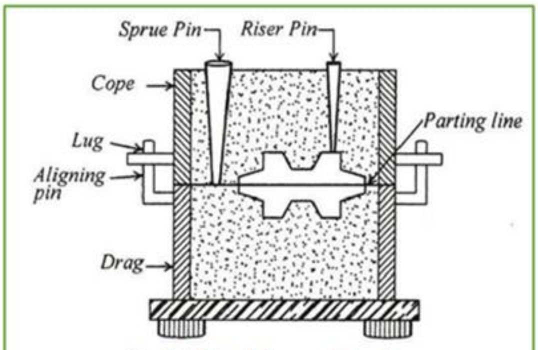


Fig.7.3 Dumble mould pattern

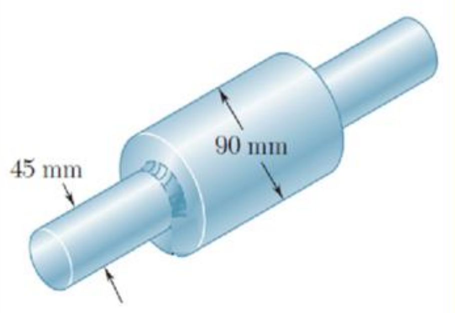


Fig.7.4 Single stpeped shaft

8. Introduction to concrete moulding and plaster of paris

Concrete is characterized by the type of aggregate or cement used, by the specific qualities it manifests, or by the methods used to produce it. In ordinary structural concrete, the character of the concrete is largely determined by a water-to-cement ratio. The lower the water content, all else being equal, the stronger the concrete. The mixture must have just enough water to ensure that each aggregate particle is completely surrounded by the cement paste, that the spaces between the aggregate are filled, and that the concrete is liquid enough to be poured and spread effectively.

Plaster of Paris is a white powder made from gypsum that mixes with water to form a paste that hardens quickly and is used chiefly for casts and moulds. It can be effectively worked with metal apparatuses or even abrasive sheets and can be shaped as per requirements. It is often applied in the form of a quick-setting paste with water.

8.1.Experiment on concrete/cement cube moulding and demonstration on plaster of paris mould making

1. Preparation of concrete cube by moulding technique as shown in Fig.8.1

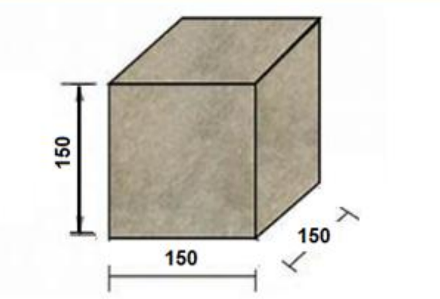


Fig.8.1 Dimensions of concrete cube

2. Demonstration on plaster of paris mould making.

Try

1. Preparation of any house hold specimens by plaster of paris mould making as shown in Fig. 8.2
2. Preparation of any intricate article by plaster of paris mould making as shown in Fig. 8.3

Fig.8.2 Plaster of paris House hold specimens

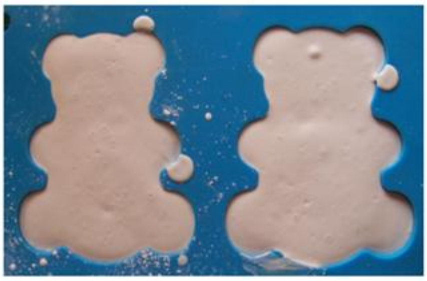


Fig.8.3 Plaster of paris intricate article

9. Introduction to hard soldering

Hard (silver) soldering (>450 °C) – Brass or silver is the bonding metal used in this process, and requires a blowtorch to achieve the temperatures at which the solder metals. Hard soldering is used to join precision components such as ferrous, brass, and copper.

