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

# ARTIFICIAL INTELLIGENCE LABORATORY

| V Semester: CSE \| IT \| CSE (CS) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Course Code | Category | Hours / Week | | | Credits | Maximum Marks | | |
| ACSD26 | Core | L | T | P | C | CIA | SEE | Total |
| | | 0 | 0 | 2 | 1 | 40 | 60 | 100 |
| Contact Classes: Nil | Tutorial Classes: Nil | Practical Classes: 45 | | | | Total Classes: 45 | | |
| Prerequisite: Python Programming | | | | | | | | |

## I. COURSE OVERVIEW:

Artificial Intelligence refers to the practical implementation and utilization of artificial intelligence (AI) technologies to solve real-world problems and address specific challenges in various domains. The objective of this laboratory course is to provide students with hands-on experience in applying AI techniques. Applications provide a foundation for the implementation of artificial intelligence across various industries, addressing specific challenges and delivering practical solutions the scope and applications of AI will vary based on the specific needs and challenges within different industries and sectors.

## II. COURSE OBJECTIVES:

**The students will try to learn:**

I. The foundational concepts of Artificial Intelligence (AI) through hands-on programming exercises covering logic programming, intelligent agents, search algorithms, and game theory.
II. The practical application in implementing classic AI techniques like A*, Best-First Search, AO* Algorithm, and Minimax in real-world problem simulations.
III. Problem-solving and reasoning skills by building expert systems, solving puzzles, and modeling decision-making processes using logical and heuristic-based approaches.
IV. The suitability of advanced applications such as AI in games and natural language processing (NLP) by building bots, performing stemming, chunking, and using bag-of-words models.

## III. COURSE OUTCOMES:

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

| | |
|---|---|
| CO1 | Apply Python programming and AI libraries to preprocess and explore datasets for developing AI models. |
| CO2 | Design and simulate intelligent agents using rule-based and model-based approaches for various environments such as the vacuum world. |
| CO3 | Implement search algorithms including uninformed, heuristic, and adversarial strategies to solve classical AI problems. |
| CO4 | Develop and evaluate expert systems using logical inference and rule-based reasoning to perform tasks like medical diagnosis or fault detection. |
| CO5 | Apply game-playing algorithms such as Minimax and Alpha-Beta pruning to build competitive bots for games like Tic-Tac-Toe and Connect Four. |
| CO6 | Solve natural language processing tasks using stemming, chunking, and Bag-of-Words models to analyze and process textual data. |

# ARTIFICIAL INTELLIGENCE LABORATORY COURSE CONTENT

IV. COURSE CONTENT:

# EXERCISES FOR ARTIFICIAL INTELLIGENCE LABORATORY

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

## 1 Getting Started Exercises

### 1.1 Installing Python and other supportive libraries

1. Install Python 3 on your machine by typing the following command on the terminal:

   ```
   $ python3 --version
   ```

2. Install various useful packages by following the relevant links:

   NumPy: http://docs.scipy.org/doc/numpy-1.10.1/user/install.html
   SciPy: http://www.scipy.org/install.html
   scikit-learn: http://scikit-learn.org/stable/install.html
   matplotlib: http://matplotlib.org/1.4.2/users/installing.html

   Note: If you are on Windows, you should have installed a SciPy-stack compatible version of Python 3

3. Install a couple of packages before starting logic programming in python like logpy and sympy using pip. These packages are useful to work with matching mathematical expressions.

   ```
   $ pip3 install logpy
   $ pip3 install sympy
   ```

   If you get an error during the installation process for logpy, you can install from source at https://github.com/logpy/logpy.

**Try:** Practice appropriate python commands to build a model and use the above libraries (relevant packages) to interact with the data. The commands may include importing of packages containing all the datasets, loading of datasets, printing data, printing labels, loading images, extracting a specific image etc.

### 1.2 Intelligent Agent in a Vacuum World

A **Simple Reflex Agent** that operates in a vacuum-cleaning environment with two rooms (A and B). The agent reacts based solely on the room's current status (Clean or Dirty), without memory or learning.

**Input:**

1. Load room status from a JSON file.
2. Simulate the agent's decision-making using a **simple rule**:

   If the current room is **Dirty** → Clean it.

   If the current room is **Clean** → Move to the other room.

3. Run the agent for a **fixed number of steps (e.g., 4)**.

4. Print each percept and action taken.

**Output**: Agent behavior log like:

Room A is Dirty

Action: Clean

Moved to Room B

Room B is Clean

**Hints**

```
import json

class SimpleReflexAgent:
    def __init__(self, env):
        self.location = 'A'
        self.environment = env

    def perceive(self):
        return self.location, self.environment[self.location]

    def act(self, percept):
        loc, status = percept
        if status == "Dirty":
            self.environment[loc] = "Clean"
            return "Clean"
        else:
            self.location = 'B' if loc == 'A' else 'A'
            return f"Move to {self.location}"

def main():
    with open("vacuum_environment.json") as f:
        environment = json.load(f)

    agent = SimpleReflexAgent(environment)

    for _ in range(4):
        percept = agent.perceive()
        action = agent.act(percept)
        print(f"Room {agent.location} - {environment[agent.location]}")
        print(f"Action: {action}\n")

main()
```

**Try:** Change `"A"` and `"B"` both to `"Clean"` and observe the agent's behavior.Swap the starting location to Room `"B"`.Add a step limit counter to prevent infinite movement.

## 1.3 Agents and Environments, Performance Measures, Structure of Intelligent Agents

Implement a modular simulation of the **Vacuum Cleaner World**. The environment consists of two or more locations (e.g., Room A and B), each of which may be **Clean** or **Dirty**. A vacuum agent perceives its location and whether that location contains dirt, and chooses an action:
**Suck**: cleans current location
**Left/Right**: moves between rooms
You must:
Build an **environment simulator** with configurable:
World size (2+ locations)
Dirt placement
Track the **agent's actions** and measure **performance** (e.g., dirt cleaned, energy used).

