ADVANCED PYTHON PROGRAMMING LABORATORY

III Semester: Common for CSE / CSE (AI&ML) / CSE (DS), IT, CSIT								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
ACSC11	Core	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	1	2	2	40	60	100
Contact Classes: Nil	**Tutorial Classes: 15**	**Practical Classes: 30**				**Total Classes: 45**		
Prerequisites: There are no prerequisites to take this course.								

I. COURSE OVERVIEW:

The course aims to focuses on providing practical experience and in-depth knowledge of advanced topics and techniques in python programming. The course is designed to build the core concepts of python programming and extend skills to more complex and sophisticated applications. It emphasizes students to become proficient in advanced python programming techniques.

II. COURSE OBJECTIVES

The students will try to learn:

I. Advanced python applications using complex data structures and understanding of object-oriented programming principles effectively in real-world applications.
II. Concepts of exception and file handling and be able to apply them effectively in their programming projects, ensuring robust and error-tolerant software.
III. Efficient, responsive, and robust multi-threaded applications in Python to improve the performance.
IV. Data manipulation libraries such as NumPy, Pandas, and Matplotlib for data analysis and visualization.

III. COURSE OUTCOMES:

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

CO 1 Develop a basic understanding of advanced python concepts, syntax, and feature for solving compl problems.
CO 2 Develop proficiency in using various python libraries and frameworks for specific tasks, for da manipulation, visualization and web development.
CO 3 Implement robust error handling using exception mechanisms and advanced file handling techniques ensure code reliability.
CO 4 Demonstrate multi-threaded applications using python's threading and multiprocessing modules understand the challenges of concurrent programming.
CO 5 Gain experience in working with databases using Python's database API and learn to interact wi relational databases.
CO 6 Design and implement graphical user interfaces (GUIs) using Python GUI libraries to devel interactive applications with event-driven programming.

IV. COURSE CONTENT:

EXERCISES FOR ADVANCED PYTHON PROGRAMING LABORATORY

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

1. Getting Started Exercises

1.1 Valid Anagram

Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:
Input: s = "anagram", t = "nagaram"
Output: true

Example 2:
Input: s = "rat", t = "car"
Output: false

Hints:

```
class Solution {
public:
    bool isAnagram(string s, string t)
 {
          #Write the code here
 }
};
```

Try

1. Take input: s = "gram", t = "garam" and verify your results.
2. Take input: s = "bat", t = "cat" and verify your results.

1.2 The Climbing Stairs Puzzle

Imagine a staircase with n steps. As you are climbing up this staircase, you either climb one or two steps at a time. The aim of this computing puzzle is to find out, using an algorithm, in how many distinct ways can you climb to the top?

Hints:

```
#The climbing stairs puzzle
def countWays(n):
#Write the code here
steps = 10
ways = countWays(steps)
print("There are " + str(ways) + " distinct ways to climb a staircase of " + str(steps) + " steps when climbing up one or two steps at a time.")
```

Try
1. Take number of steps=15 and verify your results.
2. Take number of steps=50 and verify your results.

1.3 Longest Palindrome

Given a string s which consists of lowercase or uppercase letters, return the length of the longest palindrome that can be built with those letters.
Letters are case sensitive, for example, "Aa" is not considered a palindrome here.

Example 1:
Input: s = "abccccdd"
Output: 7
Explanation: One longest palindrome that can be built is "dccaccd", whose length is 7.

Example 2:
Input: s = "a"
Output: 1
Explanation: The longest palindrome that can be built is "a", whose length is 1.

Hints:
```
class Solution {
public:
    int longestPalindrome(string s) {
        #Write the code here
    }
};
```

Try
1. Take input s= ”112322111223” and verify the results.
2. Take input s= ”xxyyxx” and verify the results.

1.4 Coin Change

You are given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money.
Return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

You may assume that you have an infinite number of each kind of coin.
Example 1:
Input: coins = [1, 2, 5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1

Example 2:
Input: coins = [2], amount = 3
Output: -1

Example 3:
Input: coins = [1], amount = 0
Output: 0

Hints:

```
class Solution(object):
    def coinChange(self, coins, amount):
        """
        # write the code here
          """
```

Try
1. Take input: coins = [3, 2, 2] and verify the results
2. Take input: coins = [1, 1, 1] and verify the results

1.5 Rotting Oranges

You are given an m x n grid where each cell can have one of three values:

- 0 representing an empty cell,
- 1 representing a fresh orange, or
- 2 representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.
Return *the minimum number of minutes that must elapse until no cell has a fresh orange*. If *this is impossible, return* -1.

Example 1:
Input: grid = [[2, 1, 1], [0, 1, 1], [1, 0, 1]]
Output: -1
Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Example 2:
Input: grid = [[0,2]]
Output: 0

Hints:

```
class Solution(object):
    def orangesRotting(self, grid):
        """
        :type grid: List[List[int]]
        :rtype: int
#Write the code here
```

Try
1. Take input: grid = [[2,1,1],[0,1,1],[1,0,1]] and verify the results.
2. Take input: grid = [[1,1,1],[1,0,1],[1,1,1]] and verify the results.
3. Take input: grid = [[1,2,1],[1,1,1],[1,1,0]] and verify the results.

1.6 Longest Palindromic Substring

Given a string s, return the longest palindromic substring in s.

Example 1:

Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"
Output: "bb"

Hints:

```
class Solution(object):
    def longestPalindrome(self, s):
        """
        :type s: str
        :rtype: str
        """
      # write code here
```

Try

1. Take input: s = "abbababbba" and verify the results.
2. Take input: s = "xyxxyyxxxyy" and verify the results.

1.7 Word Ladder

A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:
Every adjacent pair of words differs by a single letter.
Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
sk == endWord
Given two words, beginWord and endWord, and a dictionary wordList, return the number of words in the shortest transformation sequence from beginWord to endWord, or 0 if no such sequence exists.

Example 1:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.

Example 2:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList, therefore there is no valid transformation sequence.

Hints:

```
class Solution(object):
    def ladderLength(self, beginWord, endWord, wordList):
        :type beginWord: str
        :type endWord: str
        :type wordList: List[str]
        :rtype: int
```

```
            #Write the code here

        """
```

Try
1. Take input: beginWord = "hot", endWord = "cold" and verify results.
2. Take input: beginWord = "sun", endWord = "moon" and verify results.

1.8 Longest Palindromic Substring

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
```

Try
1. Take input nums = [10, 20, 30, 40] and verify the results.
2. Take input nums = [-10, 22, 36, 43] and verify the results.

1.9 Largest Rectangle in Histogram

Given an array of integer's heights representing the histogram's bar height where the width of each bar is 1, return the area of the largest rectangle in the histogram.

Hints:

```
class Solution(object):
    def largestRectangleArea(self, heights):
        :type heights: List[int]
        :rtype: int
#Write the code here
        """
```

Try
1. Write a program to find the smallest rectangle in histogram.

PROBING FOR FURTHER QUESTIONS

1. Write a function in Python to check duplicate letters. It must accept a string, i.e., a sentence. The function should return true if the sentence has any word with duplicate letters, else return False.

2. Write a code in Python to create a Morse code translator. You can take a string with alphanumeric characters in lower or upper case. The string can also have any special characters as a part of the Morse code. Special characters can include commas, colons, apostrophes, exclamation marks, periods, and question marks. The code should return the Morse code that is equivalent to the string.

3. Write a function to detect 13th Friday. The function can accept two parameters, and both will be numbers. The first parameter will be the number indicating the month, and the second will be the year in four digits. Your function should parse the parameters, and it must return True when the month contains a Friday with the 13th, else return False.

4. Write function to find the domain name from the IP address. The function will accept an IP address, make a DNS request, and return the domain name that maps to that IP address while using records of PTR DNS. You can import the Python socket library.

5. Write a function in Python to parse a string such that it accepts a parameter- an encoded string. This encoded string will contain a first name, last name, and an id. You can separate the values in the string by any number of zeros. The id will not contain any zeros. The function should return a Python dictionary with the first name, last name, and id values. For example, if the input would be "John000Doe000123". Then the function should return: { "first_name": "John", "last_name": "Doe", "id": "123" }

6. Write a function in Python to convert a decimal to a hex. It must accept a string of ASCII characters as input. The function should return the value of each character as a hexadecimal string. You have to separate each byte by a space and return all alpha hexadecimal characters as lowercase.

7. Write a program to find the largest and the smallest number and merge two sorted arrays in a given array.

2. Python Collection Exercise

2.1 Minimum Window Substring

Given two strings s and t of lengths m and n respectively, return *the* ***minimum window substring*** *of* s *such that every character in* t *(****including duplicates****) is included in the window*. If there is no such substring, return *the empty string* "".

Example 1:
Input: s = "ADOBECODEBANC", t = "ABC"
Output: "BANC"
Explanation: The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

Example 2:
Input: s = "a", t = "a"
Output: "a"
Explanation: The entire string s is the minimum window.

Example 3:
Input: s = "a", t = "aa"
Output: ""
Explanation: Both 'a's from t must be included in the window.

Since the largest window of s only has one 'a', return empty string.

Hints:

```
class Solution {
public:
    string minWindow(string s, string t) {

    }
};
```

Try

1. Take input: s = "AXYZPQRSSQTRT", t = "XYZ" and verify the results
2. Take input: s = "ABCDXYZXYZSQTRT", t = "ABC" and verify the results.

2.2 Best Time to Buy and Sell Stock

You are given an array price where prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

Example 1:

Input: prices = [7, 1, 5, 3, 6, 4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy before you sell.

Example 2:

Input: prices = [7, 6, 4, 3, 1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.

Hints:

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {

    }
};
```

Try

1. Take input prices = [6, 2, 4, 3, 6, 4] and verify the results.
2. Take input prices = [1, 1, 0, 3, 7, 9] and verify the results.

2.3 Strange Printer

There is a strange printer with the following two special properties:

- The printer can only print a sequence of the same character each time.
- At each turn, the printer can print new characters starting from and ending at any place and will cover the original existing characters.

Given a string s, return *the minimum number of turns the printer needed to print it*.

Example 1:
Input: s = "aaabbb"
Output: 2
Explanation: Print "aaa" first and then print "bbb".

Example 2:
Input: s = "aba"
Output: 2
Explanation: Print "aaa" first and then print "b" from the second place of the string, which will cover the existing character 'a'

Hints:
```
class Solution {
public:
    int strangePrinter(string s) {

    }
};
```

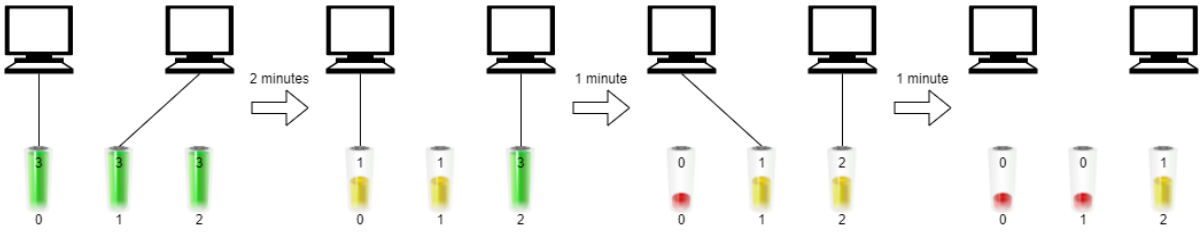
Try
1. Take Input: s = "xxxxyyyyzzzb" and verify the results
2. Take Input: s = "qqqssttrr" and verify the results

2.4 Maximum Running Time of N Computers

You have n computers. You are given the integer n and 0-indexed integer array batteries where the i^{th} battery can **run** a computer for batteries[i] minutes. You are interested in running **all** n computers **simultaneously** using the given batteries. Initially, you can insert **at most one battery** into each computer. After that and at any integer time moment, you can remove a battery from a computer and insert another battery **any number of times**. The inserted battery can be a totally new battery or a battery from another computer. You may assume that the removing and inserting processes take no time.

Note that the batteries cannot be recharged.
Return *the **maximum** number of minutes you can run all the* n *computers simultaneously.*

Example 1:



Input: n = 2, batteries = [3,3,3]
Output: 4

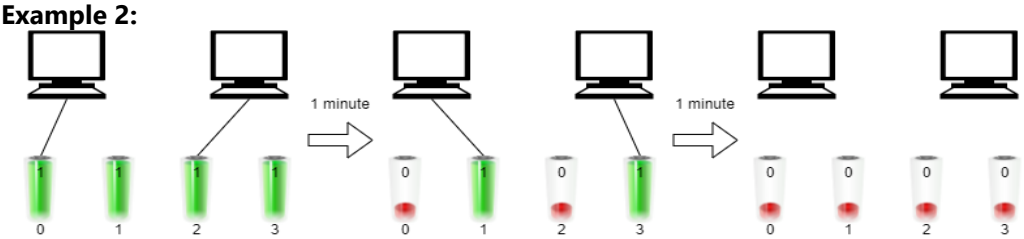
Explanation:
Initially, insert battery 0 into the first computer and battery 1 into the second computer.

After two minutes, remove battery 1 from the second computer and insert battery 2 instead. Note that battery 1 can still run for one minute.

At the end of the third minute, battery 0 is drained, and you need to remove it from the first computer and insert battery 1 instead.

By the end of the fourth minute, battery 1 is also drained, and the first computer is no longer running. We can run the two computers simultaneously for at most 4 minutes, so we return 4.

Example 2:



Input: n = 2, batteries = [1,1,1,1]
Output: 2

Explanation:
Initially, insert battery 0 into the first computer and battery 2 into the second computer.
After one minute, battery 0 and battery 2 are drained so you need to remove them and insert battery 1 into the first computer and battery 3 into the second computer.
After another minute, battery 1 and battery 3 are also drained so the first and second computers are no longer running. We can run the two computers simultaneously for at most 2 minutes, so we return 2.

Hints:

```
class Solution(object):
    def maxRunTime(self, n, batteries):
        """
        :type n: int
        :type batteries: List[int]
        :rtype: int
        """
```

Try
1. Take n = 4, batteries = [1, 1, 1, 1, 1, 1, 1] and verify the results
2. Take n = 4, batteries = [3, 3, 3, 3, 3] and verify the results.

2.5 Number of Music Playlists

Your music player contains n different songs. You want to listen to goal songs (not necessarily different) during your trip. To avoid boredom, you will create a playlist so that:

- Every song is played at least once.
- A song can only be played again only if k other songs have been played.

Given n, goals, and k, return *the number of possible playlists that you can create*. Since the answer can be very large, return it **modulo** $10^9 + 7$.

Example 1:
Input: n = 3, goal = 3, k = 1
Output: 6
Explanation: There are 6 possible playlists: [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], and [3, 2, 1].

Example 2:
Input: n = 2, goal = 3, k = 0
Output: 6
Explanation: There are 6 possible playlists: [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], and [1, 2, 2].

Example 3:
Input: n = 2, goal = 3, k = 1
Output: 2
Explanation: There are 2 possible playlists: [1, 2, 1] and [2, 1, 2].

Hints:

```
class Solution(object):
    def numMusicPlaylists(self, n, goal, k):
        """
        :type n: int
        :type goal: int
        :type k: int
        :rtype: int
        """
```

Try
1. Take Input: n = 4, goal = 4, k = 1 and verify the results
2. Take Input: n = 8, goal = 4, k = 2 and verify the results

2.6 Text Justification

Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified.
You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters.
Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right.
For the last line of text, it should be left-justified, and no extra space is inserted between words.
Note:

- A word is defined as a character sequence consisting of non-space characters only.
- Each word's length is guaranteed to be greater than 0 and not exceed maxWidth.
- The input array word contains at least one word.

Example 1:
Input: words = ["This", "is", "an", "example", "of", "text", "justification."], maxWidth = 16
Output:
[
  "This is an",
  "example of text",
  "justification. "
]

Example 2:
Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16
Output:
[
  "What must be",
  "acknowledgment ",
  "shall be   "
]
Explanation: Note that the last line is "shall be   " instead of "shall   be", because the last line must be left-justified instead of fully-justified.
Note that the second line is also left-justified because it contains only one word.

Hints:
```
class Solution(object):
    def fullJustify(self, words, maxWidth):
        """
        :type words: List[str]
        :type maxWidth: int
        :rtype: List[str]
        """
```

Try
1. Take Input: words = ["Science","is","what","we","understand","well","enough","to","explain","to","a","computer.","Art","is","everything","else","we","do"], maxWidth = 20 and verify the results.

2. Take Input: words = ["Science","is","what","we","understand","well","enough","to","explain","to","a","computer.","Art","is","everything","else","we","do"], maxWidth = 10 and verify the results

3. Python Date and Time Exercise

3.1 The Birthday Paradox

The birthday paradox is based on a counter-intuitive fact that in any class of 23 students or more, there is a higher probability of having at least two students sharing the same birthday.

Considering that there are 366 different possible dates in a year (leap year), you may first predict that it would take a group of 183 students (50% of 366) to reach a 50% probability of at least two students sharing the same birthday. However this prediction would be wrong as this is not how the probabilities work in this case. You can use this example to find out more how to work out that with a class size of only 23 students, there is a probability of 50% that at least two students sharing the same birthday.