9.1. Experiments on hard soldering

1. Soldering of two mild steel plates as shown in Fig. 9.1
2. Hard soldering of engine valve tappet as shown in Fig. 9.2

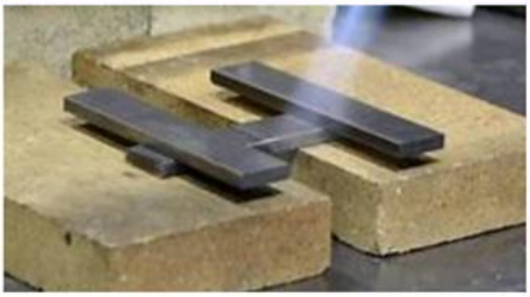


Fig.9.1 Soldering of mild steel plates

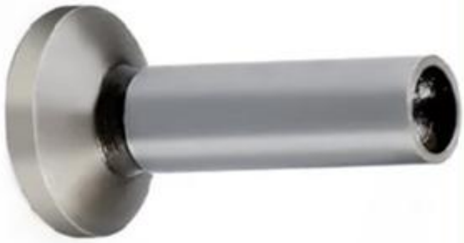


Fig.9.2 Engine valve tappet

Try

1. Hard soldering of copper with brass as shown in Fig.9.3
2. Hard soldering of stainless steel with brass as shown in Fig.9.4

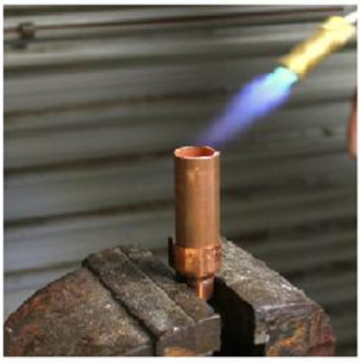


Fig.9.3 Hard soldering of copper with brass

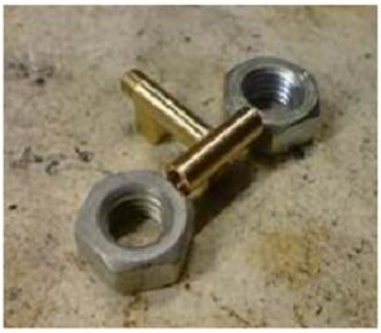


Fig.9.4 Hard soldering of stainless steel with brass

10. Demonstration on Computer Numerically Controlled (CNC) lathe

CNC turning is a highly precise and efficient subtractive machining process that works on the principle of the lathe machine. It involves placing the cutting tool against a turning workpiece to remove materials and give the desired shape.

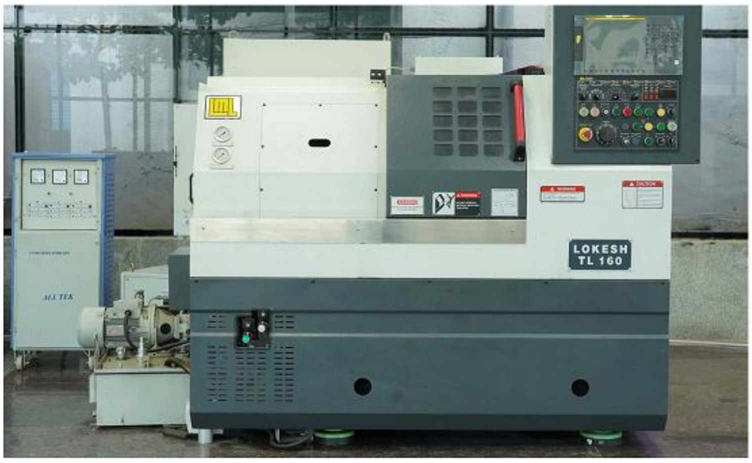


Fig.10.1 CNC lathe

1. Demonstration of the plain turning process on a CNC lathe as shown in Fg.10.1
2. Demonstration of facing operations on a CNC lathe as shown in Fg.10.1.

11. Demonstration on Computer Numerically Controlled (CNC) milling

CNC milling involves cutting a prismatic workpiece using multipoint cutting tools producing precision components used in automotive and aeronautical industries.

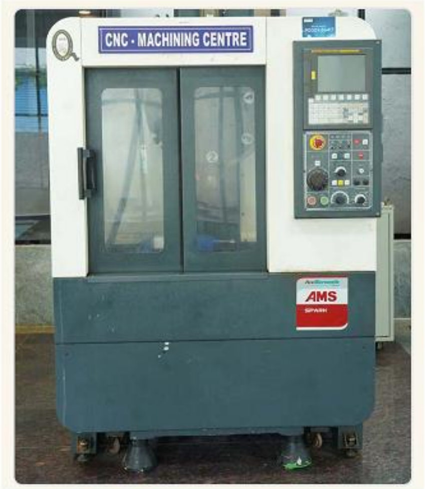


Fig.11.1 CNC machining centre

1. Demonstration of plain milling (facing) on CNC milling as shown in Fig.11.1
2. Demonstration of precision slotting on CNC milling as shown in Fig.11.1.