**Input:**

1. Initial room configuration (e.g., A and B: Clean or Dirty)
2. Maximum steps for simulation
3. Agent type (reflex, random, model-based)

Output:

1. Step-by-step log of actions and percepts
2. Final environment state
3. Performance score (e.g., dirt cleaned – moves used)

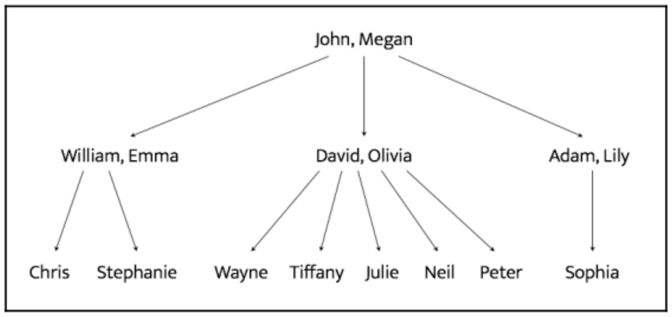
**Hints**

```
/* Import heapq. */
import json
import random

# Agent class

# Environment class


def main():
    with open("vacuum_config.json") as f:
        config = json.load(f)

    env = VacuumEnvironment(config)
    env.simulate()

if __name__ == "__main__":
    main()
```

# 2 Exercises on Search Techniques

## 2.1 Parsing a Family Tree

Use the most familiar Logic programming and solve an interesting problem of Parsing a Family Tree by considering the following diagram. John and Megan have three sons – William, David, and Adam. The wives of William, David, and Adam are Emma, Olivia, and Lily respectively. William and Emma have two children – Chris and Stephanie. David and Olivia have five children – Wayne, Tiffany, Julie, Neil, and Peter. Adam and Lily have one child – Sophia. Based on these facts, create a program that can tell us the name of Wayne's grandfather or Sophia's uncles are.



**Input**: A .json file specified with the relationship among all the people involved.

**Output**: Ask the following questions to understand if our solver can come up with the right answer.

1. Who John's children are?
2. Who is William's mother?
3. Who are Adam's parents?
4. Who are Wayne's grandparents?

**Hints**

```
/*Create a new Python file and import the following packages: */
import json
from logpy import Relation, facts, run, conde, var, eq

/* Define a function to check if x is the parent of y. We will use the logic that if
x is the parent of y, then x is either the father or the mother. We have already
defined “father” and “mother” in our fact base: */

/* Define a function to check if x is the grandparent of y. We will use the logic
that if x is the grandparent of y, then the offspring of x will be the parent of y:
*/

# Check for sibling relationship between 'a' and 'b'
def sibling(x, y):
        temp = var()
        return conde((parent(temp, x), parent(temp, y)))

/* Define the main function and initialize the relations father and mother */
/* Load the data from the relationships.json file */
/* Read the data and add them to our fact base */
Define the variable x:
```

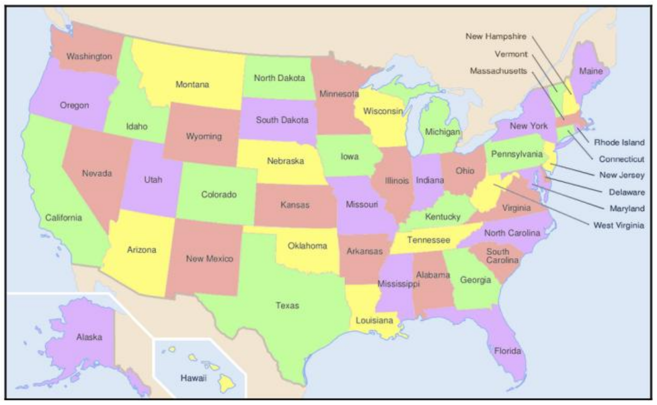
```
x = var()

/* We are now ready to ask some questions and see if our solver can come up with the
right answers. Let's ask who John's children are:

# John's children
name = 'John'
output = run(0, x, father(name, x))
print("\nList of " + name + "'s children:")
for item in output:
        print(item)

/* Start asking the questions like: Who is William's mother?, Who are Adam's parents?,
Who are Wayne's grandparents?, Who are Megan's grandchildren?, Who are David's
siblings?, Who are Tiffany's uncles? */
/* List out all the spouses in the family */
```

**Try**:

1. Who are Megan's grandchildren?
2. Who are David's siblings?
3. Who are Tiffany's uncles?
4. List out all the spouses in the family.

## 2.2 Analyzing Geography

Use logic programming to build a solver to analyze geography. In this problem, specify information about the location of various states in the US and then query our program to answer various questions based on those facts and rules. The following is a map of the US:

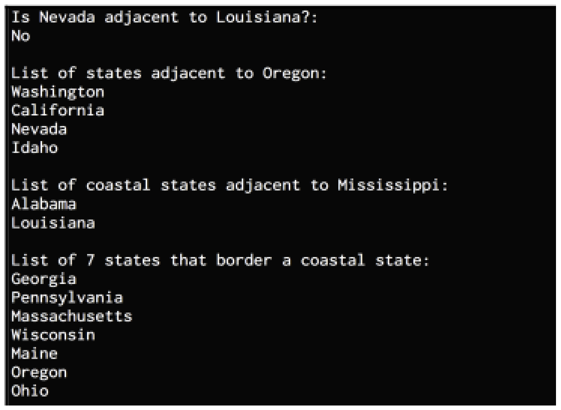


**Input**:

1. Define the input files to load the data from.
2. Read the files containing the coastal states.
3. Add the adjacency information to the fact base.
4. Initialize the variables x and y.

**Output**:

1. Print out all the states that are adjacent to Oregon.
2. List all the coastal states that are adjacent to Mississippi.
3. List all the coastal states that are adjacent to Mississippi.



```
Is Nevada adjacent to Louisiana?:
No

List of states adjacent to Oregon:
Washington
California
Nevada
Idaho

List of coastal states adjacent to Mississippi:
Alabama
Louisiana

List of 7 states that border a coastal state:
Georgia
Pennsylvania
Massachusetts
Wisconsin
Maine
Oregon
Ohio
```

### Hints

```
/*Create a new Python file and import the following: */

from logpy import run, fact, eq, Relation, var

/*Initialize the relations: */

adjacent = Relation() coastal = Relation()

/*Define the input files to load the data from: */

file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'

/*Load the data: # Read the file containing the coastal states */
with open(file_coastal, 'r') as f:
        line = f.read()
        coastal_states = line.split(',')

/*Add the information to the fact base:
# Add the info to the fact base
for state in coastal_states:
        fact(coastal, state)

/* Read the adjacency data */

/* Add the adjacency information to the fact base */

/* Initialize the variables x and y */

/* Check if Nevada is adjacent to Louisiana */

/* Print out all the states that are adjacent to Oregon, List all the coastal states
that are adjacent to Mississippi, List seven states that border a coastal state, List
states that are adjacent to both Arkansas and Kentucky */
```

**Try**: Add more questions like "list states that are adjacent to both Arkansas and Kentucky" to the program to see if it can answer them.