Hints:
```
#The Birthday Paradox
import random
import datetime

#A function to generate a random date between two given dates
def randomDate(startDate, endDate):
…..
#List of 23 students in the class
classList =
["Opal","Hugo","Malek","Terrence","Jeremiah","Abdel","Sophie","Ethan","Noah","Jing
```

```
","Ines","Oceana","Diego","Zofia","Layla","Julian","Andrei","Ishan","Chloe","Mateo
","Omar","Jouri","Lily"]
birthdayList = []
```

Try

1. Take `classList ={“Ravi”,”salmon”,”Julia”,”Adbul”,”Rusie”} and verify the results.`

3.2 Minimum Seconds to Equalize a Circular Array

You are given a **0-indexed** array nums containing n integers.At each second, you perform the following operation on the array:

For every index i in the range [0, n - 1], replace nums[i] with either nums[i], nums[(i - 1 + n) % n], or nums[(i + 1) % n].

Note that all the elements get replaced simultaneously.

Return *the **minimum** number of seconds needed to make all elements in the array* nums *equal*.

Example 1:

Input: nums = [1,2,1,2]

Output: 1

Explanation: We can equalize the array in 1 second in the following way:

- At 1st second, replace values at each index with [nums[3],nums[1],nums[3],nums[3]]. After replacement, nums = [2,2,2,2].

It can be proven that 1 second is the minimum amount of seconds needed for equalizing the array.

Example 2:

Input: nums = [2,1,3,3,2]

Output: 2

Explanation: We can equalize the array in 2 seconds in the following way:

- At 1st second, replace values at each index with [nums[0],nums[2],nums[2],nums[2],nums[3]]. After replacement, nums = [2,3,3,3,3].
- At 2nd second, replace values at each index with [nums[1],nums[1],nums[2],nums[3],nums[4]]. After replacement, nums = [3,3,3,3,3].

It can be proven that 2 seconds is the minimum amount of seconds needed for equalizing the array.

Example 3:

Input: nums = [5,5,5,5]

Output: 0

Explanation: We don't need to perform any operations as all elements in the initial array are the same.

Hints:

```
class Solution(object):
    def minimumSeconds(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
```

Try
1. Take nums = [1,1,1,2,2,2,3,3] and verify the results.
2 .Take nums = [0,1,0,1,1,2,3,1] and verify the results.

3.3 One Liner Using DATETIME

Given

1. The input is of type <string>. To use the datetime module, these strings will first be converted into type <date> using datetime.strptime(date_string, format).
2. After conversion, the dates are subtracted, i.e. (date2 - date1).days()

Note:
abs() must be used when calculating the difference as any of the dates could be bigger than the other.

Hints:

```
from datetime import datetime
class Solution:
    def daysBetweenDates(self, date1: str, date2: str) -> int:
        M = datetime.strptime(date1, '%Y-%m-%d').date()
        N = datetime.strptime(date2, '%Y-%m-%d').date()
        return abs((N - M).days)
class Solution(object):
    def daysBetweenDates(self, date1, date2):
        """
        :type date1: str
        :type date2: str
        :rtype: int
        """
```

Try
1. Take date as `'%Y-%D-%M' and verify the result.`

3.4 Time Based Key-Value Store

Design a time-based key-value data structure that can store multiple values for the same key at different time stamps and retrieve the key's value at a certain timestamp.
Implement the TimeMap class:

- TimeMap() Initializes the object of the data structure.
- void set(String key, String value, int timestamp) Stores the key key with the value value at the given time timestamp.
- String get(String key, int timestamp) Returns a value such that set was called previously, with timestamp_prev <= timestamp. If there are multiple such values, it returns the value associated with the largest timestamp_prev. If there are no values, it returns "".

Example 1:
Input
["TimeMap", "set", "get", "get", "set", "get", "get"]
[[], ["foo", "bar", 1], ["foo", 1], ["foo", 3], ["foo", "bar2", 4], ["foo", 4], ["foo", 5]]
Output
[null, null, "bar", "bar", null, "bar2", "bar2"]

Explanation

TimeMap timeMap = new TimeMap();
timeMap.set("foo", "bar", 1); // store the key "foo" and value "bar" along with timestamp = 1.
timeMap.get("foo", 1);   // return "bar"
timeMap.get("foo", 3);   // return "bar", since there is no value corresponding to foo at timestamp 3 and timestamp 2, then the only value is at timestamp 1 is "bar".
timeMap.set("foo", "bar2", 4); // store the key "foo" and value "bar2" along with timestamp = 4.
timeMap.get("foo", 4);   // return "bar2"
timeMap.get("foo", 5);   // return "bar2"

Hints:

```
class TimeMap(object):

    def __init__(self):

    def set(self, key, value, timestamp):
        """
        :type key: str
        :type value: str
        :type timestamp: int
        :rtype: None
        """
    def get(self, key, timestamp):
        """
        :type key: str
        :type timestamp: int
        :rtype: str
        """

# Your TimeMap object will be instantiated and called as such:
# obj = TimeMap()
# obj.set(key,value,timestamp)
# param_2 = obj.get(key,timestamp)
```

Try

1. Take ["TimeMap", "got", "got", "got", "hot", "hot", "hot"]
[[], ["foo", "bar", 1], ["foo", 1], ["foo", 3], ["foo", "bar2", 4], ["foo", 4], ["foo", 5]] and verify the results.

3.5 Number of Days between Two Dates

Write a program to count the number of days between two dates.

The two dates are given as strings, their format is YYYY-MM-DD as shown in the examples.

Example 1:

Input: date1 = "2019-06-29", date2 = "2019-06-30"

Output: 1

Example 2:

Input: date1 = "2020-01-15", date2 = "2019-12-31"

Output: 15

Hints:

```
class Solution(object):
```

```
    def daysBetweenDates(self, date1, date2):
        """
        :type date1: str
        :type date2: str
        :rtype: int
        """
```

Try

1. Take Input: date1 = "2020-06-29", date2 = "2023-06-30" and verify the results.
2. Take Input: date1 = "1959-06-29", date2 = "1991-06-30" and verify the results.

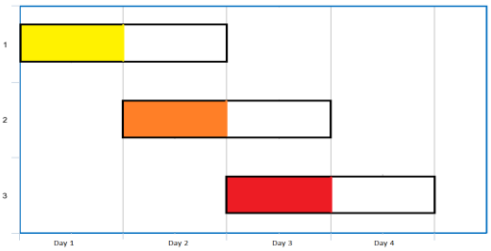
3.6 Maximum Number of Events That Can Be Attended

You are given an array of events where events[i] = [$startDay_i$, $endDay_i$]. Every event i starts at $startDay_i$ and ends at $endDay_i$.

You can attend an event i at any day d where $startTime_i$ <= d <= $endTime_i$. You can only attend one event at any time d.

Return *the maximum number of events you can attend*.

Example 1:



Input: events = [[1,2],[2,3],[3,4]]

Output: 3

Explanation: You can attend all the three events.

One way to attend them all is as shown.

Attend the first event on day 1.

Attend the second event on day 2.

Attend the third event on day 3.

Example 2:

Input: events= [[1,2],[2,3],[3,4],[1,2]]

Output: 4

Hints:

```
class Solution(object):
    def maxEvents(self, events):
        """
        :type events: List[List[int]]
        :rtype: int
        """
```

Try

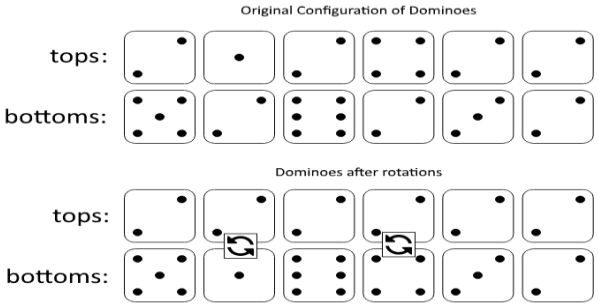
1. Take Input: events = [[11,12],[12,13],[13,14],[14,15]] and verify the results.
2. Take Input: events = [[10,12],[12,30],[30,14],[14,55]] and verify the results.

4. Problems solving using Python

4.1 Minimum Domino Rotations For Equal Row

In a row of dominoes, tops[i] and bottoms[i] represent the top and bottom halves of the i^{th} domino. (A domino is a tile with two numbers from 1 to 6 - one on each half of the tile.)
We may rotate the i^{th} domino, so that tops[i] and bottoms[i] swap values.
Return the minimum number of rotations so that all the values in tops are the same, or all the values in bottoms are the same.
If it cannot be done, return -1.

Example 1:



Input: tops = [2,1,2,4,2,2], bottoms = [5,2,6,2,3,2]
Output: 2
Explanation:
The first figure represents the dominoes as given by tops and bottoms: before we do any rotations.
If we rotate the second and fourth dominoes, we can make every value in the top row equal to 2, as indicated by the second figure.

Example 2:
Input: tops = [3,5,1,2,3], bottoms = [3,6,3,3,4]
Output: -1
Explanation:
In this case, it is not possible to rotate the dominoes to make one row of values equal.

Hints:

```
class Solution(object):
    def minDominoRotations(self, tops, bottoms):
        """
        :type tops: List[int]
        :type bottoms: List[int]
        :rtype: int
        """
```

Try
1. Take input as tops = [1,1,2,2,3,3], bottoms = [2,2,3,3,5,5] and verify the results.
2. Take input as tops = [0,0,2,2,1,1], bottoms = [1,1,0,0,7,7] and verify the results.

4.2 Search in Rotated Sorted Array

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **possibly rotated** at an unknown pivot index k (1 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].
Given the array nums **after** the possible rotation and an integer target, return *the index of* target *if it is in* nums*, or* -1 *if it is not in* nums.
You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4

Example 2:
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1

Example 3:
Input: nums = [1], target = 0
Output: -1

Hints:
```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
```

Try
1. Take Input: nums = [3,4,5,6,7,8,9], target = 3 and verify the result.
2. Take Input: nums = [10,20,30,60,70,90], target = 4 and verify the result.

4.3 Longest Consecutive Sequence

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*
You must write an algorithm that runs in O(n) time.

Example 1:
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

Example 2:
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9

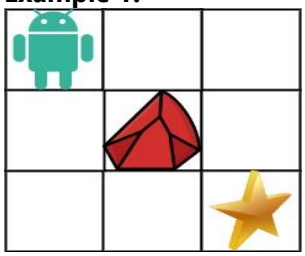
Hints:
```
class Solution(object):
    def longestConsecutive(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
```

Try
1. Take Input: nums = [0,1,7,2,5,8,5,6,3,1] and verify the result.
2. Take Input: nums = [1,8,1,5,1,5,1,3,1] and verify the result.

4.4 Unique Paths II

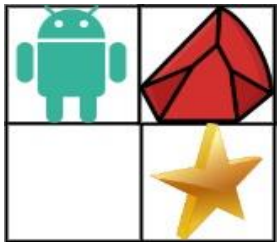
You are given an m x n integer array grid. There is a robot initially located at the **top-left corner** (i.e., grid[0][0]). The robot tries to move to the **bottom-right corner** (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.
An obstacle and space are marked as 1 or 0 respectively in grid. A path that the robot takes cannot include **any** square that is an obstacle.
Return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.
The testcases are generated so that the answer will be less than or equal to $2 * 10^9$.

Example 1:



Input: obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]
Output: 2
Explanation: There is one obstacle in the middle of the 3x3 grid above.
There are two ways to reach the bottom-right corner:
1. Right -> Right -> Down -> Down
2. Down -> Down -> Right -> Right

Example 2:



Input: obstacleGrid = [[0,1],[0,0]]
Output: 1

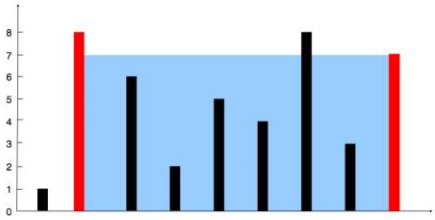
Hints:

```
class Solution(object):
    def uniquePathsWithObstacles(self, obstacleGrid):
        """
        :type obstacleGrid: List[List[int]]
        :rtype: int
        """
```

Try
1. Take Input: obstacleGrid = [[0,1 ,0],[1,1,1],[0,0,1]] and verify the result.
2. Take Input: obstacleGrid = [[1,1 ,1],[1,1,1],[0,0,1]] and verify the result.

4.5 Container with Most Water

You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the i^{th} line are (i, 0) and (i, height[i]).
Find two lines that together with the x-axis form a container, such that the container contains the most water.
Return *the maximum amount of water a container can store*.
Notice that you may not slant the container.
Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

Example 2:
Input: height = [1,1]
Output: 1

Hints:

```
class Solution(object):
    def maxArea(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
```

Try
1. Take height = [1,8,6,2,5,4,8,3,7] and verify the results.
2. Take height = [1, 1, 1] and verify the results.

4.6Frog Jump

A frog is crossing a river. The river is divided into some number of units, and at each unit, there may or may not exist a stone. The frog can jump on a stone, but it must not jump into the water.
Given a list of stones positions (in units) in sorted **ascending order**, determine if the frog can cross the river by landing on the last stone. Initially, the frog is on the first stone and assumes the first jump must be 1 unit.
If the frog's last jump was k units, its next jump must be either k - 1, k, or k + 1 units. The frog can only jump in the forward direction.

Example 1:
Input: stones = [0,1,3,5,6,8,12,17]
Output: true

Explanation: The frog can jump to the last stone by jumping 1 unit to the 2nd stone, then 2 units to the 3rd stone, then 2 units to the 4th stone, then 3 units to the 6th stone, 4 units to the 7th stone, and 5 units to the 8th stone.

Example 2:
Input: stones = [0,1,2,3,4,8,9,11]
Output: false
Explanation: There is no way to jump to the last stone as the gap between the 5th and 6th stone is too large.

Hints:

```
class Solution(object):
    def canCross(self, stones):
        """
        :type stones: List[int]
        :rtype: bool
        """
```

Try
1. Take Input: stones = [1, 2, 4, 3, 4, 8, 9, and 11] and verify the results.
2. Take Input: stones = [2, 3, 5, 7, 4, 8, 11 and 13] and verify the results.

5. Analytical problems using Python

5.1 The Monty Hall Problem

The Monty Hall problem is a counter-intuitive brain teaser based on probabilities.
This puzzle is named after Monty Hall, the TV presenter of "Let's Make a Deal", an American TV show (known as "Deal or No Deal" in the UK) where contenders swap boxes of different values in order to win a prize.
In a similar approach, the aim of the Monty Hall puzzle is to try to win a car by knocking on the right door. The player is presented with three closed doors. Behind one of these doors is a luxurious car. Behind each of the other two doors is a goat!

Here is how the game proceeds:

- You first pick a door. It could be that you picked the door with the car but you do not know that yet!
- The game show host examines the other two doors and opens one with a goat. (Note that when both doors are hiding a goat, the presenter picks one of these doors randomly.)
- You then need to make a decision: You can either stick with your initial guess and open the door to see if you win the car. Alternatively, you can decide to switch your door with the other closed door!

Hints:

```
 # The Monty Hall Problem - Frequency Analysis
import random

#Let's initialise our 3 doors
doors = ["goat", "goat", "car"]

# Randomly position the two goats and the car behind the three doors
random.shuffle(doors)

# Randomly pick a door and display the selected door number
# Get the computer to identify the two doors which have not been selected
#   If only one of these two doors contains a goat, display the door number to
reveal the goat
#   If both doors contain a goat, pick one of the two doors randomly and display
its number to reveal the goat
# Get the computer to randomly decide whether it will keep the selected door or
switch to the other closed door
# Reveal the content of all three doors
# Check if the car was behind the selected door
# Keep count of wins and loses when the user decide to switch doors or not
# Display these counters/statistics
# Repeat the above process 100 times.
# If your code is working fine you should reach statistics to confirm that:
#     When switching doors your are twice as likely to win the car
#     When not switching doors your are twice as likely to get the goat!

# Write the code here
```

Try

1. Take input as 3 doors = ["car", "goat", "car"] and verify the results.
2. Take input as 3 doors = ["car", "goat", "goat"] and verify the results.

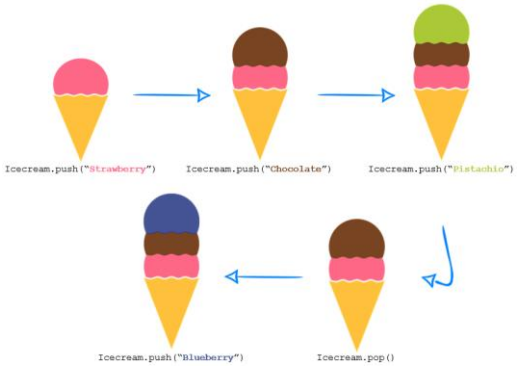
5.2 The ice cream Stack

In this blog post, we will investigate the use of a **Stack data structure** to store the different flavours of the different scoops of an ice cream!

A stack is a **FILO data structure**: **First In Last Out** and seems to be a suitable data structure for our ice cream, where the first scoop of ice-cream being added on the cone, will be the last one to be eaten!

With a Stack data structure, you can either:

- **Push new data** at the end of the stack.
- **Pop the last value** of the stack.



In order to implement our stack will use a **1D array** of 5 values, assuming that you cannot make an ice cream with more than 5 scoops! We will use a **pointer** to store the index of the next empty cell of our array where data can be added.

Using **OOP programming**, we will create an *IceCream* **class** to implement our ice cream as a stack. We will **encapsulate** both the array data structure and the stack pointer as **private properties** of the *IceCream* class and we will use two **public methods** to *push()* and *pop()* scoops of different flavours to/from our IceCream!