12. Demonstration on 3D printing machine

3D printing or additive manufacturing enables to produce geometrically complex objects, shapes and textures. It often uses less material than traditional manufacturing methods and allows the production of prototypes / products that are not possible to produce economically with conventional manufacturing.

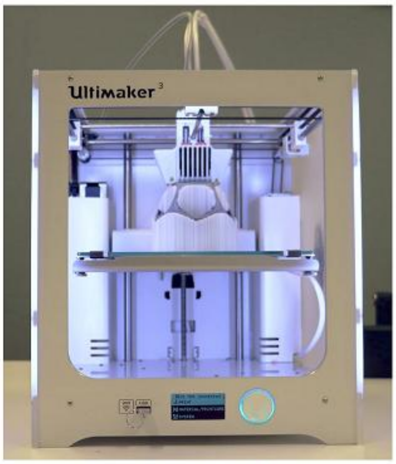


Fig.12.1 3D printer

1. Demonstration of 3D printing machine as shown in Fig.12.1 using Acrylonitrile butadiene styrene (ABS) material
2. Demonstration of 3D printing machine as shown in Fig.12.1 using Polylactic acid (PLA) material.

13. Demonstration on 6- axis robot

Robots have seen in recent years an expansion of their field of use with new requirements related to the increasing use of composites. The robots are then considered for machining operations (polishing, cutting, drilling etc.) that require high performance in terms of position, orientation, followed by trajectory precision and stiffness. The evolution of the performance of robots and programming software provides new machining solutions. For complex parts, six axis robots offer more accessibility than a machining center CNC 5 axis and allow the integration of additional axes to extend the workspace.

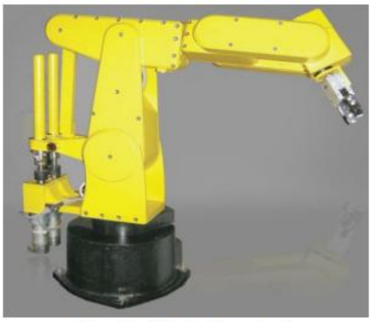


Fig.13.1 6-Axis robot

1. Demonstration of the 6 – axis aristo robot as shown in Fig.13.1.
2. Demonstration of aristo sim software for robot movements and control.

14. Demonstration on cylindrical grinding machine

Most commonly, cylindrical grinding is used for grinding pieces with a central axis of rotation, like rods and cylinders. This process involves using a cylindrical grinder, which is a type of machinery categorized by rotation style and wheel device.

A grinding machine uses an abrasive product usually a rotating wheel to shape and finish a workpiece by removing metal and generating a surface within a given tolerance. A grinding wheel is made with abrasive grains bonded together. Each grain acts as a cutting tool, removing tiny chips from the workpiece.

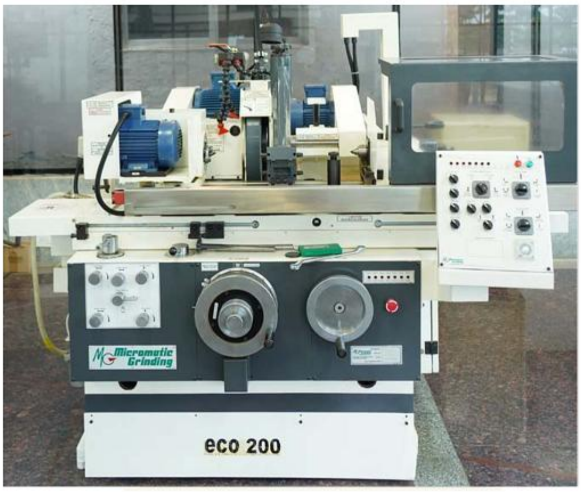


Fig.14.1 Cylindrical grinding machine

1. Demonstration of grinding process on a cylindrical grinding machine as shown in Fig.14.1
2. Demonstration of shaft grinding process on a cylindrical grinding machine as shown in Fig.14.1

V. TEXT BOOKS:

1. Hajra Choudhury S.K., Hajra Choudhury A.K. and NirjharRoy S.K., *"Elements of Workshop Technology"*, Media promoters and publishers private limited, Mumbai, 4th Edition ,2020.
2. Kalpakjian S, Steven S. Schmid, *"Manufacturing Engineering and Technology"*, Pearson Education India Edition, 7th Edition, 2019.
3. Gowri P. Hariharan, A. Suresh Babu," *Manufacturing Technology – I*", Pearson Education, 3rd Edition, 2018.