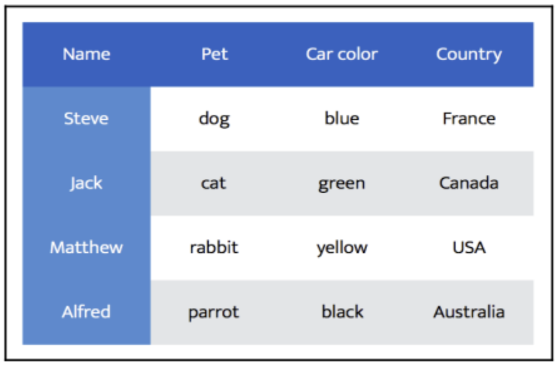
## 2.3 Building a Puzzle Solver

The interesting application of logic programming is in solving puzzles. The goal of this exercise is to specify the conditions of a puzzle, and the program has to come up with a solution. Also specify various bits and pieces of information about four people and ask for the missing piece of information.

**Input**: In the logic program, we specify the puzzle as follows:

- Steve has a blue car.
- The person who owns the cat lives in Canada Matthew lives in USA.
- The person with the black car lives in Australia.
- Jack has a cat.
- Alfred lives in Australia.
- The person who has a dog life in France.
- Who has a rabbit?

**Output:** The goal is to find the person who has a rabbit. Here are the full details about the four people:



| Name | Pet | Car color | Country |
|---|---|---|---|
| Steve | dog | blue | France |
| Jack | cat | green | Canada |
| Matthew | rabbit | yellow | USA |
| Alfred | parrot | black | Australia |

### Hints

```
/*Create a new Python file and import the following packages: */
from logpy import *
from logpy.core
import lall

/* Declare the variable people: */
# Declare the variable
people = var()

/* Define all the rules using lall, The first rule is that there are four people,
The person named Steve has a blue car */

/* The person who has a cat lives in Canada, The person named Matthew lives in USA,
The person who has a black car lives in Australia, The person named Jack has a cat,
The person named Alfred lives in Australia, The person who has a dog lives in France,
The person who has a dog lives in France.*/

/* Run the solver with the preceding constraints */

/* Extract the output from the solution */

/* Print the full matrix obtained from the solver */
```

**Try:** Demonstrate how to solve a puzzle with incomplete information. You can play around with it and see how you can build puzzle solvers for various scenarios.

## 3. Exercises on Heuristic Search Techniques

### 3.1 Best First Search Algorithm

The Best First Search algorithm is a set of rules that work together to perform a search. It considers the various characteristics of a prioritized queue and heuristic search. The goal of this algorithm is to reach the state of final or goal in the shortest possible time.

**Input:** Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list.
3. Repeat the next steps until the GOAL node is reached.
    - If the OPEN list is empty, then EXIT the loop returning 'False'.
    - Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node.
    - If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path.
    - If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list.
    - Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

**Output:**

a. Perform the search process by using additional information to determine the next step towards finding the solution.
b. Perform the search process using an evaluation function to decide which among the various available nodes is the most promising before traversing to that node.
c. Apply priority queues and heuristic search functions to track the traversal.

**Hints**

```
/* Import heapq. */

/* Initialize the graph and heuristic function, dictionary representing the graph and
heuristic values. */

/* Perform the best-first search by defining the priority queue to store nodes with
their heuristic values. */

/* Get the node with the lowest heuristic value. */

/* If the goal is reached, return the path and total cost. */

/* Mark the current node as visited, explore all the neighbors. */
/* Return the solution if exists otherwise return None. */
```

**Try:** Make use of Heuristic Search principle and apply the best first search to solve the 8-puzzle problem.

## 3.2 A* Algorithm

A* Algorithm – A square grid is composed of many obstacles that are scattered randomly. The goal is to find the final cell of the grid in the shortest possible time. Implement A* algorithm to search for the shortest path among the given initial and the final state.

**Input**: Define the graph as nodes with neighboring nodes and cost. Define the heuristic values of each node.

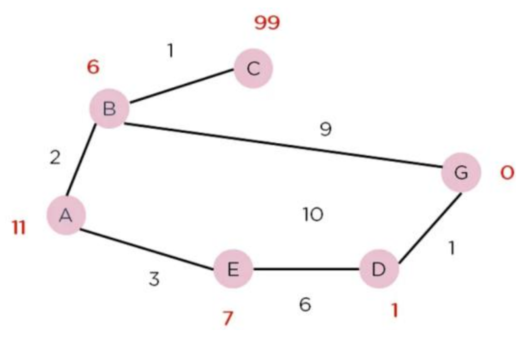
**Output**:

a. Initially represent the problem statement as a graph traversal problem.
b. Perform the search process to obtain the shorter path first, thus making it optimal.
c. Find the least cost outcome for the problem by finding all the possible outcomes.
d. Make use of weighted graph by using numbers to represent the cost of taking each path and find the best route with the least cost in terms of distance and time.

**Hints**

```
/* Implements the A* algorithm to find the shortest path in a weighted graph.*/

/* Represent the dictionary for a graph and heuristic values. */

/* Define the starting and goal node keeping in mind to return the tuple containing
the optimal path and the total cost. */

/* Pop the node with the lowest f_cost, if the goal is reached, return the path and
cost. */

/* Mark the current node as visited */

/* Explore the neighbors to accumulate the path cost by getting the heuristic values,
total path cost. */

/* If the path is found then return, otherwise return None */
```

**Try:** Implement the A* algorithm and calculate the shortest distance between the initial and the goal states by considering the following weighted graph:



## 3.3 AO* Algorithm

Implement the algorithm to generate AND-OR graph or tree to represent the solution by dividing the problem into sub problems and solve them separately to obtain the result by combining all the sub solutions.

**Input:** The AO* algorithm works on the formula given below : f(n) = g(n) + h(n) where,

- g(n): The actual cost of traversal from initial state to the current state.
- h(n): The estimated cost of traversal from the current state to the goal state.
- f(n): The actual cost of traversal from the initial state to the goal state.

**Output:**

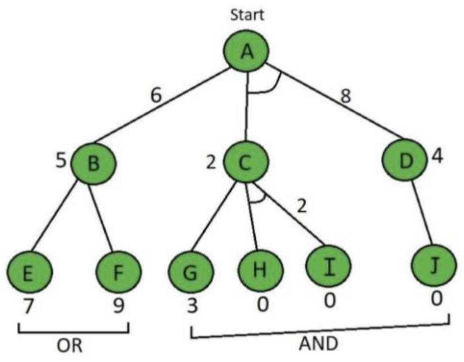
a. Follow problem decomposition approach and solve each sub problem separately and later combine all the solutions.
b. Traverse the graph starting at the initial node and following the current best path and accumulate the set of nodes that are on the path and have not yet been expanded.
c. Pick one of these best unexpanded nodes and expand it. Add its successors to the graph and compute cost of the remaining distance for each of them.
d. Change the cost estimate of the newly expanded node to reflect the new information produced by its successors. Propagate this change backward through the graph. Decide which of the current best path