Hints:

```
#A class to implement a Stack that can hold up to 5 values
class IceCream():
    #Constructor...
    def __init__(self):
        self.__array = [None,None,None,None,None] #__Private Property
        self.__pointer = 0  #__Private Property
        self.__maxCapacity = len(self.__array) #__Private Property

    #A method to push a new scoop/flavour to the ice cream
    def push(self, flavour):
     ……

    #A method to pop the last scoop/flavour of the ice cream
    def pop(self):
        #Write the code here
#Let's instantiate our first ice cream!
myIceCream = IceCream()
```

Try

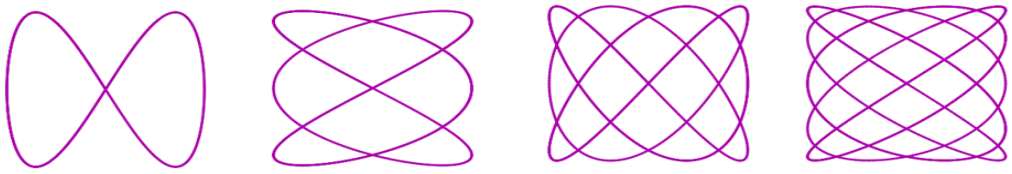
1. Take input a `stack that can hold up to 10 values and verify the results.`
2. Take input a `stack that can hold up to 15 values and verify the results.`

5.3 Lissajous Curve Tracing Algorithm

Lissajous curves are a family of curves described by the following parametric equations:

$$x(t) = A \sin(at + \delta)$$
$$y(t) = B \sin(bt)$$

Lissajous curves have applications in physics, astronomy, and other sciences. Below are a few examples of Lissajous curves that you will be able to reproduce in the Python Trinket provided below by changing the values of constant A and B in the Python code.



a=1, b=2 δ = π / 2 | a=3, b=2 δ = π / 2 | a=3, b=4 δ = π / 2 | a=5, b=4 δ = π / 2

Hints:

```
#Python Turtle - Lissajous Curve
import turtle
from math import cos,sin
from time import sleep

window = turtle.Screen()
window.bgcolor("#FFFFFF")

myPen = turtle.Turtle()
myPen.hideturtle()
myPen.tracer(0)
myPen.speed(0)
myPen.pensize(3)
myPen.color("#AA00AA")

myPen.penup()

A = 100
B = 100
a = 3
b = 4
delta = 3.14/2
t=0

for i in range(0,1000):
    #Write the code here
sleep(0.5)
```

Try

1. Take input as A = 100, B = 100, a = 3, b = 4, delta = 3.14/2, t=0 and verify the results.
2. Take input as A = 150, B = 150, a = 4, b = 3, delta = 3.14/2, t=2 and verify the results.

5.4 An interactive calculator

You're going to write an interactive calculator! User input is assumed to be a formula that consist of a number, an operator (at least + and -), and another number, separated by white space (e.g. 1 + 1). Split user input using str.split(), and check whether the resulting list is valid:

- If the input does not consist of 3 elements, raise a FormulaError, which is a custom Exception.
- Try to convert the first and third input to a float (like so: float_value = float(str_value)). Catch any ValueError that occurs, and instead raise a FormulaError
- If the second input is not '+' or '-', again raise a FormulaError

If the input is valid, perform the calculation and print out the result. The user is then prompted to provide new input, and so on, until the user enters quit.
An interaction could look like this:

```
>>> 1 + 1
2.0
```

```
>>> 3.2 - 1.5
1.7000000000000002
>>> quit
```

Hints:

```
Class FormulaError(Exception): pass


def parse_input(user_input):

  input_list = user_input.split()
  if len(input_list) != 3:
    raise FormulaError('Input does not consist of three elements')
  n1, op, n2 = input_list
  try:
    n1 = float(n1)
    n2 = float(n2)
  except ValueError:
    raise FormulaError('The first and third input value must be numbers')
  return n1, op, n2


def calculate(n1, op, n2):
  #Write the code here
```

Try

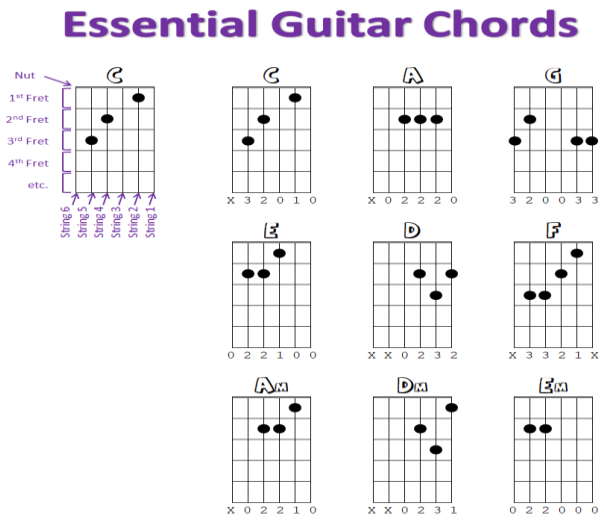
1. Take input as an operator (* and /) and verify the results.

5.5 Guitar Chords Reader

The idea of this python challenge is to write a python program to help guitar players learn and practise new songs.

Our program will read all the chords used in a song and display and animate a visual representation/chart of the chord being played.

First, let's recap what are the main chords when playing the guitar:



The code appearing underneath each chord indicates the position of the fingers as described on each chart. With this code:

- "x" means the string is not played,
- "0" means the string played as an open string,
- a number tells you which fret to place your finger on.

Hints:

```
#Guitar Chords Reader - www.101computing.net/guitar-chords-reader/
import time
import os

#A Python Dictionary matching chord names with "fret notation"
chords = {"C": "x32010", "A":"x02220", "G": "320033", "E": "022100", "D":
"xx0232", "F": "x3321x", "Am": "x02210", "Dm": "xx0231", "Em": "022000"}

#A procedure to display where to position your fingers to play a given chord
def displayChord(chord):
  fretNotation = chords[chord]

  print("  " + chord)
  nut=""
  for string in fretNotation:
    if string=="x":
      nut=nut+"x"  # x means don't play this string
    else:
      nut = nut + "_"
  print(nut) #Guitar Nut
  for fretNumber in range(1,5):
    fret=""
    for string in fretNotation:
      if string==str(fretNumber):
        fret=fret+"O"
      else:
        fret = fret + "|"
    print(fret)

#Main Program Starts Here
song = "C,D,G,Em,C,D,G,Em"

#Let's read this song, one chord at a time
songChords = song.split(",")
for chord in songChords:
  displayChord(chord)
  time.sleep(2)
  #Clear the screen
  os.system("cls")
```

Try

1. Take input chords = {"C": "x31010", "A":"x01110", "G": "320022", "E": "11100", "D": "xx0242", "F": "x3321x", "Am": "x02110", "Dm": "xx0211", "Em": "011000"} and verify the results.
2. Take input chords = {"C": "x31110", "A":"x01110", "G": "xx0022", "E": "xx100", "D": "xx1242", "F": "x1121x", "Am": "x011110", "Dm": "xx0111", "Em": "011000"} and verify the results.

6. Game problems using Python

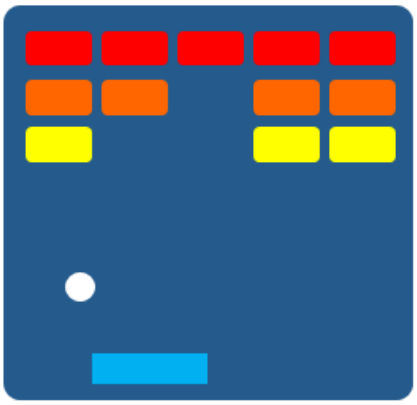
6.1 Breakout Tutorial using Pygame: Adding a Brick Wall

This tutorial is a series of following Pygame modules:

- Adding the Paddle
- Controlling the Paddle
- Adding a Bouncing Ball
- Adding a Brick Wall

The final stage of our tutorial focuses on adding a brick wall and a scoring system to our Breakout game:

- The player will score a point if the ball bounces against a brick.
- The player will lose a life if the ball bounces against the bottom edge of the screen.
- Both the score and number of lives will be displayed at the top of the screen.
- A "Level Complete" message will be displayed if all bricks have been removed.
- A "Game Over" message will be displayed if the number of lives reaches zero.



The final code for the main.py is provided below. We made several changes to the code as follows:

- **On line 6** we import the *Brick* class. (Code provided in the *brick.py* tab)
- **On lines 39 to 57** we create three rows of bricks and add them to a group called *all_bricks*.
- **On lines 93 to 103** we take a life away when the ball hit the bottom edge of the screen. If the number of lives reaches zero, we display a "Game Over" message..
- **On lines 114 to 129** we detect if the ball hits a brick. If so we remove the brick (using the *kill()* method) and increment the score by one.

Hints:

```
#Import the pygame library and initialise the game engine
import pygame
#Let's import the Paddle Class & the Ball Class
from paddle import Paddle
from ball import Ball
from brick import Brick
pygame.init()
# Define some colors
WHITE = (255,255,255)
DARKBLUE = (36,90,190)
LIGHTBLUE = (0,176,240)
RED = (255,0,0)
ORANGE = (255,100,0)
YELLOW = (255,255,0)
```

```
score = 0
lives = 3
# Open a new window
size = (800, 600)
screen = pygame.display.set_mode(size)
pygame.display.set_caption("Breakout Game")

#Write code here
```

Try

1. Take input WHITE = (90, 90, 90), DARKBLUE = (90, 90,190), LIGHTBLUE = (90,176,240), RED = (0, 0,0)
   ORANGE = (255,100,255), YELLOW = (255,255,255), score = 0, lives = 3 and verify the results.

6.2 Dice Score Frequency Analysis

Let's consider a single 6-sided dice.

When using single dice, the score of the dice can be any of the 6 values/sides (1 to 6). Each value has an equal probability of 1/6.

The dice experiment can be simulated using a computer algorithm to simulate throwing a dice a 1,000 times and keeping a record of the number of times each of the 6 values appeared. We can then calculate the frequency of each value as a percentage. In theory, each value should have a very similar frequency of around 16.67% (matching the 1/6 probability).

When completing such an experiment in real life, we would mot likely use a tally chart to record the number of occurrences for each of the 6 possible values.

Dice Value	Tallies	Frequency
	𝍸 𝍸 ||	12
	𝍸 𝍸	10
	𝍸 𝍸 |||	13
	𝍸 𝍸 ||	12
	𝍸 𝍸 |	11
	𝍸 𝍸	10

To record our "tallies" in a Python program we will use a **dictionary** with 6 keys (the 6 values of a dice, from 1 to 6) and for each key, the associated value will be the number of occurrences of when the value appeared.

Hints:

```
#Dice Score Frequency Analysis

#A function to simulate throwing a 6-sided dice
def throwDice():
  dice = random.randint(1,6)
  return dice


### Main Code Starts Here...
```

```
#Initialise a dictionary to store the frequency (tally count) of each of the 6
dice values
tallyChart = {1:0, 2:0, 3:0, 4:0, 5:0, 6:0}

#Number of throws
n = 1000
#Let's start the experiment and repeat it n times! Let's complete our tally chart!
for i in range(0,n):
  score = throwDice()
  tallyChart[score] += 1

#Let's display the results:
print(" Dice Value | Frequency | Percentage ")
print("-----------------------------------")
for i in range(1,7):
  frequency = tallyChart[i]
  percentage = round((frequency * 100) / n , 2)
  print("      " + str(i) + "     |    " + str(frequency)+ "    |   " +
str(percentage) + "%")
```

Try

1. Write a function to simulate throwing a two 6-sided dice and increase number of steps to 1000.

6.3 Boggle Challenge

Boggle is a word game based on a 4×4 grid of 16 letters. Each time the game is played a new grid is generated. In the real game this is done by shaking a cube that contains 16 letter dice. The The aim of the game is to find words that can be constructed from sequentially adjacent letters from the 4×4 grid. "Adjacent" letters are those horizontally, vertically, and diagonally neighbouring. Words must be at least three letters long, may include singular and plural (or other derived forms) separately, but may not use the same letter from the grid more than once per word.

A scoring system can be used based on the number of words identified and by adding the number of letters in each word.

D	E	T	I
C	O	E	G
S	D	L	N
V	X	I	A

Hints:

```
#Boggle Challenge
import random

letter=chr(random.randint(65,90)) #Generate a random Uppercase letter
(based on ASCII code)
print(letter)
```

Try

1. Write a code to `generate a random lower and uppercase letter (based on ASCII code).`
2. Write a code to `generate a random lower letter (based on ASCII code).`

6.4 Langton's Ant

At the start of the simulation, the ant is randomly positioned on a 2D-grid of white cells. The ant is also given a direction (either facing up, down, left or right).

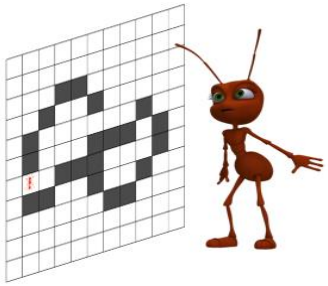
The ant then moves according to the colour of the cell it is currently sitting in, with the following rules:

1. If the cell is white, it changes to black and the ant turns right 90°.
2. If the cell is black, it changes to white and the ant turns left 90°.
3. The ant then moves forward to the next cell, and repeat from step 1.

These simple rules lead to complex behaviours. Three distinct modes of behaviour are apparent, when starting on a completely white grid:

1. **Simplicity:** During the first few hundred moves it creates very simple patterns which are often symmetric.
2. **Chaos:** After a few hundred moves, a big, irregular pattern of black and white squares appears. The ant traces a pseudo-random path until around 10,000 steps.
3. **Emergent order:** Finally the ant starts building a recurrent "highway" pattern of 104 steps that repeats indefinitely.

All finite initial configurations tested eventually converge to the same repetitive pattern, suggesting that the "highway" is an attractor of Langton's ant, but no one has been able to prove that this is true for all such initial configurations.



```
#Langton's Ant
import turtle
import time
from random import randint

#Change this value to speed up or slow down this animation
animationSpeed=2

gridSize = 15
myPen = turtle.Turtle()
myPen.shape("turtle")
myPen.tracer(0)
myPen.speed(0)
myPen.color("#000000")
topLeft_x=-180
topLeft_y=180

#Draw the grid on screen (intDim is the width of a cell on the grid)
def drawGrid(grid,intDim):
# Write code here
```

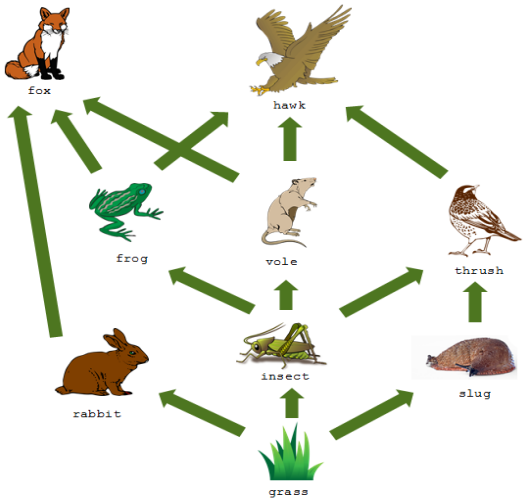
Try

1. Write a code to verify the result considering input as animationSpeed=1 and gridSize = 5
2. Write a code to verify the result considering input as animationSpeed=6 and gridSize = 25

6.5 Food Chain Game Using Python

For this challenge you will write a Python program that stores the organisms a food chain using a **list**.

```
foodChain = ["Grass" , "Grasshopper" , "Frog", "Snake", "Eagle"]
```



You program will randomly pick two organisms from the food chain. One for the player, and one for the computer.
The program will find out the positions of these organisms in the given food chain. (This is known as the **trophic level** of an organism which is the position it holds in a food chain).
The program will compare both positions; the player with the highest position in the food chain will win the game.
We have started the code for you. Complete this code to:

- Randomly select the "computer organism" from the list.
- Make sure that both selected organisms are different.
- Compare the positions of both organisms to decide who, between the computer and the player, wins the round.

Hints:

```
#Food Chain Game Using Python
import random

foodChain = ["Grass" , "Grasshopper" , "Frog", "Snake", "Eagle"]

foodChainLength = len(foodChain)

playerPosition = random.randint(0,foodChainLength-1)
playerOrganism = foodChain[playerPosition]

print("Player Organism: " + playerOrganism)

#Complete code here to select a different organism for the computer
```

```
#Then work out who has the highest position to identify the winner (Player or
computer)

# Write he code here
```

Try

1. Write a program and verify the results by considering input as foodChain = ["Grass", "Grasshopper", "Frog", "Snake", "Eagle"] in reverse order.

7. Multithreading and Regular Expressions

7.1 Fizz Buzz Multithreaded

You have the four functions:

- printFizz that prints the word "fizz" to the console,
- printBuzz that prints the word "buzz" to the console,
- printFizzBuzz that prints the word "fizzbuzz" to the console, and
- printNumber that prints a given integer to the console.

You are given an instance of the class FizzBuzz that has four functions: fizz, buzz, fizzbuzz and number. The same instance of FizzBuzz will be passed to four different threads:

- **Thread A:** calls fizz() that should output the word "fizz".
- **Thread B:** calls buzz() that should output the word "buzz".
- **Thread C:** calls fizzbuzz() that should output the word "fizzbuzz".
- **Thread D:** calls number() that should only output the integers.