VI. REFERENCE BOOKS:

1. Gowri P. Hariharan, A. Suresh Babu, *"Manufacturing Technology – I"*, Pearson Education, 5th Edition, 2018.
2. Roy A. Lindberg, *"Processes and Materials of Manufacture",* Prentice Hall India, 4th Edition, 2017.
3. Rao P.N., "*Manufacturing Technology"*, Vol. I and Vol. II, Tata McGraw-Hill House, 2017.

VII. ELECTRONICS RESOURCES:

1. https://elearn.nptel.ac.in/shop/iit-workshops/ongoing/additive-manufacturing-technologies-for-practicing-engineers/.
2. https://akanksha.iare.ac.in/index?route=course/details&course_id=337

VIII. MATERIALS ONLINE:

3. Course Template
4. Laboratory manual

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

UNDERTAKING BY STUDENT / PARENT

“To make the students attend the classes regularly from the first day of starting of classes and be aware of the College regulations, the following undertaking form is introduced which should be signed by both student and parent. The same should be submitted to the Dean of Academic”.

I, Mr. / Ms. -- joining I Semester / III Semester for the academic year 20 - 20 / 20 - 20 in Institute of Aeronautical Engineering, Hyderabad, do hereby undertake and abide by the following terms, and I will bring the ACKNOWLEDGEMENT duly signed by me and my parent and submit it to the Dean of Academic.

1. I will attend all the classes as per the timetable from the starting day of the semester specified in the institute Academic Calendar. In case, I do not turn up even after two weeks of starting of classes, I shall be ineligible to continue for the current academic year.
2. I will be regular and punctual to all the classes (theory/laboratory/project) and secure attendance of not less than 75% in every course as stipulated by Institute. I am fully aware that an attendance of less than 65% in more than 60% of theory courses in a semester will make me lose one year.
3. I will compulsorily follow the dress code prescribed by the college.
4. I will conduct myself in a highly disciplined and decent manner both inside the classroom and on campus, failing which suitable action may be taken against me as per the rules and regulations of the institute.
5. I will concentrate on my studies without wasting time in the Campus/Hostel/Residence and attend all the tests to secure more than the minimum prescribed Class/Sessional Marks in each course. I will submit the assignments given in time to improve my performance.
6. I will not use Mobile Phone in the institute premises and also, I will not involve in any form of ragging inside or outside the campus. I am fully aware that using mobile phone to the institute premises is not permissible and involving in Ragging is an offence and punishable as per JNTUH/UGC rules and the law.
7. I declare that I shall not indulge in ragging, eve-teasing, smoking, consuming alcohol drug abuse or any other anti-social activity in the college premises, hostel, on educational tours, industrial visits or elsewhere.
8. I will pay tuition fees, examination fees and any other dues within the stipulated time as required by the Institution / authorities, failing which I will not be permitted to attend the classes.
9. I will not cause or involve in any sort of violence or disturbance both within and outside the college campus.
10. If I absent myself continuously for 3 days, my parents will have to meet the HOD concerned / Principal.
11. I hereby acknowledge that I have received a copy of BT23 Academic Rules and Regulations, course catalogue and syllabus copy and hence, I shall abide by all the rules specified in it.

ACKNOWLEDGEMENT

I have carefully gone through the terms of the undertaking mentioned above and I understand that following these are for my/his/her own benefit and improvement. I also understand that if I/he/she fail to comply with these terms, shall be liable for suitable action as per Institute/JNTUH/AICTE/UGC rules and the law. I undertake that I/he/she will strictly follow the above terms.

Signature of Student with Date

Signature of Parent with Date
Name & Address with Phone Number