**Optimal Solution Path: A -> B -> D Total Cost: 5**

**Hints**

```
/* Define a class to represent a node in the AND-OR graph. */

/* Define a class to implement AO* algorithm to find the optimal solution path. */

/* Mention the initial state, goal state, the optimal solution path and its total
cost. */

/* Pick the best node to expand (lowest f_cost) and expand the node. */

/* Propagate changes backward to reflect updated costs, Add successors to the open
list if not already processed. */

/* Generate the optimal path */

/* Expand the given node by calculating f_cost for its successors. */
/* Propagate cost changes backward through the graph.*/

/* Generate the optimal solution path by following the best successors. */

/* Define heuristic values for the nodes, Create nodes for the graph, Define the AND-
OR graph as node connections, Instantiate the AO* algorithm, Perform the AO* search.
*/

/* Generate the result as optimal solution path and total cost.*/
```

**Try:** Implement the algorithm to generate AND-OR graph or tree to represent the solution by dividing the problem into sub problems and solve them separately to obtain the result by combining all the sub solutions.

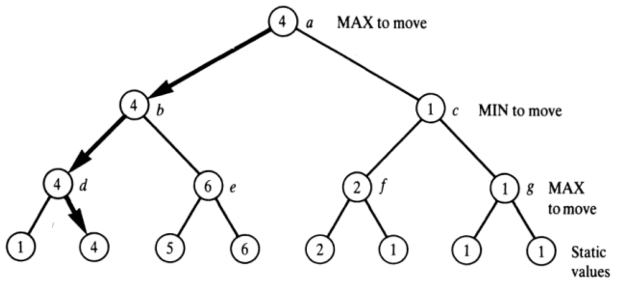


## 4. Exercises on Building Games with AI

### 4.1. Minimax Algorithm

As searching game trees exhaustively is not feasible for interesting games, other methods that rely on searching only part of the game tree have been developed. Among these, a standard technique used in computer game playing (chess) is based on the minimax principle. The goal of this exercise is to implement the minimax principle and identify the changes that a player has to with a game.

**Input:** Game Tree and a Search Tree like:



**Output:** Make use of static and backup values and estimate the best position from a list of candidate positions.

**Hints**

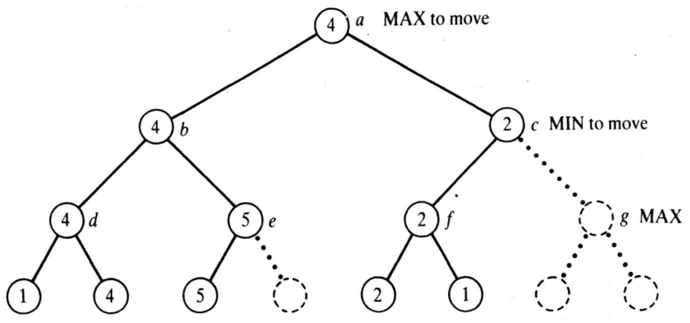
```
/* Calculate the heuristic estimator using the estimation function and estimate the
changes that a player must win. */

/* Implement the procedure as: Minimax procedure: minimax( Pos, BestSucc, Val)  Pos
is a position, Val is its minimax value; best move Vo from Pos leads to position
BestSuc */
```

**Try:** Create a knowledge base including the clauses that help in implementing the straightforward method of minimax principle.

## 4.2. Alpha Beta Pruning

Create a knowledge base representing the straightforward procedure to implement alpha-beta algorithm. Develop a prolog program that systematically visits all the positions in the search tree, up to its terminal positions in a depth-first fashion, and statically evaluates all the terminal positions of this tree.

**Input:** Search Tree, starting point, legal moves, maximum successors of each move, and apply backtracking wherever applicable.



**Output:** Compute the exact value of a root position P by setting the bounds as follows:

$$V(P, -infinity, +infinity) = V(P)$$

**Hints**

```
alphabeta(/* Pass the 4 arguments like position, alpha beta values, good position and
the current value */) :-
        moves(Pos, PostList), !,
        boundedbest(Postlist, Alpha, Beta, GoodPos, Val);
        staticval(Pos, Val).

boundedbest( [Pos | Poslist], Alpha, Beta, GoodPos, GoodVal) :-
        alphabeta( Pos, Alpha, Beta, _, Val),
        goodenough( Poslist, Alpha, Beta, Pos, Val, GoodPos, GoodVal).

goodenough( /* Mention an empty list */, -, -, Pos, Val, Pos, Val) '- !. % No other
candidate

goodenough( -, Alpha, Beta, Pos, Val, Pos, Val) :-
        min-to-rnove( Pos), Val ) Beta, !; % Maximizer attained upper bound
        marto-mov{ Pos), Val ( Alpha, !. % Minimizer attained lower bound
```

```
goodenough( Poslist, Alpha, Beta, Pos, Val, GoodPos, GoodVal) :-
        newbounds( /* Pass the 6 arguments like alpha beta values, good position,
current value, new alpha and beta values */), % Refine bounds boundedbes( Poslist,
NewAlpha, NewBeta, Posl, Vall),
        betterof( Pos, Val, Posl, Vall, GoodPos, GoodVal).

newbounds( Alpha, Beta, Pos, Val, Val, Beta) :-
        min-to-rnove( Pos), Vd > Alpha, !. % Maximizer increased lower bound

newbounds( Alpha, Beta, Pos, Val, Alpha, Vat) :-
        marto-rnove( Pos), Val < Beta, !. % Minimizer decreased upper bound

newbounds( Alpha, Beta, -, -, Alpha, Beta)

betterof( Pos, Val, Posl, Val1, Pos, Val) :-
        min-to-rnove( Pos), Vd > Vall, !;
        marto:nove( Pos), Val < Vall, !.

betterof( -, -, Pos1, Val1, Posl, Vall).
```

**Try:** Consider a two-person game (for example, some non-trivial version of tic-tac-toe). Write game-definition relations (legal moves and terminal game positions) and propose a static evaluation function to be used for playing the game with the alpha-beta procedure. Use the principle of alpha beta to reduce the search in the tree mentioned above.

## 4.3. Iterative Deepening Techniques

The goal is to prove that search is ubiquitous in artificial intelligence. The performance of most AI systems is dominated by the complexity of a search algorithm in their inner loops. Prove with an example that this algorithm gives optimal solution for exponential tree searches.