Modify the given class to output the series [1, 2, "fizz", 4, "buzz", ...] where the i^{th} token (**1-indexed**) of the series is:

- "fizzbuzz" if i is divisible by 3 and 5,
- "fizz" if i is divisible by 3 and not 5,
- "buzz" if i is divisible by 5 and not 3, or
- i if i is not divisible by 3 or 5.

Implement the FizzBuzz class:

- FizzBuzz(int n) Initializes the object with the number n that represents the length of the sequence that should be printed.
- void fizz(printFizz) Calls printFizz to output "fizz".
- void buzz(printBuzz) Calls printBuzz to output "buzz".
- void fizzbuzz(printFizzBuzz) Calls printFizzBuzz to output "fizzbuzz".
- void number(printNumber) Calls printnumber to output the numbers.

Example 1:
Input: n = 15
Output: [1,2,"fizz",4,"buzz","fizz",7,8,"fizz","buzz",11,"fizz",13,14,"fizzbuzz"]

Example 2:
Input: n = 5
Output: [1,2,"fizz",4,"buzz"]

Hints:

```
Class FizzBuzz(object):
    def __init__(self, n):
        self.n = n

    # printFizz() outputs "fizz"
    def fizz(self, printFizz):
        """
        :type printFizz: method
        :rtype: void
        """
    # printBuzz() outputs "buzz"
    def buzz(self, printBuzz):
        """
        :type printBuzz: method
        :rtype: void
        """
    # printNumber(x) outputs "x", where x is an integer.
    def number(self, printNumber):
        """
        :type printNumber: method
        :rtype: void
        """
```

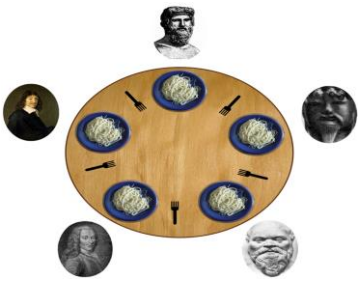
Try
1. Write a program and verify the results by giving input: n = 15.
2. Take input: n = 50 and verify the results.

7.2 The Dining Philosophers

Five silent philosophers sit at a round table with bowls of spaghetti. Forks are placed between each pair of adjacent philosophers.

Each philosopher must alternately think and eat. However, a philosopher can only eat spaghetti when they have both left and right forks. Each fork can be held by only one philosopher and so a philosopher can use the fork only if it is not being used by another philosopher. After an individual philosopher finishes eating, they need to put down both forks so that the forks become available to others. A philosopher can take the fork on their right or the one on their left as they become available, but cannot start eating before getting both forks.
Eating is not limited by the remaining amounts of spaghetti or stomach space; an infinite supply and an infinite demand are assumed.

Design a discipline of behaviour (a concurrent algorithm) such that no philosopher will starve; *i.e.*, each can forever continue to alternate between eating and thinking, assuming that no philosopher can know when others may want to eat or think.



The philosophers' ids are numbered from **0** to **4** in a **clockwise** order. Implement the function void wantsToEat(philosopher, pickLeftFork, pickRightFork, eat, putLeftFork, putRightFork) where:

- philosopher is the id of the philosopher who wants to eat.
- pickLeftFork and pickRightFork are functions you can call to pick the corresponding forks of that philosopher.
- eat is a function you can call to let the philosopher eat once he has picked both forks.
- putLeftFork and putRightFork are functions you can call to put down the corresponding forks of that philosopher.
- The philosophers are assumed to be thinking as long as they are not asking to eat (the function is not being called with their number).

Five threads, each representing a philosopher, will simultaneously use one object of your class to simulate the process. The function may be called for the same philosopher more than once, even before the last call ends.

Example 1:
Input: n = 1
Output:
[[4,2,1],[4,1,1],[0,1,1],[2,2,1],[2,1,1],[2,0,3],[2,1,2],[2,2,2],[4,0,3],[4,1,2],[0,2,1],[4,2,2],[3,2,1],[3,1,1],[0,0,3],[0,1,2],[0,2,2],[1,2,1],[1,1,1],[3,0,3],[3,1,2],[3,2,2],[1,0,3],[1,1,2],[1,2,2]]

Explanation:
n is the number of times each philosopher will call the function.
The output array describes the calls you made to the functions controlling the forks and the eat function, its format is:
output[i] = [a, b, c] (three integers)
- a is the id of a philosopher.
- b specifies the fork: {1: left, 2: right}.
- c specifies the operation: {1: pick, 2: put, 3: eat}.

Hints:

```
class DiningPhilosophers(object):

    # call the functions directly to execute, for example, eat()
    def wantsToEat(self, philosopher, pickLeftFork, pickRightFork, eat,
putLeftFork, putRightFork):
        """
        :type philosopher: int
        :type pickLeftFork: method
        :type pickRightFork: method
        :type eat: method
```

```
        :type putLeftFork: method
        :type putRightFork: method
        :rtype: void
        """
```

Try

1. Write a program and verify the results by considering input n=2 in anti-clockwise direction.
2. Write a program and verify the results by considering input n=2 in clockwise direction.

7.3 Print FooBar Alternately

Suppose you are given the following code:

```
class FooBar {
  public void foo() {
    for (int i = 0; i < n; i++) {
      print("foo");
    }
  }
  public void bar() {
    for (int i = 0; i < n; i++) {
      print("bar");
    }
  }
}
```

The same instance of FooBar will be passed to two different threads:

- thread A will call foo(), while
- thread B will call bar().

Modify the given program to output "foobar" n times.

Example 1:

Input: n = 1
Output: "foobar"
Explanation: There are two threads being fired asynchronously. One of them calls foo(), while the other calls bar().
"foobar" is being output 1 time.

Example 2:

Input: n = 2
Output: "foobarfoobar"
Explanation: "foobar" is being output 2 times.

Hints:

```
class FooBar(object):
    def __init__(self, n):
        self.n = n

    def foo(self, printFoo):
        """
        :type printFoo: method
        :rtype: void
        """
        for i in xrange(self.n):
```

```
            # printFoo() outputs "foo". Do not change or remove this line.
            printFoo()

    def bar(self, printBar):
        """
        :type printBar: method
        :rtype: void
        """
        for i in xrange(self.n):

            # printBar() outputs "bar". Do not change or remove this line.
            printBar()
```

Try
1. Take Input: n = 3 and verify the results.
2. Take Input: n = 4 and verify the results.

7.4 Minimum Penalty for a Shop

You are given the customer visit log of a shop represented by 0-indexed string customers consisting only of characters 'N' and 'Y':

- if the i^{th} character is 'Y', it means that customers come at the i^{th} hour
- whereas 'N' indicates that no customers come at the i^{th} hour.

If the shop closes at the j^{th} hour (0 <= j <= n), the **penalty** is calculated as follows:

- For every hour when the shop is open and no customers come, the penalty increases by 1.
- For every hour when the shop is closed and customers come, the penalty increases by 1.

Return *the **earliest** hour at which the shop must be closed to incur a **minimum** penalty.*
Note that if a shop closes at the j^{th} hour, it means the shop is closed at the hour j.

Example 1:
Input: customers = "YYNY"
Output: 2
Explanation:
- Closing the shop at the 0^{th} hour incurs in 1+1+0+1 = 3 penalty.
- Closing the shop at the 1^{st} hour incurs in 0+1+0+1 = 2 penalty.
- Closing the shop at the 2^{nd} hour incurs in 0+0+0+1 = 1 penalty.
- Closing the shop at the 3^{rd} hour incurs in 0+0+1+1 = 2 penalty.
- Closing the shop at the 4^{th} hour incurs in 0+0+1+0 = 1 penalty.
Closing the shop at 2^{nd} or 4^{th} hour gives a minimum penalty. Since 2 is earlier, the optimal closing time is 2.

Example 2:
Input: customers = "NNNNN"
Output: 0
Explanation: It is best to close the shop at the 0^{th} hour as no customers arrive.

Example 3:
Input: customers = "YYYY"
Output: 4

Explanation: It is best to close the shop at the 4th hour as customers arrive at each hour.

Hints:

```
class Solution(object):
    def bestClosingTime(self, customers):
        """
        :type customers: str
        :rtype: int
        """
```

Try

1. Take input customers = "XXYY" and verify the results.
2. Take input customers = "NNYY" and verify the results.

7.5 Print Zero Even Odd

You have a function printNumber that can be called with an integer parameter and prints it to the console.

For example, calling printNumber(7) prints 7 to the console.
You are given an instance of the class ZeroEvenOdd that has three functions: zero, even, and odd. The same instance of ZeroEvenOdd will be passed to three different threads:

- Thread A: calls zero() that should only output 0's.
- Thread B: calls even() that should only output even numbers.
- Thread C: calls odd() that should only output odd numbers.

Modify the given class to output the series "010203040506..." where the length of the series must be 2n.
Implement the ZeroEvenOdd class:

- ZeroEvenOdd(int n) Initializes the object with the number n that represents the numbers that should be printed.
- void zero(printNumber) Calls printNumber to output one zero.
- void even(printNumber) Calls printNumber to output one even number.
- void odd(printNumber) Calls printNumber to output one odd number.

Example 1:
Input: n = 2
Output: "0102"
Explanation: There are three threads being fired asynchronously.
One of them calls zero(), the other calls even(), and the last one calls odd().
"0102" is the correct output.

Example 2:
Input: n = 5
Output: "0102030405"

Hints:

```
class ZeroEvenOdd(object):
    def __init__(self, n):
        self.n = n
```

```
    # printNumber(x) outputs "x", where x is an integer.
    def zero(self, printNumber):
        """
        :type printNumber: method
        :rtype: void
        """

    def even(self, printNumber):
        """
        :type printNumber: method
        :rtype: void
        """

    def odd(self, printNumber):
        """
        :type printNumber: method
        :rtype: void
        """
```

Try

1. Take input n=5 and n=3 and verify the results.
2. Take input n=3 and verify the results.

7.6 Print in Order

Suppose we have a class:

```
public class Foo {
  public void first() { print("first"); }
  public void second() { print("second"); }
  public void third() { print("third"); }
}
```

The same instance of Foo will be passed to three different threads. Thread A will call first(), thread B will call second(), and thread C will call third(). Design a mechanism and modify the program to ensure that second() is executed after first(), and third() is executed after second().

Note:

We do not know how the threads will be scheduled in the operating system, even though the numbers in the input seem to imply the ordering. The input format you see is mainly to ensure our tests' comprehensiveness.

Example 1:

Input: nums = [1,2,3]
Output: "firstsecondthird"
Explanation: There are three threads being fired asynchronously. The input [1,2,3] means thread A calls first(), thread B calls second(), and thread C calls third(). "firstsecondthird" is the correct output.

Example 2:

Input: nums = [1,3,2]
Output: "firstsecondthird"
Explanation: The input [1,3,2] means thread A calls first(), thread B calls third(), and thread C calls second(). "firstsecondthird" is the correct output.

Hints:

```
Class Foo(object):
```

```
    def __init__(self):
        pass

    def first(self, printFirst):
        """
        :type printFirst: method
        :rtype: void
        """

        # printFirst() outputs "first". Do not change or remove this line.
        printFirst()

    def second(self, printSecond):
        """
        :type printSecond: method
        :rtype: void
        """

        # printSecond() outputs "second". Do not change or remove this line.
        printSecond()


    def third(self, printThird):
        """
        :type printThird: method
        :rtype: void
        """

        # printThird() outputs "third". Do not change or remove this line.
        printThird()
```

Try

1. Take Input: nums = [1,2,2] and verify the results.
2. Take Input: nums = [2,2,2] and verify the results.

7.7 Prime Number Finder

We will create a class called PrimeNumberThread that derives from the Thread class of the threading library. This class/thread will be used to find very large prime numbers from any given starting position (e.g. Checking any number greater than 100,000,000,000,000 one by one to see if they are a prime number or not).

We will then run three threads concurrently, each thread starting with a different starting position .e.g.

Thread 1 will look up for prime numbers, starting from number 100,000,000,000,000
Thread 2 will look up for prime numbers, starting from number 300,000,000,000,000
Thread 3 will look up for prime numbers, starting from number 500,000,000,000,000

Hints:

```
#Introduction to Multi-Threading
import threading
import time

class PrimeFinderThread(threading.Thread):
  def __init__(self, id,startPos):
    # Calling parent class constructor
    threading.Thread.__init__(self)
```

```
    self.id = id
    self.startPos = startPos
  ……
#Main Program Starts Here...
#Let's intialise three different threads.Each thread will be used to dientify
prime numbers starting with a different starting position
thread1 = PrimeFinderThread(1,100000000000000)
thread2 = PrimeFinderThread(2,300000000000000)
thread3 = PrimeFinderThread(3,500000000000000)

#Let's start our three threads to implement concurrent processing!
thread1.start()
thread2.start()
thread3.start()
```

Try

1. Write a program considering input as four threads in to implement concurrent processing.
2. Write a program considering input as five threads in to implement concurrent processing.

7.8 Regular expression in Python

Regular Expressions (RegEx) is a special sequence of characters that uses a search pattern to find a string or set of strings. It can detect the presence or absence of a text by matching it with a particular pattern, and also can split a pattern into one or more sub-patterns. Python provides a **re** module that supports the use of regex in Python. Its primary function is to offer a search, where it takes a regular expression and a string. Here, it either returns the first match or else none.

Python regex methods

The Python regex module consists of multiple methods. Below is the list of regex methods and their meaning.

Python regex methods

Method	Description
re.compile('pattern')	Compile a regular expression pattern provided as a string into a re.Pattern object.
re.search(pattern, str)	Search for occurrences of the regex pattern inside the target string and return only the first match.
re.match(pattern, str)	Try to match the regex pattern at the start of the string. It returns a match only if the pattern is located at the beginning of the string.
re.fullmatch(pattern, str)	Match the regular expression pattern to the entire string from the first to the last character.
re.findall(pattern, str)	Scans the regex pattern through the entire string and returns all matches.
re.finditer(pattern, str)	Scans the regex pattern through the entire string and returns an iterator yielding match objects.
re.split(pattern, str)	It breaks a string into a list of matches as per the given regular expression pattern.
re.sub(pattern, replacement, str)	Replace one or more occurrences of a pattern in the string with a replacement.

re.subn(pattern, replacement, str)	Same as re.sub(). The difference is it will return a tuple of two elements.
	First, a new string after all replacement, and second the number of replacements it has made.

Hints

```
# import the RE module
import re

target_string = "Jessa salary is 8000$"

# compile regex pattern
# pattern to match any character
str_pattern = r"\w"
pattern = re.compile(str_pattern)

# match regex pattern at start of the string
res = pattern.match(target_string)
# match character
print(res.group())
# Output 'J'

# search regex pattern anywhere inside string
# pattern to search any digit
res = re.search(r"\d", target_string)
print(res.group())
# Output 8

# pattern to find all digits
res = re.findall(r"\d", target_string)
print(res)
# Output ['8', '0', '0', '0']

# regex to split string on whitespaces
res = re.split(r"\s", target_string)
print("All tokens:", res)
# Output ['Jessa', 'salary', 'is', '8000$']

# regex for replacement
# replace space with hyphen
res = re.sub(r"\s", "-", target_string)
# string after replacement:
print(res)
# Output Jessa-salary-is-8000$
```

7.9 Regular Expressions in Python (Search, Match and Find All)

1. Searching an occurrence of pattern

re.search() : This method either returns None (if the pattern doesn't match), or a re.MatchObject that contains information about the matching part of the string. This method stops after the first match, so this is best suited for testing a regular expression more than extracting data.

```
# A Python program to demonstrate working of re.match().
import re

# Lets use a regular expression to match a date string
# in the form of Month name followed by day number
```

```
regex = r"([a-zA-Z]+) (\d+)"

match = re.search(regex, "I was born on June 24")

if match != None:

    # We reach here when the expression "([a-zA-Z]+) (\d+)"
    # matches the date string.

    # This will print [14, 21), since it matches at index 14
    # and ends at 21.
    print ("Match at index %s, %s" % (match.start(), match.end()))

    # We us group() method to get all the matches and
    # captured groups. The groups contain the matched values.
    # In particular:
    # match.group(0) always returns the fully matched string
    # match.group(1) match.group(2), ... return the capture
    # groups in order from left to right in the input string
    # match.group() is equivalent to match.group(0)

    # So this will print "June 24"
    print ("Full match: %s" % (match.group(0)))

    # So this will print "June"
    print ("Month: %s" % (match.group(1)))

    # So this will print "24"
    print ("Day: %s" % (match.group(2)))

else:
    print ("The regex pattern does not match.")
```

Output:
Match at index 14, 21
Full match: June 24
Month: June
Day: 24

2. Matching a Pattern with Text
re.match() : This function attempts to match pattern to whole string. The re.match function returns a match object on success, None on failure.

```
# A Python program to demonstrate working of
# findall()
import re

# A sample text string where regular expression
# is searched.
string  = """Hello my Number is 123456789 and
             my friend's number is 987654321"""

# A sample regular expression to find digits.
regex = '\d+'

match = re.findall(regex, string)
print(match)
```

Output:

Match at index 14, 21
Full match: June 24
Month: June
Day: 24

3. Finding all occurrences of a pattern

re.findall() : Return all non-overlapping matches of pattern in string, as a list of strings. The string is scanned left-to-right, and matches are returned in the order found (Source: Python Docs).

```
# A Python program to demonstrate working of
# findall()
import re

# A sample text string where regular expression
# is searched.
string  = """Hello my Number is 123456789 and
             my friend's number is 987654321"""

# A sample regular expression to find digits.
regex = '\d+'

match = re.findall(regex, string)
print(match)
```

Output:
['123456789', '987654321']

TRY

1. Check that a string contains only a certain set of characters (in this case a-z, A-Z and 0-9).

2. Search for literal strings within a string given below.
   Sample text : 'The quick brown fox jumps over the lazy dog.'
   Searched words : 'fox', 'dog', 'horse'

3. Search for literal strings within a string given below.
   Sample text : 'The quick brown fox jumps over the lazy dog.'
   Searched words : 'fox'

7. Python JSON

7.1 Working with JSON Data in Python

JSON JavaScript Object Notation is a format for structuring data. It is mainly used for storing and transferring data between the browser and the server. Python too supports JSON with a built-in package called json. This package provides all the necessary tools for working with JSON Objects including parsing, serializing, deserializing, and many more.

JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called JSON. To use this feature, we import the JSON package in Python script. The text in JSON is done through quoted-string which contains the value in key-value mapping within { }. It is similar to the dictionary in Python. JSON shows an API similar to users of Standard Library marshal and pickle modules and Python natively supports JSON features. For example

Hints:

```
EXAMPLE 1:
# Python program showing
# use of json package

import json

# {key:value mapping}
a ={"name":"John",
   "age":31,
    "Salary":25000}

# conversion to JSON done by dumps() function
 b = json.dumps(a)

# printing the output
print(b)
```

Example1 Output:

{"age": 31, "Salary": 25000, "name": "John"}

```
EXAMPLE 2:
# Python program showing that
# json support different primitive
# types

import json

# list conversion to Array
print(json.dumps(['Welcome', "to", "GeeksforGeeks"]))
# tuple conversion to Array
print(json.dumps(("Welcome", "to", "GeeksforGeeks")))

# string conversion to String
print(json.dumps("Hi"))

# int conversion to Number
print(json.dumps(123))

# float conversion to Number
print(json.dumps(23.572))

# Boolean conversion to their respective values
print(json.dumps(True))
print(json.dumps(False))

# None value to null
print(json.dumps(None))
```

Example2 Output:

["Welcome", "to", "GeeksforGeeks"]
["Welcome", "to", "GeeksforGeeks"]
"Hi"
123
23.572
true
false
null

Try

1. Write a python program to showing that json support different primitive by taking input as ['Hearty Welcome', "to", "IARE College"])).

7.2 Read JSON files using Python

The full form of JSON is JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called JSON. To use this feature, we import the JSON package in Python script. The text in JSON is done through quoted-string which contains the value in key-value mapping within { }.

Python Parse JSON – How to Read a JSON File

It's pretty easy to load a JSON object in Python. Python has a built-in package called JSON, which can be used to work with JSON data. It's done by using the JSON module, which provides us with a lot of methods which among loads() and load() methods are gonna help us to read the JSON file.

Deserialize a JSON String to an Object in Python

The Deserialization of JSON means the conversion of JSON objects into their respective Python objects. The load()/loads() method is used for it. If you have used JSON data from another program or obtained it as a string format of JSON, then it can easily be deserialized with load()/loads(), which is usually used to load from string, otherwise, the root object is in list or dict. See the following table given below.

JSON OBJECT	PYTHON OBJECT
object	dict
array	list
string	str
null	None
number (int)	int
number (real)	float
TRUE	TRUE
FALSE	FALSE

1. json.load() method

The json.load() accepts the file object, parses the JSON data, populates a Python dictionary with the data, and returns it back to you.

Syntax:

json.load(file object)
Parameter: It takes the file object as a parameter.
Return: It return a JSON Object.

Loading a JSON File in Python
Here we are going to read a JSON file named **data.json** the screenshot of the file is given below.
Emp_details: [{
“emp_name”:”Shubhan”,
“email”:”ksingh.shbh@gmail.com”
“job_profile”:intern”
}
{
“emp_name”:”Gaurav”,
“email”:”gaurav.singh@gmail.com”,
“job_profile”:developer”
}
{
“emp_name”:”Nikhil”,
“email”:”Nikhil@greeksforgreeks.org”,
“job_profile”:Full Time”
}]

Hints:

```
# Python program to read
# json file

import json

# Opening JSON file
f = open('data.json')

# returns JSON object as
# a dictionary
data = json.load(f)

# Iterating through the json
# list
for i in data['emp_details']:
    print(i)

# Closing file
f.close()
```

Output:

```
{'emp_name': 'Shubham', 'email': 'ksingh.shubh@gmail.com', 'job_profile': 'intern'}
{'emp_name': 'Gaurav', 'email': 'gaurav.singh@gmail.com', 'job_profile': 'developer'}
{'emp_name': 'Nikhil', 'email': 'nikhil@geeksforgeeks.org', 'job_profile': 'Full Time'}
```

Try
1. Write a program to read student details a JSON input file and verify the results.

2. json.loads() Method

If we have a JSON string, we can parse it by using the json.loads() method. json.loads() does not take the file path, but the file contents as a string, to read the content of a JSON file we can use fileobject.read() to convert the file into a string and pass it with json.loads(). This method returns the content of the file.

Syntax:
json.loads(S)
Parameter: it takes a string, bytes, or byte array instance which contains the JSON document as a parameter (S).
Return Type: It returns the Python object.
Python – Read JSON String

```
# Python program to read
# json file

import json

# JSON string
j_string = '{"name": "Bob", "languages": "English"}'

# deserializes into dict and returns dict.
y = json.loads(j_string)

print("JSON string = ", y)
print()

# JSON file
f = open ('data.json', "r")

# Reading from file
data = json.loads(f.read())

# Iterating through the json list
for i in data['emp_details']:
    print(i)

# Closing file
f.close()
```

Output:

```
JSON string =  {'name': 'Bob', 'languages': 'English'}

{'emp_name': 'Shubham', 'email': 'ksingh.shubh@gmail.com', 'job_profile': 'intern'}
{'emp_name': 'Gaurav', 'email': 'gaurav.singh@gmail.com', 'job_profile': 'developer'}
{'emp_name': 'Nikhil', 'email': 'nikhil@geeksforgeeks.org', 'job_profile': 'Full Time'}
```

Try
1. Read student details a JSON input file and verify the results.

7.3 Reading and Writing JSON to a File in Python

Writing JSON to a file in Python
Serializing JSON refers to the transformation of data into a series of bytes (hence serial) to be stored or transmitted across a network. To handle the data flow in a file, the JSON library in Python uses dump() or dumps() function to convert the Python objects into their respective JSON object, so it makes it easy to write data to files. See the following table given below.

PYTHON OBJECT	JSON OBJECT
Dict	object
list, tuple	array
str	string
int, long, float	numbers
TRUE	TRUE
FALSE	FALSE
None	null

Method 1: Writing JSON to a file in Python using json.dumps()
The JSON package in Python has a function called json.dumps() that helps in converting a dictionary to a JSON object. It takes two parameters:

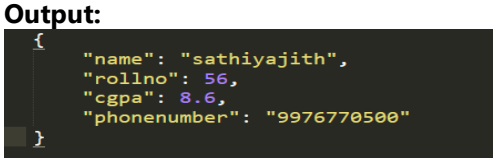
- **dictionary –** the name of a dictionary which should be converted to a JSON object.
- **indent –** defines the number of units for indentation

After converting the dictionary to a JSON object, simply write it to a file using the "write" function.

Hints:

```
import json

# Data to be written
dictionary = {
    "name": "sathiyajith",
    "rollno": 56,
    "cgpa": 8.6,
    "phonenumber": "9976770500"
}

# Serializing json
json_object = json.dumps(dictionary, indent=4)

# Writing to sample.json
with open("sample.json", "w") as outfile:
    outfile.write(json_object)
```

Output:



Method 2: Writing JSON to a file in Python using json.dump()

Another way of writing JSON to a file is by using json.dump() method The JSON package has the "dump" function which directly writes the dictionary to a file in the form of JSON, without needing to convert it into an actual JSON object. It takes 2 parameters:

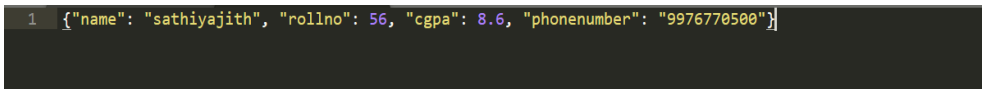
- **dictionary –** the name of a dictionary which should be converted to a JSON object.
- **file pointer –** pointer of the file opened in write or append mode.

Hints:

```
# Python program to write JSON
```

```
# to a file

import json

# Data to be written
dictionary = {
    "name": "sathiyajith",
    "rollno": 56,
    "cgpa": 8.6,
    "phonenumber": "9976770500"
}

with open("sample.json", "w") as outfile:
    json.dump(dictionary, outfile)
```

Output:



Reading JSON from a file using Python
Deserialization is the opposite of Serialization, i.e. conversion of JSON objects into their respective Python objects. The load() method is used for it. If you have used JSON data from another program or obtained it as a string format of JSON, then it can easily be deserialized with load(), which is usually used to load from a string, otherwise, the root object is in a list or Dict.

Reading JSON from a file using json.load()
The JSON package has json.load() function that loads the JSON content from a JSON file into a dictionary. It takes one parameter:

- **File pointer:** A file pointer that points to a JSON file.

Hints:

```
import json

# Opening JSON file
with open('sample.json', 'r') as openfile:

    # Reading from json file
    json_object = json.load(openfile)

print(json_object)
print(type(json_object))
```

Output:

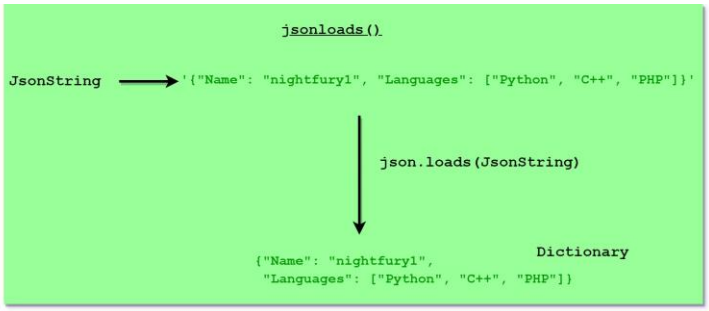
```
E:\MATERIALS\content_writing\JSON>python reading_JSON.py
{'name': 'sathiyajith', 'rollno': 56, 'cgpa': 8.6, 'phonenumber': '9976770500'}
<class 'dict'>
```

7.4 Parse Data from JSON into Python

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write for machines to parse and generate. Basically, it is used to represent data in a specified format to access and work with data easily. Here we will learn, how to create and parse data from JSON and work with it.

Before starting the details of parsing data, We should know about ***'json'*** module in Python. It provides an API that is similar to pickle for converting in-memory objects in Python to a serialized representation as well as makes it easy to parse JSON data and files. Here are some ways to parse data from JSON using Python below:

- **Python JSON to Dictionary:** With the help of ***json.loads()*** function, we can parse JSON objects to dictionary.



Hints:

```
# importing json library
import json

geek = '{"Name": "nightfury1", "Languages": ["Python", "C++", "PHP"]}'
geek_dict = json.loads(geek)

# printing all elements of dictionary
print("Dictionary after parsing: ", geek_dict)

# printing the values using key
print("\nValues in Languages: ", geek_dict['Languages'])
```

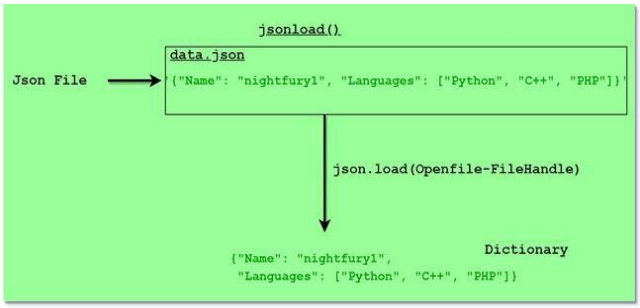
Output:
Dictionary after parsing: {'Name': 'nightfury1', 'Languages': ['Python', 'C++', 'PHP']}
Values in Languages: ['Python', 'C++', 'PHP']

Python JSON to Ordered Dictionary: We have to use same *json.loads()* function for parsing the objects, but for getting in ordered, we have to add keyword '*object_pairs_hook=OrderedDict*' from *collections* module.

Hints:

```
import json
from collections import OrderedDict

#create Ordered Dictionary using keyword
# 'object_pairs_hook=OrderDict'
data = json.loads('{"GeeksforGeeks":1, "Gulshan": 2, "nightfury_1": 3, "Geek":
4}',
                  object_pairs_hook=OrderedDict)
print("Ordered Dictionary: ", data)
```

Parse using JSON file: With the help of *json.load()* method, we can parse JSON objects to dictionary format by opening the required JSON file.



Hints:

```
# importing json library
import json

with open('data.json') as f:
  data = json.load(f)
# printing data after loading the json file
print(data)
```

Output:

{'Name': 'nightfury1', 'Language': ['Python', 'C++', 'PHP']}

Try

1. Access the value of key2 from the JSON
2. Sort JSON keys in and write them into a file
3. Convert the JSON into Vehicle Object
4. Convert Python objects into JSON strings. Print all the values.
5. Convert Python dictionary object (sort by key) to JSON data. Print the object members with indent level 4.
6. Convert JSON encoded data into Python objects.

8. Python NumPy

8.1 Finding 'n' Fibonacci numbers.

All of us are familiar with Fibonacci Series. Each number in the sequence is the sum of the two numbers that precede it. So, the sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…… In this tutorial, we will implement the same using NumPy with the aid of Binet formula.

Binet Formula

$$\text{fn} = \left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right] * \frac{1}{\sqrt{5}}$$
$$\text{where, alpha} = \left(\frac{1+\sqrt{5}}{2}\right), beta = \left(\frac{1-\sqrt{5}}{2}\right)$$

'n' is the parameter which relates the first 'n' numbers of Fibonacci Series. In the first example we are going to findout first 10 numbers of Fibonacci Series (n = 10), after that we takes the parameter 'n' from user and produce the corresponding result.

Hints:

```
import numpy as np
```

```
# We are creating an array contains n elements
# for getting first 'n' Fibonacci numbers
fNumber = int(input("Enter the value of n + 1'th number : "))
a = np.arange(1, fNumber)
….

print("The first {} numbers of Fibonacci series are {} . ".format(length_a, Fn))
```

Output:

```
# Here user input was 10
Enter the value of n+1'th number :10
The first 9 numbers of Fibonacci series are [ 1.  1.  2.  3.  5.  8. 13. 21. 34.]
```

Try

1. Take input value as 50 and verify the results.
2. Take input value as 100 and verify the results.

8.2 Matrix Multiplication in NumPy

Let us see how to compute matrix multiplication with NumPy. We will be using the numpy.dot() method to find the product of 2 matrices.
For example, for two matrices A and B.
A = [[1, 2], [2, 3]]
B = [[4, 5], [6, 7]]
So, A.B = [[1*4 + 2*6, 2*4 + 3*6], [1*5 + 2*7, 2*5 + 3*7]
So the computed answer will be: [[16, 26], [19, 31]]

Hints:

```
# importing the module
import numpy as np

# creating two matrices
p = [[1, 2], [2, 3]]
q = [[4, 5], [6, 7]]
….
print(result)
```

Output:

```
Matrix p:
[[1, 2], [2, 3]]
Matrix q:
[[4, 5], [6, 7]]
The matrix multiplication is:
[[16 19]
[26 31]]
```

Try

1. Take input A = [[1, 5], [5, 10]] B = [[14, 15], [61, 71]] and verify the results.
2. Take input A = [[10, 50], [50, 100]] B = [[140, 105], [601, 701]] and verify the results.
3. Take input A = [[31, 25], [55, 103]] B = [[134, 175], [61, 751]] and verify the results.

8.3 Numpy Indexing

Given numpy array, the task is to replace negative value with zero in numpy array

Hints:

```
# Python code to demonstrate
# to replace negative value with 0
import numpy as np

ini_array1 = np.array([1, 2, -3, 4, -5, -6])
……

# printing result
print("New resulting array: ", ini_array1)
```

Output:

```
initial array [ 1 2 -3 4 -5 -6]

New resulting array:[1 2 0 4 0 0]
```

Try

1. Take input array1 = np.array ([11, 12, -13, 14, -15, -16]) and verify the results.
2. Take input array1 = np.array ([101, 102, -103, 140, -105, -106]) and verify the results.

8.4 NumPy Linear Algebra

Write a Program to get the random samples of geometric distribution and return the random samples of numpy array by using **numpy.random.geometric()** method.

$$f(k) = (1-p)^{k-1}p$$

Syntax: *numpy.random.geometric(p, size=None)*
Return: *Return the random samples of numpy array.*

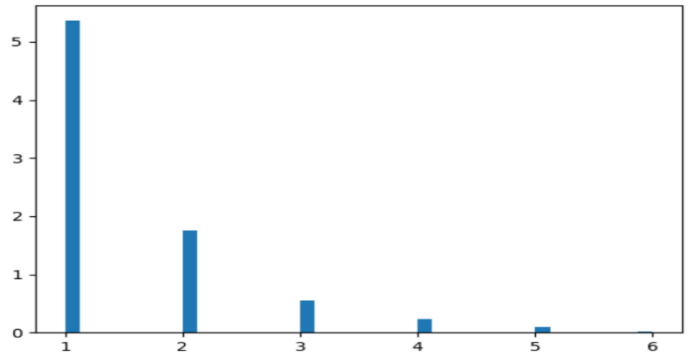
Example #1:

In this example we can see that by using **numpy.random.geometric()** method, we are able to get the random samples of geometric distribution and return the random samples as numpy array by using this method.