**Input:** Starting node and goal node.

**Output:**

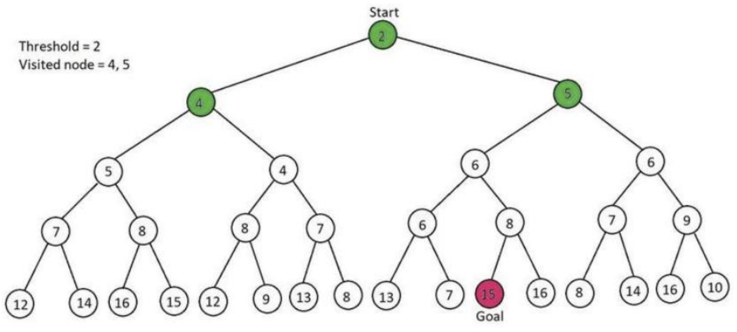
a. Complete the search process if the branching factor is finite and there is a solution at some finite depth and obtain optimal in finding the shortest solution first.
b. Avoid exploring each non-solution branch of the tree, omit cycle detection and retain completeness.
c. Use additional logical features of Prolog to terminate the search process whenever if there are no solutions identified even after backtracking.
d. Document the steps if the search process does not obtain optimal solution even after backtracking.

```
      A
     / \
    B   C
   / \  | \
  D   E F  G
      |
      H
```

**Hints**

```
/* Define a class to initialize the graph, start node, and goal node. */

/* Perform the DLS to a specified depth with current node, current depth limit, set
of visited nodes and return the tuple with a path cost and the path. */

/* Perform the iterative deepening DFS with maximum depth limit to explore.*/

/* Generate the optimal path if found otherwise generate None. */

/* Apply backtracking if the solution is not found. */
```

**Try:** The 15-puzzle problem is a classic example of a **sliding puzzle game**. It consists of a 4×4 grid of numbered tiles with one tile missing. The aim is to rearrange the tiles to form a specific goal configuration. The state space of the puzzle can be represented as a tree where each node represents a configuration of the puzzle, and each edge represents a legal move. IDA* can be used to find the **shortest sequence of moves** to reach the goal state from the initial state.



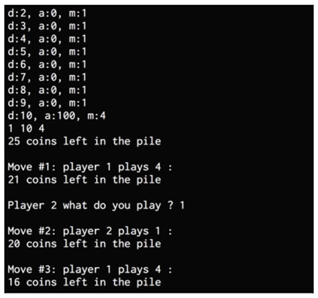
## 5. Exercises on Building Games with AI

### 5.1 Building a Bot to Play Last Coin Standing

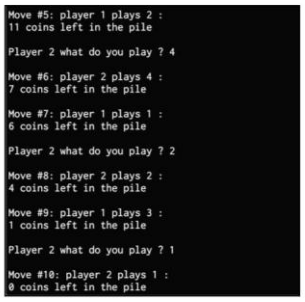
This is a game where we have a pile of coins, and each player takes turns to take a number of coins from the pile. There is a lower and an upper bound on the number of coins that can be taken from the pile. The goal of the game is to avoid taking the last coin in the pile. This recipe is a variant of the Game of Bones recipe given in the easyAI library. Develop the python code to build a game where the computer can play against the user.

**Input:** Libraries like TwoPlayerGame, id_solve, Human_Player, AI_Player, and TT

**Output:** Force the computer to take the last coin, so that you win the game.



```
d:2, a:0, m:1
d:3, a:0, m:1
d:4, a:0, m:1
d:5, a:0, m:1
d:6, a:0, m:1
d:7, a:0, m:1
d:8, a:0, m:1
d:9, a:0, m:1
d:10, a:100, m:4
1 10 4
25 coins left in the pile

Move #1: player 1 plays 4 :
21 coins left in the pile

Player 2 what do you play ? 1

Move #2: player 2 plays 1 :
20 coins left in the pile

Move #3: player 1 plays 4 :
16 coins left in the pile
```



```
Move #5: player 1 plays 2 :
11 coins left in the pile

Player 2 what do you play ? 4

Move #6: player 2 plays 4 :
7 coins left in the pile

Move #7: player 1 plays 1 :
6 coins left in the pile

Player 2 what do you play ? 2

Move #8: player 2 plays 2 :
4 coins left in the pile

Move #9: player 1 plays 3 :
1 coins left in the pile

Player 2 what do you play ? 1

Move #10: player 2 plays 1 :
0 coins left in the pile
```

**Hints**

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player from easyAI.AI
import TT
% Create a class to handle all the operations of the game.


/* Define who is going to start the game */


/* Define the maximum number of coins that can be taken out in any move. */


/* Define all the possible moves, define a method to remove the coins and keep track
of the number of coins remaining in the pile.  */


/* Check if somebody won the game by checking the number of coins remaining. */


/* Stop the game after somebody wins it and compute the score based on the win method.
*/


/* Define a method to show the current status of the pile. */


/* Define the main function and start by defining the transposition table. */


/* Define the method ttenttry to get the number of coins */
```

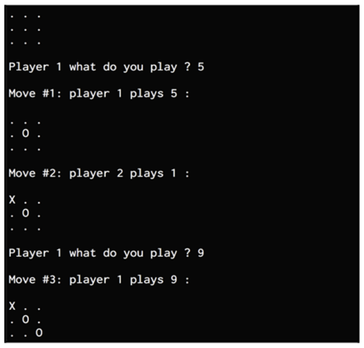
**Try:** Implement the same code so that you can win the game instead of a computer.

## 5.2 Building two Bots to Play Tic-Tac-Toe

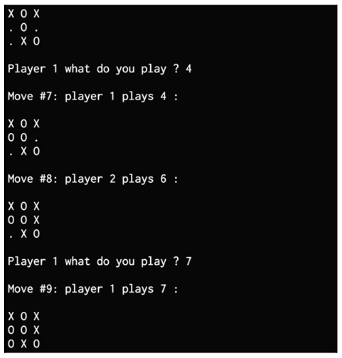
Tic-Tac-Toe (Noughts and Crosses) is probably one of the most famous games. Let's see how to build a game where the computer can play against the user. This is a minor variant of the Tic-Tac-Toe recipe given in the easyAI library.

**Input:** Libraries like TwoPlayerGame, id_solve, Human_Player, and AI_Player.

**Output:** Force the computer to take the last coin, so that you win the game.



```
. . .
. . .
. . .

Player 1 what do you play ? 5

Move #1: player 1 plays 5 :

. . .
. O .
. . .

Move #2: player 2 plays 1 :

X . .
. O .
. . .

Player 1 what do you play ? 9

Move #3: player 1 plays 9 :

X . .
. O .
. . O
```



```
X O X
. O .
. X O

Player 1 what do you play ? 4

Move #7: player 1 plays 4 :

X O X
O O .
. X O

Move #8: player 2 plays 6 :

X O X
O O X
. X O

Player 1 what do you play ? 7

Move #9: player 1 plays 7 :

X O X
O O X
O X O
```

### Hints

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player from easyAI.AI
import TT
% Create a class to handle all the methods to play the game.

/* Define a method to compute all the possible moves by defining a 3 x 3 board. */

/* Define the method to update the board after making a move. */

/* Define a method to see if somebody has lost the game and check if the game is over
using the loss_condition method.  */

/* Define a method to show the current progress and compute the score using the
loss_condition method. */

/* Define the main function and start by defining the algorithm and start the game.
*/
```

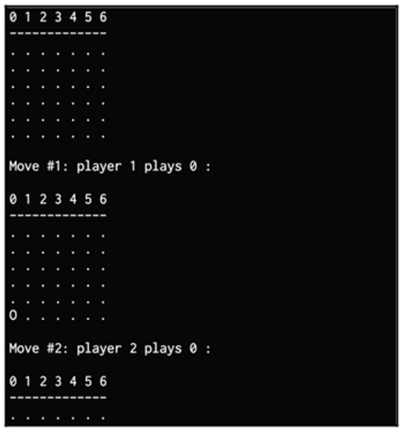
**Try:** Implement the same code so that you can win the game instead of a computer.

## 5.3 Building two Bots to Play Connect Four against each other

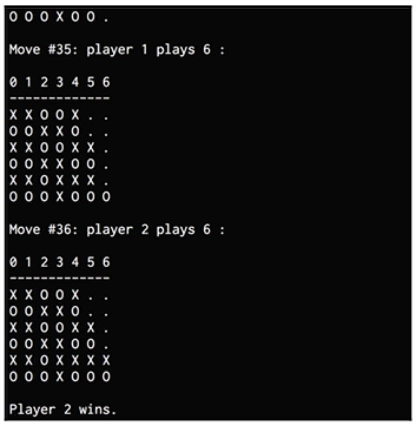
Connect Four™ is a popular two-player game sold under the Milton Bradley trademark. It is also known by other names such as Four in a Row or Four Up. In this game, the players take turns dropping discs into a vertical grid consisting of six rows and seven columns. The goal is to get four discs in a line. This is a variant of the Connect Four recipe given in the easyAI library. Develop the python code to build it instead of playing against the computer, we will create two bots that will play against each other.

**Input:** Libraries like TwoPlayerGame, id_solve, Human_Player, and AI_Player.

**Output:** Algorithm for each bot and see which one wins.



```
0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .

Move #1: player 1 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
O . . . . . .

Move #2: player 2 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
```



```
O O O X O O .

Move #35: player 1 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X .
O O O X O O O

Move #36: player 2 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X X
O O O X O O O

Player 2 wins.
```

**Hints**

```
% Create a new Python file and import the following packages:
from easyAI import TwoPlayersGame, id_solve, Human_Player, AI_Player, Negamax, and
SSS
from easyAI.AI import TT
% Create a class that contains all the methods needed to play the game.

/* Define the board with six rows and seven columns. */

/* Define who's going to start the game while defining the board positions. */

/* Define a method to get all the possible moves.  */

/* Define a method to control how to make a move and to show the current status. */

/* Define a method to compute what a loss looks like and check whether the game is
over or not. */

/* Compute the score to decide the winning bot */
```

**Try:** Implement the same code so that SSS algorithm can win the game instead of a Negamax algorithm.

# 6. Exercises on Logic Programming

## 6.1 Water Jug Problem

Given two jugs, a 4-gallon one and a 3-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 2 gallons of water into a 4-gallon jug?

**Input:** 4 3 2

**Output:** {( 0,0),(0,3),(3,0),(3,3),(4,2),(0,2)}

a. Describe the state space as a set of ordered pairs of integers.
b. Generate production rules and perform basic operations to achieve the goal.
c. Initialize the start state and apply the rules iteratively until the goal state is reached.
d. Generate a search tree (Depth-First Search / Breadth-First Search)

**Hints**

```
/* Define a function to initialize the dictionary elements with a default value.*/

/* Initialize dictionary with default values as false.*/

/* Define a recursive function and print the intermediate steps to reach the final
solution and return the Boolean values.*/

/* Check whether the goal is achieved and return true if achieved. */

/* Check if you have already visited the combination or not. If not, then proceed
further. */

/* Check whether all the six possibilities and see if a solution is found in any one
of them. */

/* Return false if the combination is already visited to avoid repetition otherwise
recursion will enter an infinite loop*/

/* Call the function and pass the initial amount of water present in both the jugs.
*/
```

**Try:** Given two jugs, a 7-gallon one and a 11-gallon one. Neither has any measuring markers on it. There is a pump that can be used to fill the jugs with water. How can we get exactly 7 gallons of water into a 7-gallon jug?

## 6.2 Monkey Banana Problem

Imagine a room containing a monkey, chair and some bananas that have been hung from the center of ceiling. If the monkey is clever enough, he can reach the bananas by placing the box directly below the bananas and climbing on the chair .The problem is to prove whether the monkey can reach the bananas. The monkey wants it but cannot jump high enough from the floor. At the window of the room there is a box that the monkey can use.

**Input**: The monkey can perform the following actions:-

1) Walk on the floor.
2) Climb the box.
3) Push the box around (if it is beside the box).
4) Grasp the banana if it is standing on the box directly under the banana.

**Output:**

a. Write down the initial state description and action schemes.
b. Prepare all the required predicates that will make the monkey to perform some action and move from one state to the other until the goal state is reached.
c. Set the initial position of the monkey (initial state) and raise questions to whether the knowledge represented can make the monkey get the banana.
d. Trace the flow of actions from initial state to goal state.

**Hints**

```
/* Define a class and initialize all the states */

/* Check if the goal state is achieved.*/

/* Return a list of possible actions given the current state */

/* Apply an action and return the resulting state. */

/* Solve the problem using breadth-first search by simultaneously checking the goal
state, generating the successors */

/* Return result if exist otherwise return None */
```

**Try:** Extensively make use of state space search to represent and solve Tic-Tac-Toe. Represent the problem as a state space and define the rules. In the state space, represent the starting state, set of legal moves, and the goal state.

## 6.3 Eight Puzzle Problem

The 8-puzzle consists of a 3×3 board with eight numbered tiles and a blank space. A tile adjacent to the blank space can slide into the space. The task is to reach a specified goal state, such as the one shown on the right of the figure. The objective is to place the numbers on tiles to match the final configuration using the empty space. You can slide four adjacent (left, right, above, and below) tiles into the empty space.