Hints:

```
# import numpy and geometric
import numpy as np
import matplotlib.pyplot as plt

# Using geometric() method
#Write the code here
plt.show()
```

Output:



8.5 NumPy Sorting and Searching

Write a program to find the k number of the smallest values from a NumPy array.

Examples:
Input: [1,3,5,2,4,6]
k = 3
Output: [1,2,3]

Write a Program to get the random samples of geometric distribution and return the random samples of numpy array by using numpy.random.geometric() method.

Method 1: Using np.sort() .
Approach:

Create a NumPy array.
Determine the value of k.
Sort the array in ascending order using the sort() method.
Print the first k values of the sorted array.

Method 2: Using np.argpartition()
Approach:

1. Create a NumPy array.
2. Determine the value of k.
3. Get the indexes of the smallest k elements using the argpartition() method.
4. Fetch the first k values from the array obtained from argpartition() and print their index values with respect to the original array.

Hints:

```
# importing the module
import numpy as np

# creating the array
arr = np.array([23, 12, 1, 3, 4, 5, 6])
print("The Original Array Content")
print(arr)

# value of k
k = 4
```

```
# using np.argpartition()
result = np.argpartition(arr, k)

# k smallest number of array
print(k, "smallest elements of the array")
print(arr[result[:k]])
```

Output:
The Original Array Content
[23 12 1 3 4 5 6]
4 smallest elements of the array
[1 3 4 5]
TRY
1. Take input: [1, -1, 3, 0, 5, -2, 4, -6] and verify the results.
2. Take input: [2, -2, 3, 5, 8, -2, 9, -10] and verify the results.

8.6 NumPy Mathematics

Evaluate Einstein's summation convention of two multidimensional NumPy arrays.
Syntax: numpy.einsum(subscripts, *operands, out=None)

Parameters:
subscripts: str

Specifies the subscripts for summation as comma separated list of subscript labels. An implicit (classical Einstein summation) calculation is performed unless the explicit indicator '->' is included as well as subscript labels of the precise output form.
operands: list of array like
These are the arrays for the operation.
out: ndarray, optional

If provided, the calculation is done into this array.
Returns: The calculation based on the Einstein summation convention.

Hints:

```
Einstein's summation convention of two 2X2 matrices
# Importing library
import numpy as np

# Creating two 2X2 matrix
matrix1 = np.array([[1, 2], [0, 2]])
matrix2 = np.array([[0, 1], [3, 4]])

print("Original matrix:")
print(matrix1)
print(matrix2)

# Output
result = np.einsum("mk,kn", matrix1, matrix2)

print("Einstein's summation convention of the two matrix:")
```

```
print(result)
```

Output:
Original matrix:
[[1 2]
 [0 2]]
[[0 1]
 [3 4]]

Einstein's summation convention of the two matrix:
[[6 9]
[6 8]]

8.7 NumPy Statistics

Calculate the average, variance and standard deviation in Python using NumPy. **Numpy** in Python is a general-purpose array-processing package. It provides a high-performance multidimensional array object and tools for working with these arrays. It is the fundamental package for scientific computing with Python. Numpy provides very easy methods to calculate the average, variance, and standard deviation.

Average
Average a number expressing the central or typical value in a set of data, in particular the mode, median, or (most commonly) the mean, which is calculated by dividing the sum of the values in the set by their number.

Average in Python Using Numpy:
One can calculate the average by using **numpy.average()** function in python.

Syntax:
numpy.average(a**,** axis=None**,** weights=None**,** returned=False**)**

Parameters:
a: Array containing data to be averaged
axis: Axis or axes along which to average a
weights: An array of weights associated with the values in a
returned: Default is False. If True, the tuple is returned, otherwise only the average is returned

Hints:

```
# Python program to get average of a list
# Importing the NumPy module
import numpy as np
#Write the code here
```

Try
1. Find the number of occurrences of a sequence in a NumPy array
2. Find the most frequent value in a NumPy array
3. Combining a one and a two-dimensional NumPy Array
4. Build an array of all combinations of two NumPy arrays
5. Flatten a Matrix in Python using NumPy
6. Flatten a 2d numpy array into 1d array
7. Move axes of an array to new positions
8. Interchange two axes of an array
9. Counts the number of non-zero values in the array
10. Count the number of elements along a given axis
11. Trim the leading and/or trailing zeros from a 1-D array
12. Change data type of given numpy array
13. Reverse a numpy array

14. Count the number of elements along a given axis
15. Trim the leading and/or trailing zeros from a 1-D array
16. Change data type of given numpy array
17. Get the eigen values of a matrix
18. Multiply matrices of complex numbers using NumPy in Python
19. Compute the outer product of two given vectors using NumPy in Python
20. Calculate inner, outer, and cross products of matrices and vectors using NumPy
21. Compute the covariance matrix of two given NumPy arrays
22. Convert covariance matrix to correlation matrix using Python
23. Compute the Kronecker product of two multidimension NumPy arrays
24. Convert the matrix into a list.

9. PYTHON PANDAS

9.1 Calculate Special Bonus

Table: Employees

```
+--------------+---------+
| Column Name | Type    |
+--------------+---------+
| employee_id | int      |
| name         | varchar  |
| salary       | int      |
+-------------+---------+
```

employee_id is the primary key (column with unique values) for this table.
Each row of this table indicates the employee ID, employee name, and salary.

Write a solution to calculate the bonus of each employee. The bonus of an employee is 100% of their salary if the ID of the employee is an odd number and the employee's name does not start with the character 'M'. The bonus of an employee is 0 otherwise return the result table ordered by employee_id.

The result format is in the following example.
Example 1:
Input:
Employees table:

```
+-------------+---------+--------+
| employee_id | name    | salary |
+-------------+---------+--------+
| 2           | Meir    | 3000   |
| 3           | Michael | 3800   |
| 7           | Addilyn | 7400   |
| 8           | Juan    | 6100   |
| 9           | Kannon  | 7700   |
+-------------+---------+--------+
```

Hints:

```
import pandas as pd

def calculate_special_bonus(employees: pd.DataFrame) -> pd.DataFrame:
```

Output:
Output:

```
+-------------+-------+
| employee_id | bonus |
+-------------+-------+
```

```
| 2           | 0    |
| 3           | 0    |
| 7           | 7400 |
| 8           | 0    |
| 9           | 7700 |
+-------------+-------+
```

Explanation:
The employees with IDs 2 and 8 get 0 bonus because they have an even employee_id.
The employee with ID 3 gets 0 bonus because their name starts with 'M'.
The rest of the employees get a 100% bonus.

9.2 Daily Leads and Partners

Table: DailySales

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| date_id     | date    |
| make_name   | varchar |
| lead_id     | int     |
| partner_id  | int     |
+-------------+---------+
```

There is no primary key (column with unique values) for this table. It may contain duplicates.
This table contains the date and the name of the product sold and the IDs of the lead and partner it was sold to.
The name consists of only lowercase English letters.

For each date_id and make_name, find the number of **distinct** lead_id's and **distinct** partner_id's.
Return the result table in **any order**.
The result format is in the following example.

Example 1:
Input:
DailySales table:

```
+-----------+-----------+---------+------------+
| date_id   | make_name | lead_id | partner_id |
+-----------+-----------+---------+------------+
| 2020-12-8 | toyota    | 0       | 1          |
| 2020-12-8 | toyota    | 1       | 0          |
| 2020-12-8 | toyota    | 1       | 2          |
| 2020-12-7 | toyota    | 0       | 2          |
| 2020-12-7 | toyota    | 0       | 1          |
| 2020-12-8 | honda     | 1       | 2          |
| 2020-12-8 | honda     | 2       | 1          |
| 2020-12-7 | honda     | 0       | 1          |
| 2020-12-7 | honda     | 1       | 2          |
| 2020-12-7 | honda     | 2       | 1          |
+-----------+-----------+---------+------------+
```

Output:

```
+-----------+-----------+--------------+-----------------+
| date_id   | make_name | unique_leads | unique_partners |
+-----------+-----------+--------------+-----------------+
```

```
| 2020-12-8 | toyota  | 2          | 3              |
| 2020-12-7 | toyota  | 1          | 2              |
| 2020-12-8 | honda   | 2          | 2              |
| 2020-12-7 | honda   | 3          | 2              |
+-----------+----------+-------------+----------------+
```

Explanation:
For 2020-12-8, toyota gets leads = [0, 1] and partners = [0, 1, 2] while honda gets leads = [1, 2] and partners = [1, 2].
For 2020-12-7, toyota gets leads = [0] and partners = [1, 2] while honda gets leads = [0, 1, 2] and partners = [1, 2].

Hints:

```
import pandas as pd

def daily_leads_and_partners(daily_sales: pd.DataFrame) -> pd.DataFrame:
```

9.3 Game Play Analysis I

Activity

```
+--------------+---------+
| Column Name  | Type    |
+--------------+---------+
| player_id    | int     |
| device_id    | int     |
| event_date   | date    |
| games_played | int     |
+--------------+---------+
```

(player_id, event_date) is the primary key (combination of columns with unique values) of this table.
This table shows the activity of players of some games.
Each row is a record of a player who logged in and played a number of games (possibly 0) before logging out on someday using some device.

Write a solution to find the **first login date** for each player.
Return the result table in **any order**.
The result format is in the following example.

Example 1:
Input:
Activity table:

```
+-----------+-----------+------------+--------------+
| player_id | device_id | event_date | games_played |
+-----------+-----------+------------+--------------+
| 1         | 2         | 2016-03-01 | 5            |
| 1         | 2         | 2016-05-02 | 6            |
| 2         | 3         | 2017-06-25 | 1            |
| 3         | 1         | 2016-03-02 | 0            |
| 3         | 4         | 2018-07-03 | 5            |

+-----------+-----------+------------+--------------+
```

Output:

```
+-----------+-------------+
| player_id | first_login |
+-----------+-------------+
| 1         | 2016-03-01  |
| 2         | 2017-06-25  |
| 3         | 2016-03-02  |
+-----------+-------------+
```

Hints:

```
import pandas as pd

def game_analysis(activity: pd.DataFrame) -> pd.DataFrame:
```

9.4 Article Views I

Table: Views

```
+---------------+---------+
| Column Name   | Type    |
+---------------+---------+
| article_id    | int     |
| author_id     | int     |
| viewer_id     | int     |
| view_date     | date    |
+---------------+---------+
```

There is no primary key (column with unique values) for this table, the table may have duplicate rows.
Each row of this table indicates that some viewer viewed an article (written by some author) on some date.
Note that equal author_id and viewer_id indicate the same person.

Write a solution to find all the authors that viewed at least one of their own articles.
Return the result table sorted by id in ascending order.
The result format is in the following example.

Example 1:
Input:
Views table:

```
+------------+-----------+-----------+------------+
| article_id | author_id | viewer_id | view_date  |
+------------+-----------+-----------+------------+
| 1          | 3         | 5         | 2019-08-01 |
| 1          | 3         | 6         | 2019-08-02 |
| 2          | 7         | 7         | 2019-08-01 |
| 2          | 7         | 6         | 2019-08-02 |
| 4          | 7         | 1         | 2019-07-22 |
| 3          | 4         | 4         | 2019-07-21 |
| 3          | 4         | 4         | 2019-07-21 |
+------------+-----------+-----------+------------+
```

Output:

```
+------+
| id   |
+------+
| 4    |
| 7    |
+------+
```

Hints:

```
import pandas as pd

def article_views(views: pd.DataFrame) -> pd.DataFrame:
```

9.5 Count Salary Categories

Table: Accounts

```
+-------------+------+
| Column Name | Type |
+-------------+------+
| account_id  | int  |
| income      | int  |
+-------------+------+
```

account_id is the primary key (column with unique values) for this table.
Each row contains information about the monthly income for one bank account.

Write a solution to calculate the number of bank accounts for each salary category. The salary categories are:

- "Low Salary": All the salaries **strictly less** than $20000.
- "Average Salary": All the salaries in the **inclusive** range [$20000, $50000].
- "High Salary": All the salaries **strictly greater** than $50000.

The result table **must** contain all three categories. If there are no accounts in a category, return 0.
Return the result table in **any order**.
The result format is in the following example.

Example 1:
Input:
Accounts table:

```
+------------+--------+
| account_id | income |
+------------+--------+
| 3          | 108939 |
| 2          | 12747  |
| 8          | 87709  |
| 6          | 91796  |
+------------+--------+
```

Output:

```
+---------------+---------------+
| category      | accounts_count |
+---------------+---------------+
| Low Salary    | 1             |
| Average Salary | 0            |
| High Salary   | 3             |
+---------------+---------------+
```

Explanation:
Low Salary: Account 2.
Average Salary: No accounts.
High Salary: Accounts 3, 6, and 8.

Hints:

```
import pandas as pd

def count_salary_categories(accounts: pd.DataFrame) -> pd.DataFrame:
```

9.6 Managers with at Least 5 Direct Reports

Table: Employee

```
+-------------+---------+
| Column Name | Type    |
+-------------+---------+
| id          | int     |
| name        | varchar |
| department  | varchar |
| managerId   | int     |
+-------------+---------+
```

- id is the primary key (column with unique values) for this table.
- Each row of this table indicates the name of an employee, their department, and the id of their manager.
- If managerId is null, then the employee does not have a manager.
- No employee will be the manager of themself.

Write a solution to find managers with at least **five direct reports**.
Return the result table in **any order**.
The result format is in the following example.

Example 1:
Input:
Employee table:

```
+-----+-------+------------+-----------+
| id  | name  | department | managerId |
+-----+-------+------------+-----------+
| 101 | John  | A          | None      |
| 102 | Dan   | A          | 101       |
| 103 | James | A          | 101       |
| 104 | Amy   | A          | 101       |
| 105 | Anne  | A          | 101       |
| 106 | Ron   | B          | 101       |
+-----+-------+------------+-----------+
```

Output:

```
+------+
| name |
+------+
| John |
+------+
```

Hints:

```
import pandas as pd

def find_managers(employee: pd.DataFrame) -> pd.DataFrame:
```

10. Python Database

10.1 Connect to MySQL Database in Python

Let's see how to connect the MySQL database in Python using the 'MySQL Connector Python' module.
Arguments required toconnect.

You need to know the following detail of the MySQL server to perform the connection from Python.

Argument	Description
Username	The username that you use to work with MySQL Server. The default username for the MySQL database is a **root**.
Password	Password is given by the user at the time of installing the MySQL server. If you are using root then you won't need the password.
Host name	The server name or Ip address on which MySQL is running. if you are running on localhost, then you can use **localhost** or its IP 127.0.0.0
Database name	The name of the database to which you want to connect and perform the operations.

How to Connect to MySQL Database in Python

1. **Install MySQL connector module**
   Use the pip command to install MySQL connector Python.
   pip install mysql-connector-python

2. **Import MySQL connector module**
   Import using a import mysql.connector statement so you can use this module's methods to communicate with the MySQL database.

3. **Use the connect() method**
   Use the connect() method of the MySQL Connector class with the required arguments to connect MySQL. It would return a MySQLConnection object if the connection established successfully.

4. **Use the cursor() method**
   Use the cursor() method of a MySQLConnection object to create a cursor object to perform various SQL operations.

5. **Use the execute() method**
   The execute() methods run the SQL query and return the result.

6. **Extract result using fetchall()**
   Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

7. **Close cursor and connection objects**
   use cursor.clsoe() and connection.clsoe() method to close open connections after your work completes

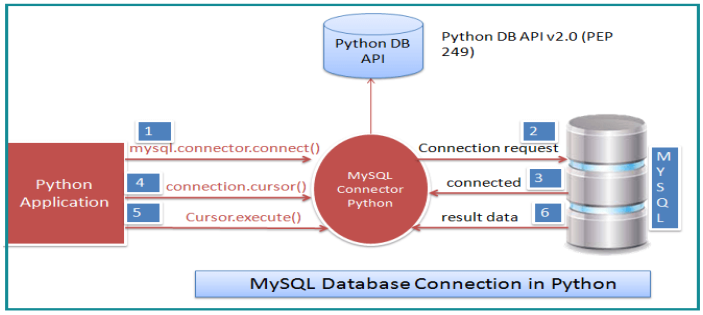


Fig: MySQL database connection in Python

Run the below query on the MySQL console if you have not created any database in MySQL. Otherwise, you can skip the below query.

Hints:

```
Create Database in MySQL
Create database Electronics;
Example to connect to MySQL Database in Python
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Electronics',
                                         user='pynative',
                                         password='pynative@#29')
except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Output.
Connected to MySQL Server version 5.7.19
You're connected to database: ('electronics',)
MySQL connection is closed.

10.2 Create MySQL table from Python

Now you know how to connect to a MySQL server from Python, in this section, we will learn how to create a table in MySQL from Python. Let's create table 'Laptop' under the 'Electronics' database.