**Input**:

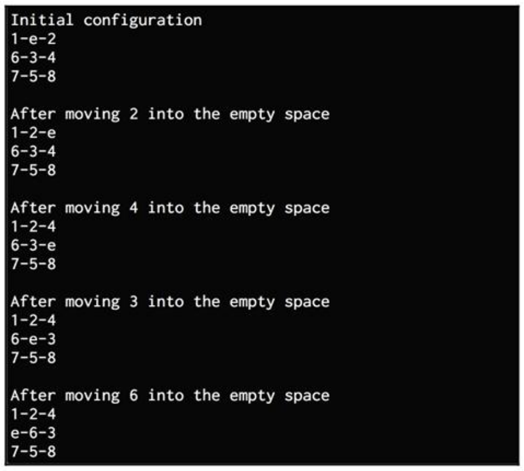
Initial State

| | | |
|---|---|---|
| | 2 | 3 |
| 1 | 4 | 6 |
| 7 | 5 | 8 |

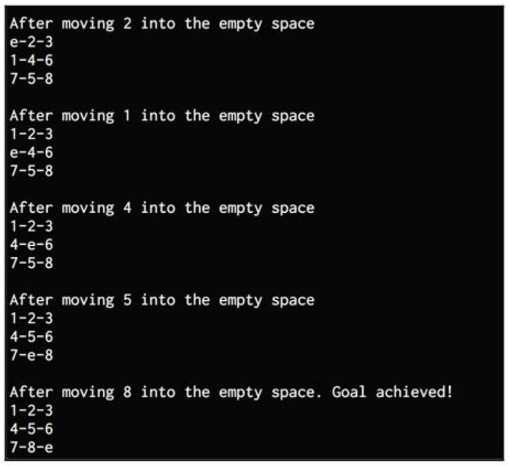
Goal State

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |   |

**Output**: Goal state as a long output on the terminal just like:



```
Initial configuration
1-e-2
6-3-4
7-5-8

After moving 2 into the empty space
1-2-e
6-3-4
7-5-8

After moving 4 into the empty space
1-2-4
6-3-e
7-5-8

After moving 3 into the empty space
1-2-4
6-e-3
7-5-8

After moving 6 into the empty space
1-2-4
e-6-3
7-5-8
```

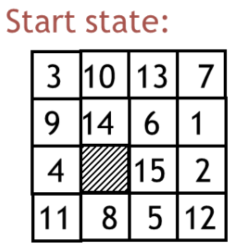


```
After moving 2 into the empty space
e-2-3
1-4-6
7-5-8

After moving 1 into the empty space
1-2-3
e-4-6
7-5-8

After moving 4 into the empty space
1-2-3
4-e-6
7-5-8

After moving 5 into the empty space
1-2-3
4-5-6
7-e-8

After moving 8 into the empty space. Goal achieved!
1-2-3
4-5-6
7-8-e
```

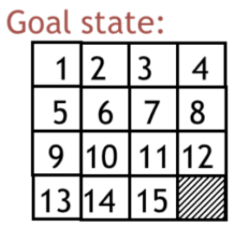
## Hints

```
/* Create a class that contains the methods to solve the 8-puzzle by importing the
following packages. */
from simpleai.search import astar, SearchProblem

/* Define a class that contains the methods to solve the 8-puzzle: */

/* Override the actions method to align it with our problem: */

/* Check the location of the empty space and create the new action: */

/* Check if the goal has been reached.*/

/* Define the heuristics method and compute the distance.*/

/* Define a function to convert a list of string */

/* Define a function to convert a string to a list. */

/* Define a function to get the location of a given element in the grid. */


/* Define the initial state and the final goal we want to achieve: */

/* Track the goal positions for each piece by creating a variable: */

/* Create the A* solver object using the initial state we defined earlier and extract
the result: */

/* Print the solution */
```

**Try**: Extensively make use of state space search to represent and solve the 15 Puzzle problem. Represent the problem as a state space and define the rules.


Start state:

| | | | |
|---|---|---|---|
| 3 | 10 | 13 | 7 |
| 9 | 14 | 6 | 1 |
| 4 | ▨ | 15 | 2 |
| 11 | 8 | 5 | 12 |

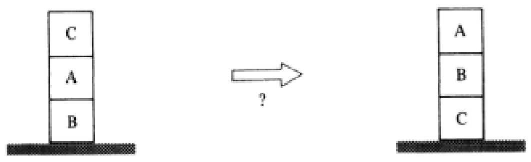

Goal state:

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | ▨ |

## 6.4 Blocks Rearrangement Problem

The problem is to find a plan for rearranging a stack of blocks as shown below. We are allowed to move one block at a time. A block can be grasped only when its top is clear. A block can be put on the table or on some other blocks. To find a required plan, we must find a sequence of moves that accomplish the given transformation. Think the problem as a problem of exploring among possible alternatives.

**Input:**



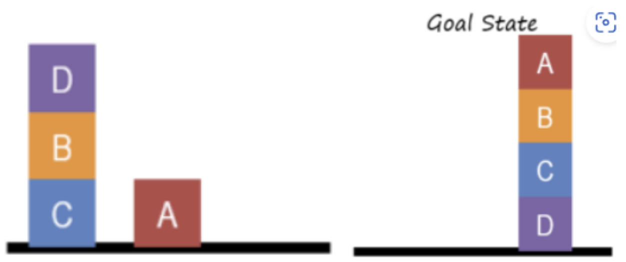
**Output:**

a. Generate the rules involving accomplishing various tasks involving the blocks world.
b. Formulate the more careful definitions for program and the actions.
c. Present the graphical representation of the problem (state space representations) including initial state and the goal state.

**Hints**

```
/* Define a class to initialize all the starting states of all the blocks. */

/* Do the same for the goal state also.*/

/* Check whether the current state matches the goal state. */

/* Generate all possible moves from the current state. */

/* Allow any block can be moved to the table or on top of another block. */

/* If a block is not blocked by another, it can be moved. Move onto another block.*/

/* Apply a move to the current state and return the new state.

/* Solve the problem using BFS by checking if the goal state is reached by getting
all the possible moves . */

/* Return the solution if found otherwise return None. */
```

**Try:** The problem is to find a plan for rearranging a stack of blocks as shown below. We are allowed to move one block at a time. A block can be grasped only when its top is clear. A block can be put on the table or on some other blocks. To find a required plan, we must find a sequence of moves that accomplish the given transformation. Think of the problem as a problem of exploring among possible alternatives.



The goal here is to move Block B from the middle of the pile on the left and onto the top of the pile on the right. Hence this sequence of moves would be an acceptable solution:

[("C", "Table"), ("B", "E"), ("C", "A")]

## 7. Exercises on Knowledge Representation using FOL

### 7.1 Knowledge Representation using FOL – Translate

The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL). Later convert the FOL into a prolog program and asking questions.