Hints:

```
import mysql.connector

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Electronics',
                                         user='pynative',
```

```
                                        password='pynative@#29')

    mySql_Create_Table_Query = """CREATE TABLE Laptop (
                               Id int(11) NOT NULL,
                               Name varchar(250) NOT NULL,
                               Price float NOT NULL,
                               Purchase_date Date NOT NULL,
                               PRIMARY KEY (Id)) """

    cursor = connection.cursor()
    result = cursor.execute(mySql_Create_Table_Query)
    print("Laptop Table created successfully ")

except mysql.connector.Error as error:
    print("Failed to create table in MySQL: {}".format(error))

finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

Output:

Laptop Table created successfully
MySQL connection is closed

10.3 Python MySQL Connection arguments list

We already discussed the four mandatory arguments required to connect the MySQL Server.
Let see what other connection arguments we can use to communicate with the MySQL server from Python. Below is the list of all other connection arguments and their meaning.

- port: The TCP/IP port of the MySQL server. This value must be an integer. We can specify the different port number if the MySQL server is listening to a different port. The default value for this port argument is 3306.
- use_unicode: Specify whether to use Unicode or not. The default value is True.
- charset: MySQL character set to use, character set variables relate to a client's interaction with the server. There are almost 30 to 40 charset MySQL server supports. The default value of the charset argument is "utf8".
- auto-commit: Set it to true if you want to auto-commit transactions. If you wish to manage transactions in MySQL from Python, you need to set this value true or false. The default value is False, i.e., the changes are not committed to the database. You need to explicitly call a commit method to persist your changes in the database.
- get_warnings: To fetch warning, this is helpful to know the connection is established but with warnings. The default value is False.
- raise_on_warnings: Set it when you want to raise an exception on warnings. The Default value is False.
- connection_timeout (connect_timeout*) : Timeout for the TCP and Unix socket connections. The connection terminates after this timeout expired.
- buffered: If true, the cursor objects fetch the results immediately after executing queries. The default value is False.
- raw: If true, MySQL results are returned as-is rather than converting into Python types. The default value is False. You can set it to true if you want a query result in MySQL type.

- force_ipv6: When setting to True, uses IPv6 when an address resolves to both IPv4 and IPv6. By default, IPv4 is used in such cases. The default value for this argument is false.
- pool_name: It is the Connection pool name that you are creating or using.
- pool_size: Connection pool size that you want to create. The default value is 5.
- pool_reset_session: Reset session variables when the connection is returned to the pool. The default is True.
- use_pure: Specify whether to use pure Python or C Extension. If use_pure=False, then a pure Python module is used; otherwise, it connects MySQL using C extension. Moreover, if C Extension is not available, MySQL Connector Python automatically falls back to the pure Python implementation.
- unix_socket: The location of the Unix socket file. These enable communication between two processes.
- auth_plugin: Authentication plugin to use, Added in 1.2.1.
- collation: MySQL collation to use. You can use the collation that you set while installing MySQL Server. The default value is utf8_generalW_chiich.
- sql_mode: Set the sql_mode session variable at connection time.

Use the Dictionary to keep MySQL Connection arguments

Furthermore, let see how to use a dictionary to store all of these connection arguments.

If you have lots of connection arguments, it's best to keep them in a dictionary and use the ** operator. for example, you know you require a minimum of four arguments (i.e., username, password, hostname, database name) to connect MySQL.

If you have lots of connection arguments, it's best to keep them in a dictionary and use the ** operator. In exceptional cases, we need more than four arguments in the connect method to connect the MySQL database. Let's understand this. For example, below are **three more connection arguments** we can use in the connect() method.

1. connection_timeout – **Timeout** for the TCP and Unix socket connections
2. auto_commit – Whether to auto-commit transactions. The default is false
3. pool_size – Connection pool size if you want to use connection pooling.

Hints:

```
import mysql.connector
from mysql.connector import Error

try:
    connection_config_dict = {
        'user': 'pynative',
        'password': 'pynative@123',
        'host': '127.0.0.1',
        'database': 'Electronics',
        'raise_on_warnings': True,
        'use_pure': False,
        'autocommit': True,
        'pool_size': 5
    }
    connection = mysql.connector.connect(**connection_config_dict)
```

```
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL Server version ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database();")
        record = cursor.fetchone()
        print("Your connected to database: ", record)

except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if connection.is_connected():
        cursor.close()
        connection.close()
        print("MySQL connection is closed")
```

10.4 Connect to MySQL Using Connector Python C Extension

Python connector module has a C Extension interface to connect the MySQL database. The use_pure connection argument determines whether to connect to MySQL using a pure Python interface or a C Extension.

The default value of use_pure is False means it uses the pure Python implementation to connect that we already discussed. The below example demonstrates how to connect using a C extension.

Hints:

```
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Electronics',
                                         user='pynative',
                                         password='pynative@#29', use_pure=True)

    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Connected to MySQL database... MySQL Server version on ", db_Info)
except Error as e:
    print("Error while connecting to MySQL", e)
finally:
    if connection.is_connected():
        connection.close()
        print("MySQL connection is closed")
```

10.5 Python SQLite Database Connection

This section lets you know how to connect to SQLite database in Python using the sqlite3 module.
Use the following steps to connect to SQLite
How to Connect to SQLite Database in Python

1. **Import sqlite3 module**
   import sqlite3 statement imports the sqlite3 module in the program. Using the classes and methods defined in the sqlite3 module we can communicate with the SQLite database.

2. **Use the connect() method**

Use the connect() method of the connector class with the database name. To establish a connection to SQLite, you need to pass the database name you want to connect. If you specify the database file name that already presents on the disk, it will connect to it. But if your specified SQLite database file doesn't exist, SQLite creates a new database for you.
This method returns the SQLite Connection Object if the connection is successful.

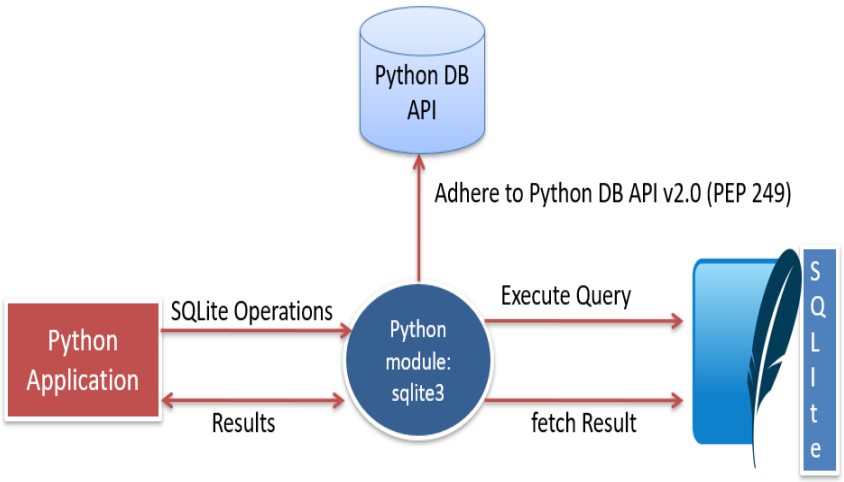
3. **Use the cursor() method**
   Use the cursor() method of a connection class to create a cursor object to execute SQLite command/queries from Python.

4. **Use the execute() method**
   The execute() methods run the SQL query and return the result.

5. **Extract result using fetchall()**
   Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

6. **Close cursor and connection objects**
   use cursor.clsoe() and connection.clsoe() method to close the cursor and SQLite connections after your work completes

7. **Catch database exception if any that may occur during this connection process.**



Python sqlite3 module working

The following Python program creates and connects to the new database file "**SQLite_Python.db"** and prints the SQLite version details.
Repeat step to with a new vertex until all edges are colored.

Hints:

```
import sqlite3

try:
    sqliteConnection = sqlite3.connect('SQLite_Python.db')
    cursor = sqliteConnection.cursor()
    print("Database created and Successfully Connected to SQLite")

    sqlite_select_Query = "select sqlite_version();"
    cursor.execute(sqlite_select_Query)
```

```
    record = cursor.fetchall()
    print("SQLite Database Version is: ", record)
    cursor.close()

except sqlite3.Error as error:
    print("Error while connecting to sqlite", error)
finally:
    if sqliteConnection:
        sqliteConnection.close()
        print("The SQLite connection is closed")
```

Output.
Database created and successfully connected.

10.6 Python Database Programming Exercise

Hospital Information System
MySQL
1. Create Database
2. Create Hospital Table
3. Create Doctor Table

These tables should look like this.
Hospital table

Hospital_Id	Hospital_Name	Bed Count
1	Mayo Clinic	200
2	Cleveland Clinic	400
3	Johns Hopkins	1000
4	UCLA Medical Center	1500

Doctor table

Doctor_Id	Doctor_Name	Hospital_Id	Joining_Date	Speciality	Salary	Experience
101	David	1	2005-02-10	Pediatric	40000	*NULL*
102	Michael	1	2018-07-23	Oncologist	20000	*NULL*
103	Susan	2	2016-05-19	Garnacologist	25000	*NULL*
104	Robert	2	2017-12-28	Pediatric	28000	*NULL*
105	Linda	3	2004-06-04	Garnacologist	42000	*NULL*
106	William	3	2012-09-11	Dermatologist	30000	*NULL*
107	Richard	4	2014-08-21	Garnacologist	32000	*NULL*
108	Karen	4	2011-10-17	Radiologist	30000	*NULL*

Try
Write the database code for the following queries.
Exercise 1: Connect to your database server and print its version
Exercise 2: Fetch Hospital and Doctor Information using hospital Id and doctor Id
Exercise 3: Get the list Of doctors as per the given specialty and salary
Exercise 4: Get a list of doctors from a given hospital
Exercise 5: Update doctor experience in years

Hints:

```
CREATE database python_db;
CREATE TABLE Hospital (
       Hospital_Id serial NOT NULL PRIMARY KEY,
```

```
	Hospital_Name VARCHAR (100) NOT NULL,
	Bed_Count serial
);

INSERT INTO Hospital (Hospital_Id, Hospital_Name, Bed_Count)
VALUES
('1', 'Mayo Clinic', 200),
('2', 'Cleveland Clinic', 400),
('3', 'Johns Hopkins', 1000),
('4', 'UCLA Medical Center', 1500);

CREATE TABLE Doctor (
	Doctor_Id serial NOT NULL PRIMARY KEY,
	Doctor_Name VARCHAR (100) NOT NULL,
	Hospital_Id serial NOT NULL,
	Joining_Date DATE NOT NULL,
	Speciality VARCHAR (100) NOT NULL,
	Salary INTEGER NOT NULL,
	Experience SMALLINT
);
INSERT INTO Doctor (Doctor_Id, Doctor_Name, Hospital_Id, Joining_Date, Speciality,
Salary, Experience)
VALUES
('101', 'David', '1', '2005-2-10', 'Pediatric', '40000', NULL),
('102', 'Michael', '1', '2018-07-23', 'Oncologist', '20000', NULL),
('103', 'Susan', '2', '2016-05-19', 'Garnacologist', '25000', NULL),
('104', 'Robert', '2', '2017-12-28', 'Pediatric ', '28000', NULL),
('105', 'Linda', '3', '2004-06-04', 'Garnacologist', '42000', NULL),
('106', 'William', '3', '2012-09-11', 'Dermatologist', '30000', NULL),
('107', 'Richard', '4', '2014-08-21', 'Garnacologist', '32000', NULL),
('108', 'Karen', '4', '2011-10-17', 'Radiologist', '30000', NULL);
```

PROBING FOR FUTHER QUESTIONS

Practice the following Python MySQL queries by using above database

Python MySQL – Select Query
Python MySQL – Where Clause
Python MySQL – Order By Clause
Python MySQL – Delete Query
Python MySQL – Drop Table
Python MySQL – Update Query
Python MySQL – Limit Clause
Python MySQL – Join
Commit & RollBack Operation in Python

Practice the following Python Python SQLite queries by using above database

Python SQLite Queries
Python SQLite – Cursor Object
Python SQLite – Create Table
Python SQLite – Insert Data
Python SQLite – Select Data from Table
Python SQLite – WHERE Clause
Python SQLite – ORDER BY Clause
Python SQLite – LIMIT Clause
Python SQLite – JOIN Clause

Python SQLite – Deleting Data in Table
Python SQLite – DROP Table
Python SQLite – Update Data

11. Graphical User Interface

11.1 Simple GUI calculator using Tkinter

Python offers multiple options for developing a GUI (Graphical User Interface). Out of all the GUI methods, Tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with Tkinter outputs the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.

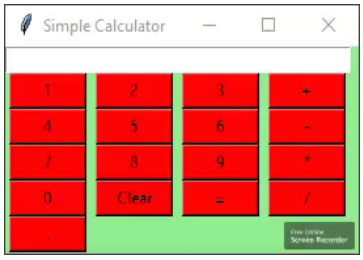
To create a Tkinter:

1. Importing the module – tkinter
2. Create the main window (container)
3. Add any number of widgets to the main window
4. Apply the event Trigger on the widgets.

Hints:

```
# Python program to create a simple GUI
# calculator using Tkinter

# import everything from tkinter module
from tkinter import *

# globally declare the expression variable
expression = ""


# Function to update expression
# in the text entry box
def press(num):
    # point out the global expression variable
    global expression

#write the code here
```

Output:



Try

1. GUI program to add a button in your application using tkinter module.
2. GUI program to add a canvas in your application using tkinter module.

3. GUI program to create two buttons exit and hello using tkinter module.

11.2 Temperature Converter using Tkinter

Python Tkinter is a GUI programming package or built-in library. Tkinter provides the Tk GUI toolkit with a potent object-oriented interface. Python with Tkinter is the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.
Approach:

- Importing the module – tkinter, functools from partial
- Create the main window
- Add number of widgets to the main window: Button, Entry, Label
- Displaying message
- Apply the event trigger on the widgets.

Hints:

```
import tkinter as tk
from tkinter import messagebox
from functools import partial

# Declaration of global variable
temp_Val = "Celsius"

# getting drop down value
def store_temp(set_temp):
    global temp_Val
    temp_Val = set_temp

# Lay out widgets
root.grid_columnconfigure(1, weight = 1)
root.grid_rowconfigure(1, weight = 1)

inputNumber = tk.StringVar()
var = tk.StringVar()
……
# Driver code to test above function
# Initial values assumed
a =-200
b = 300
regulaFalsi(a, b)
Muller(a, b, c);
```

Try

1. GUI program to create a Combobox with three options using tkinter module.
2. GUI program to create a Checkbutton widget using tkinter module.
3. GUI program to create a Spinbox widget using tkinter module.

11.3 Python: Weight Conversion GUI using Tkinter

Python offers multiple options for developing a GUI (Graphical User Interface). Out of all the GUI methods, Tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with Tkinter outputs the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.

Steps to Create a Tkinter:

- Importing the module – tkinter
- Create the main window (container)
- Add any number of widgets to the main window
- Apply the event Trigger on the widgets.

Below is what the GUI looks like:

Hints:

```
# Python program to create a simple GUI
# weight converter using Tkinter

from tkinter import *
# Create a GUI window
window = Tk()

# Function to convert weight
# given in kg to grams, pounds
# and ounces
def from_kg():

.....
", command = from_kg)

# grid method is used for placing
# the widgets at respective positions
# in table like structure
e1.grid(row = 0, column = 0)
e2.grid(row = 0, column = 1)
e3.grid(row = 1, column = 0)
e4.grid(row = 1, column = 1)
e5.grid(row = 1, column = 2)
t1.grid(row = 2, column = 0)
t2.grid(row = 2, column = 1)
t3.grid(row = 2, column = 2)
b1.grid(row = 0, column = 2)
```

Try

1. GUI program to create a Text widget using tkinter module. Insert a string at the beginning then insert a string into the current text. Delete the first and last character of the text.
2. GUI program to create three single line text-box to accept a value from the user using tkinter module.
3. GUI program to create three radio buttons widgets using tkinter module.

11.4 Create a GUI Marksheet using Tkinter

Create a python GUI mark sheet. Where credits of each subject are given, enter the grades obtained in each subject and click on Submit. The credits per subject, the total credits as well as the SGPA are displayed after being calculated automatically. Use Tkinter to create the GUI interface. *Refer the below articles to get the idea about basics of tkinter and Python.*

- *Tkinter introduction*
- *Basics of Python*

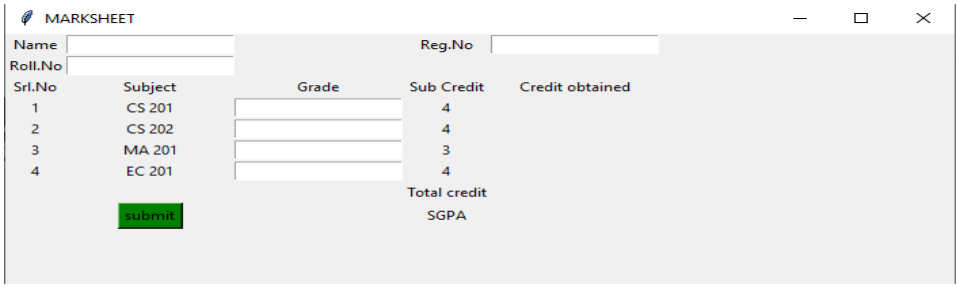
Python offers multiple options for developing a GUI (Graphical User Interface). Out of all the GUI methods, Tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with Tkinter outputs the fastest and easiest way to create GUI applications. Creating a GUI using Tkinter is an easy task.

To create a Tkinter:

- Importing the module – Tkinter
- Create the main window (container)

- Add any number of widgets to the main window
- Apply the event Trigger on the widgets.

This is how the GUI would look:



Hints:

```
# Python program to create a
# GUI mark sheet using tkinter

# function to display the total subject
# credits total credits and SGPA according
# to grades entered
def display():
 …….
# end of display function
# label to enter name
tk.Label(master, text="Name").grid(row=0, column=0)
…..
tk.Label(master, text="Total credit").grid(row=7, column=3)
tk.Label(master, text="SGPA").grid(row=8, column=3
```

Try

1. GUI program to create a ScrolledText widgets using tkinter module.
2. GUI program to create a Progress bar widgets using tkinter module.

12. GRAPHICS-1

12.1 Algorithm Plotting using Turtle

To make use of the turtle methods and functionalities, we need to import turtle."turtle" comes packed with the standard Python package and need not be installed externally. The roadmap for executing a turtle program follows 4 steps:

1. Import the turtle module
2. Create a turtle to control.
3. Draw around using the turtle methods.
4. Run turtle.done().

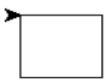
from turtle import *
# or
import turtle

1. Write a function to draw rectangle, triangle, star, polygon, circle and sphere.

Hints:

```
# Python program to draw square
# using Turtle Programming
import turtle
skk = turtle.Turtle()

for i in range(4):
    skk.forward(50)
    skk.right(90)
     -------
turtle.done()
```

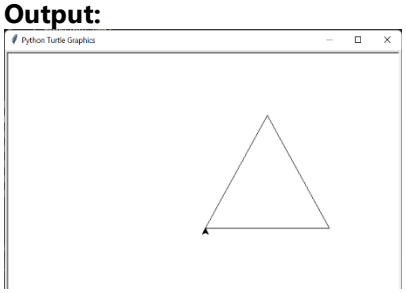
Output:



Hints:

```
# Python program to draw triangle
import turtle

t = turtle.Turtle()

t.forward(200)

t.left(120)
t.forward(200)

t.left(120)
t.forward(200)
```

Output:



Try

1. Draw star plotting using Turtle.
2. Draw polygon plotting using Turtle.
3. Draw circle and sphere plotting using Turtle.
4. Draw sphere plotting using Turtle.

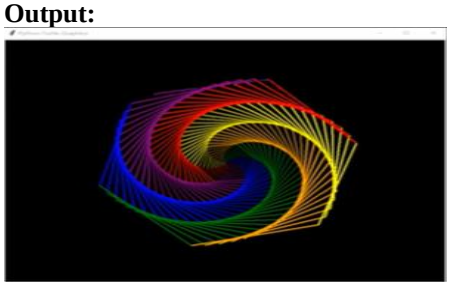
12.2 Drawing of Spiral Helix Pattern and Rainbow Benzene

1. Write a python program to draw spiral helix pattern

Hints:

```
# Python program to draw
# Spiral Helix Pattern
# using Turtle Programming

import turtle
loadWindow = turtle.Screen()
turtle.speed(2)

for i in range(100):
    turtle.circle(5*i)
    turtle.circle(-5*i)
    turtle.left(i)

turtle.exitonclick()
```

Output:



Try

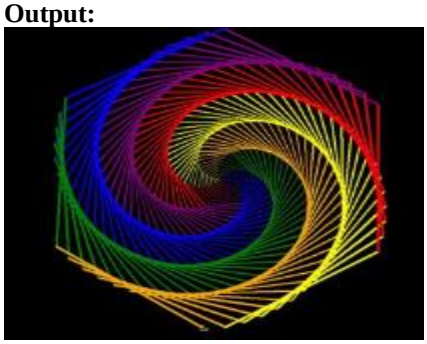
1. Draw spiral helix pattern using red and green colour only.
2. Draw spiral helix pattern using yellow and green colour only.

2. Write a python program to draw rainbow benzene.

Hints:

```
# Python program to draw
# Rainbow Benzene
# using Turtle Programming
import turtle
colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow']
t = turtle.Pen()
turtle.bgcolor('black')
for x in range(360):
    t.pencolor(colors[x%6])
    t.width(x//100 + 1)
    t.forward(x)
    t.left(59)
```

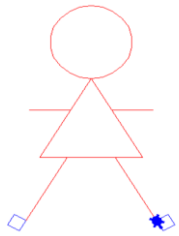
Output:



Try

1. Take input colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow'] and verify the results.
2. Take input colors = ['pink', 'red', 'blue', 'green', 'purple', 'yellow'] and verify the results.

12.3 Drawing a person using circle, triangle and lines



Hints:

```
import turtle

def draw_dream():
    window = turtle.Screen()
    window.bgcolor("white")
    draw_Scarlett()
    window.exitonclick()

 def draw_Scarlett():
    brad = turtle.Turtle()
    brad.shape("turtle")
    brad.color("red")
    draw_head(brad)
    draw_body(brad)
```

```
    draw_arm(brad)
    draw_leg1(brad)
    draw_leg2(brad)

def draw_head(brad):
    brad.circle(60)
    brad.speed(3)
    brad.right(60)

def draw_body(brad):
    num = 0
    while num < 3:
        brad.forward(150)
        brad.right(120)
        brad.speed(1)
        num += 1
#Write the code here
```

Try

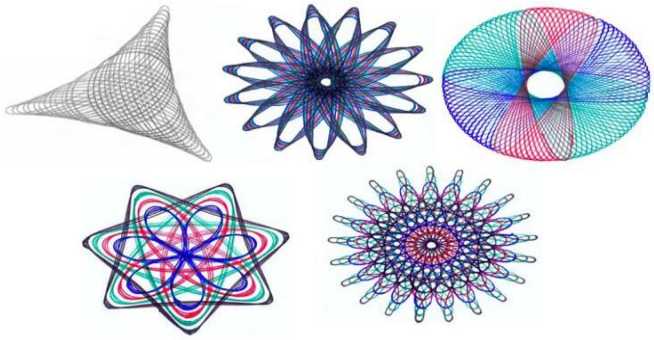
1. By using turtle to draw a person using circle, triangle and lines in reverse order.

12.4 Python Turtle Spirograph

A **Spirograph** is a geometric drawing toy that produces mathematical roulette curves of the variety technically known as **hypotrochoids** and **epitrochoids**. It was developed by British engineer Denys Fisher and first sold in 1965.

Write a python Program to print the below spirographs.

Examples of patterns created using a spirograph:



Hints:

```
#Python Turtle - Spirograph
import turtle
from math import cos,sin
from time import sleep

window = turtle.Screen()
window.bgcolor("#FFFFFF")

mySpirograph = turtle.Turtle()
mySpirograph.hideturtle()
mySpirograph.tracer(0)
mySpirograph.speed(0)
mySpirograph.pensize(2)

myPen = turtle.Turtle()
myPen.hideturtle()
```

```
myPen.tracer(0)
myPen.speed(0)
myPen.pensize(3)
myPen.color("#AA00AA")

R = 125
r = 75
d = 125

angle = 0

myPen.penup()
myPen.goto(R-r+d,0)
myPen.pendown()

theta = 0.2
steps = int(6*3.14/theta)

for t in range(0,steps):
…..
    mySpirograph.getscreen().update()
    sleep(0.05)
```

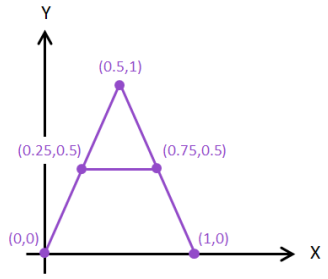
Try

1. By using python turtle implement different spirograph as shown in above figure.

12.5 Python Turtle-WordArt Challenge

In this challenge we will use Python Turtle to draw text on screen and customise the appearance of our text.

To do so we have created our own font as a Python dictionary. Each letter of the alphabet is represented as a set of (x, y) coordinates as follows:



"A" : ((0,0),(0.5,1),(0.75,0.5),(0.25,0.5),(0.75,0.5),(1,0))

We will then use these coordinates to trace a line using Python Turtle using a "dot-to-dot" approach.

Hints:

```
#Python Turtle - WordArt Challenge
import turtle
import random
from alphabet import alphabet
from math import cos, sin, atan2, radians, degrees

myPen = turtle.Turtle()
myPen.hideturtle()
myPen.speed(0)
window = turtle.Screen()
window.bgcolor("#000000")
```

```
myPen.pensize(2)

def displayMessage(message,fontSize,color,x,y,rotationAngle):
  …….

#Main Program Starts Here
fontSize = 30
fontColor="#FF00FF"
characterSpacing = 5
cursorX = -150
cursorY = -100
```

Try

1. By using python turtle to implement WordArt like B, C, E.
2. By using python turtle to implement WordArt like X, Y, Z.

13. GRAPHICS-2

13.1 DDA Line generation Algorithm in Computer Graphics

Introduction:

DDA (Digital Differential Analyzer) is a line drawing algorithm used in computer graphics to generate a line segment between two specified endpoints. It is a simple and efficient algorithm that works by using the incremental difference between the x-coordinates and y-coordinates of the two endpoints to plot the line.

The steps involved in DDA line generation algorithm are:

1. Input the two endpoints of the line segment, (x1,y1) and (x2,y2).
2. Calculate the difference between the x-coordinates and y-coordinates of the endpoints as dx and dy respectively.
3. Calculate the slope of the line as m = dy/dx.
4. Set the initial point of the line as (x1,y1).
5. Loop through the x-coordinates of the line, incrementing by one each time, and calculate the corresponding y-coordinate using the equation y = y1 + m(x – x1).
6. Plot the pixel at the calculated (x,y) coordinate.
7. Repeat steps 5 and 6 until the endpoint (x2,y2) is reached.

DDA algorithm is relatively easy to implement and is computationally efficient, making it suitable for real-time applications. However, it has some limitations, such as the inability to handle vertical lines and the need for floating-point arithmetic, which can be slow on some systems. Nonetheless, it remains a popular choice for generating lines in computer graphics.

In any 2-Dimensional plane, if we connect two points (x0, y0) and (x1, y1), we get a line segment. But in the case of computer graphics, we cannot directly join any two coordinate points, for that, we should calculate intermediate points' coordinates and put a pixel for each intermediate point, of the desired color with the help of functions like putpixel(x, y, K) in C, where (x,y) is our co-ordinate and K denotes some color.

Examples:

Input: *For line segment between (2, 2) and (6, 6) :*
Output: *we need (3, 3) (4, 4) and (5, 5) as our intermediate points.*
Input: *For line segment between (0, 2) and (0, 6) :*
Output: *we need (0, 3) (0, 4) and (0, 5) as our intermediate points.*

Hints:

```
/ calculate dx , dy

dx = X1 – X0;
dy = Y1 – Y0;

// Depending upon absolute value of dx & dy
// choose number of steps to put pixel as

#Write the code here

for (int i = 0; i <= steps; i++)
{
    putpixel (round(X),round(Y),WHITE);
    X += Xinc;
    Y += Yinc;
}
```

Try

1. Take input for line segment between (3, 3) and (7, 7) and verify the results.
2. Take input for line segment between (8, 8) and (5, 5) and verify the results.

13.2 Bresenham's Line Generation Algorithm

Given the coordinate of two points A (x1, y1) and B (x2, y2). The task is to find all the intermediate points required for drawing line AB on the computer screen of pixels. Note that every pixel has integer coordinates.

Examples:

Input: A(0,0), B(4,4)
Output: (0,0), (1,1), (2,2), (3,3), (4,4)
Input: A(0,0), B(4,2)
Output: (0,0), (1,0), (2,1), (3,1), (4,2)

Below are some assumptions to keep the algorithm simple.
We draw lines from left to right.
x1 < x2 and y1< y2
Slope of the line is between 0 and 1. We draw a line from lower left to upper right.

Hints:

```
# A naive way of drawing line
def naiveDrawLine(x1, x2, y1, y2):
    m = (y2 - y1) / (x2 - x1)

    # for (x = x1; x <= x2; x++) {
    for x in range(x1, x2 + 1):
    #Write the code here

    print(x, y)
```

Try

1. Take input: A (1,0), B(3,4) and verify the results.
2. Take input: A (1,1), B(4,4) and verify the results.

13.3 Bresenham's circle drawing algorithm

Given the coordinate of two points A(x1, y1) and B(x2, y2). The task is to find all the intermediate points required for drawing line AB on the computer screen of pixels. Note that every pixel has integer coordinates.

Hints:

```
// Python program for circle drawing
// using Bresenham's Algorithm
// in computer-graphics

def bresenham_circle(x0, y0, radius):
  x = radius
  y = 0
  err = 0

  while x >= y:
    print(x0 + x, y0 + y)
    print(x0 + y, y0 + x)
    print(x0 - y, y0 + x)
    print(x0 - x, y0 + y)
    print(x0 - x, y0 - y)
    print(x0 - y, y0 - x)
    print(x0 + y, y0 - x)
    print(x0 + x, y0 - y)

    y += 1
    err += 1 + 2*y
    if 2*(err-x) + 1 > 0:
      x -= 1
      err += 1 - 2*x
```

Output:

Try

1. Take input as xc = 150, yc = 150, r = 30 and verify the results.
2. Take input as xc = 165, yc = 153, r = 15 and verify the results.

13.4 Boundary Fill Algorithm

Introduction: Boundary Fill Algorithm starts at a pixel inside the polygon to be filled and paints the interior proceeding outwards towards the boundary. This algorithm works **only if** the color with which the region has to be filled and the color of the boundary of the region are different. If the boundary is of one single color, this approach proceeds outwards pixel by pixel until it hits the boundary of the

region.

Boundary Fill Algorithm is recursive in nature. It takes an interior point(x, y), a fill color, and a boundary color as the input. The algorithm starts by checking the color of (x, y). If it's color is not equal to the fill color and the boundary color, then it is painted with the fill color and the function is called for all the neighbours of (x, y). If a point is found to be of fill color or of boundary color, the function does not call its neighbours and returns. This process continues until all points up to the boundary color for the region have been tested.

The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels. **4-connected pixels :** After painting a pixel, the function is called for four neighbouring points. These are the pixel positions that are right, left, above, and below the current pixel. Areas filled by this method are called 4-connected.

Hints:

```
void boundaryFill4(int x, int y, int fill_color,int boundary_color)
{
    if(getpixel(x, y) != boundary_color &&
       getpixel(x, y) != fill_color)
    {
        #Write the code here
}
```

Try

1. Program to fill the boundary with different colours.

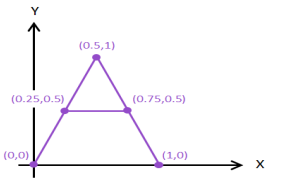
13.5 Morphing Algorithm

Tweening / Morphing effects are often used in Computer Animations to change the shape of an object by morphing an object from one shape into another.

In Tweening, key frames are provided and "in-between" frames are calculated to make a smooth looking animation.

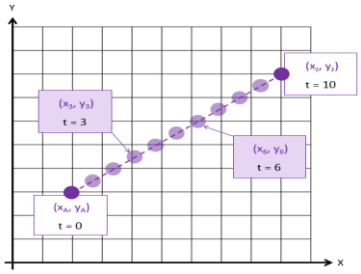
In this blog post we will implement a Tweening algorithm to morph a letter of the alphabet (e.g. "A" into another letter "Z") using a linear interpolation.

We will consider a letter as being a list of connected nodes (dots) where each dot has its own set of (x,y) coordinates.



"A" : ((0,0),(0.5,1),(0.75,0.5),(0.25,0.5),(0.75,0.5),(1,0))

Linear Interpolation Formulas:



Using the following linear interpolation formulas we can calculate the (x,y) coordinates of each dot for any "in-between" frame.

$x(t) = x_A + (x_Z - x_A) * t / 10$

$y(t) = y_A + (y_Z - y_A) * t / 10$

- "t" represents the time: in other words the frame number (e.g. between 0 and 10)
- (x_A,y_A) the coordinates of a node/dot from the starting letter (e.g. A)
- (x_Z,y_Z) the coordinates of a node/dot from the ending letter (e.g. Z)

Using these formulas we can see that:

$x(0) = x_A$

$y(0) = y_A$

$x(10) = x_Z$

$y(10) = y_Z$

Hints:

```
#Python Turtle
import turtle
import random
from alphabet import alphabet
from time import sleep

myPen = turtle.Turtle()
myPen.hideturtle()
myPen.tracer(0)
myPen.speed(0)
window = turtle.Screen()
window.bgcolor("#000000")
myPen.pensize(4)

def morphing(letter1,letter2,t,fontSize,color,x,y):
  …..
#Write the code here
```

Try

1. Write a program to draw different alphabets using the linear interpolation equations and verify the result.

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. GseeksforGeeks – 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. Martin C. Brown , "Python: The Complete Reference" McGraw Hill Education, Fourth edition , 2018.

VI. REFERENCE BOOKS:

1. John M Zelle, "Python Programming: An Introduction to Computer Science", Ingram short title, 3rd Edition, 2016.
2. R. Nageswara Rao , "Core Python Programming" Dreamtech Press India Pvt Ltd 2018
3. Yashavant Kanetkar, Aditya Kanetkar, "Let Us Python", BPB Publications, 2nd Edition, 2019.
4. Paul Barry, "Head First Python: A Brain-Friendly Guide", O'Reilly, 2nd Edition, 2016
5. Taneja Sheetal, Kumar Naveen, "Python Programming – A Modular Approach", Pearson, 1st Edition, 2017.

VII. ELECTRONICS RESOURCES

1. https://realPython.com/Python3-object-oriented-programming/
2. https://Python.swaroopch.com/oop.html
3. https://Python-textbok.readthedocs.io/en/1.0/Object_Oriented_Programming.html
4. https://www.programiz.com/Python-programming/
5. https://www.geeksforgeeks.org/python-programming-language/

VIII. MATERIALS ONLINE (Include full stack)

1. Course template
2. Lab Manual