**Input**: Domain Knowledge like:

(a) Not all cars have carburetors.
(b) Some people are either religious or pious.
(c) No dogs are intelligent.
(d) All babies are illogical.
(e) Every number is either negative or has a square root.
(f) Some numbers are not real.
(g) Every connected and circuit-free graph is a tree.

**Output**: Equivalent FOL statement.

**Hints**

$$\neg\forall x\ [car(x) \rightarrow carburetors(x)] \text{ or}$$
$$\exists x\ [car(x) \wedge \neg carburetors(x)]$$

```
# Translate the other statements referring the above
```

**Try**: Translate each of the following sentences into First Order Logic (FOL):

(a) Not every graph is connected.
(b) All that glitters is not gold.
(c) Not all that glitters is gold.
(d) There is a barber who shaves all men in the town who do not shave themselves.
(e) There is no business-like show business.

## 7.2 Knowledge Representation using FOL - Translate

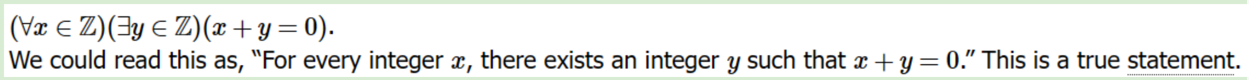
The goal of this exercise is to rewrite each proposition symbolically, given that the universe of discourse is a set of real numbers.

**Input**: Domain Knowledge like:

(a) For each integer x, there exist an integer y such that x + y = 0.
(b) There exist an integer x such that x + y = y for every integer y.
(c) For all integers x and y, x.y = y.x
(d) There are integers x and y such that x+y=5.

**Output**: Equivalent FOL statement

**Hints**



$(\forall x \in \mathbb{Z})(\exists y \in \mathbb{Z})(x + y = 0)$.

We could read this as, "For every integer $x$, there exists an integer $y$ such that $x + y = 0$." This is a true statement.

```
# Translate the other statements referring the above
```

**Try**: Using FOL, express the following:

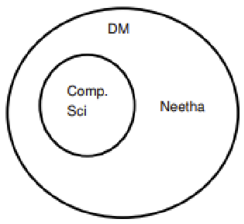
(a) Every student in this class has taken exactly two mathematics courses at this school.
(b) Someone has visited every country in the world except Libya.
(c) No one has climbed every mountain in the Himalayas.

## 7.3 Check the Availability

Every computer science student takes discrete mathematics. Neetha is taking discrete mathematics. Therefore, Neetha is a computer science student. The given conclusion is false. The following Venn diagram is a counter example for the given conclusion.

If it does not rain or it is not foggy then the sailing race will be held, and lifesaving demonstrations will go on. If the sailing race is held, then the trophy will be awarded. The trophy was not awarded. Therefore, it rained. The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL)

**Input**: Venn Diagram as a counter example for the given conclusion.



**Output**: Prove that the above statements are TRUE.

**Hints**

```
# Consider the below statements and infer the statement by the following arguments
```

$$\begin{array}{lll}
premise\ \neg R \lor \neg F \to S \land D & & \ldots(1) \\
premise\ S \to T & & \ldots(2) \\
premise\ \neg T & & \ldots(3) \\
1 & \neg R \lor \neg F \to S & \ldots(4) \\
4, 2 & \neg R \lor \neg F \to T & \ldots(5) \\
5 & \neg T \to \neg(\neg R \lor \neg F) & \ldots(6) \\
6, 3 & \neg(\neg R \land \neg F) & \ldots(7) \\
7 & R \land F & \ldots(8) \\
8 & R &
\end{array}$$

**Try**: Prove or Disprove: All doctors are college graduates. Some doctors are not golfers. Hence, some golfers are not college graduates.

# 8. Exercises on First Order Logic ( FOL )

## 8.1. Rewrite the sentences.

The goal of this exercise is to translate each of the following sentences into First Order Logic (FOL). Later convert the FOL into a prolog program and asking questions.

**Input**:

(a) Some boys are sharp and intelligent.
UOD(x): all persons.
Sharp(x): x is sharp.
Boy(x): x is a boy.
Intelligent(x): x is intelligent.

(b) Not all boys are intelligent.

(c) Some students of DM course have cleared JEE main and the rest cleared SAT.
UOD(x): all persons.
ClearJEE(x): x clears JEE main.
ClearSAT(x): x clears SAT.

(d) Something that is white is not always milk, whereas the milk is always white.
UOD(x): things.
White(x): x is white.
M ilk(x): x is milk.

(e) Breakfast is served in mess on all days between 7am and 9am except Sunday. And, on Sundays it is served till 9.15 am. UOD(x): days. Day(x): x is a day of the week. Breakfast-time-non-sunday(x): Breakfast is served in mess on x between 7am and 9am. Breakfast-time-sunday(x): Breakfast is served in mess on x till 9.15am.

**Output**: Equivalent FOL statement.

### Hints

$\exists x \mathrm{boys}(x) \land \mathrm{intelligent}(x)$

```
# Translate the other statements referring the above
```

**Try**: Translate each of the following sentences into First Order Logic (FOL):

(a) The speed of light is not same in all mediums. The speed of light in fiber is 2×108 m/s. Therefore, there exists at least two mediums having different speed of light. UOD(x): mediums. Medium(x): Light travels in medium x. Speed(x): Speed of light in medium x. P: Speed of light in fiber is 2 × 108 m/s.

(b) Some students have joined IIITDM. There exists a student who has not joined any IIITDM. Not all students have cleared JEE advanced. Therefore, some students have joined deemed universities. UOD(x) : people. UOD(y) : Educational institutes. Stud(x): x is a student. IIIT DM(y) : y is a IIITDM. JoinIIIT DM(x, y) : x joins IIITDM y. ClearJEE(x) : x cleared JEE advanced. JoinDeemed(x) : x joins a deemed university.

## 8.2 Propositions

Identify propositions from the following. If not a proposition, justify, why it is not.

(a) I shall sleep or study.

(b) x 2 + 5x + 6 = 0 such that x ∈ integers.

**Input:** Propositions
**Output**: Justify the statements either to be a proposition or not

### Hints

```
# The rule of logic allows to distinguish between valid and invalid arguments.
Example:
If x+1=5, then x=4=4. Therefore, if x≠4, then x+1≠5.
If I watch Monday night football, then I will miss the following Tuesday 8 a.m. class.
Therefore, if I do not miss my Tuesday 8 a.m. class, then I did not watch football
the previous Monday night.
# Use the same format:
If p then q. Therefore, if q is false then p is false.
If we can establish the validity of this type of argument, then we have proved at
once that both arguments are legitimate. In fact, we have also proved that any
argument using the same format is also credible.
# Use the above example and give the justifications
```

Try: Express the following in first order logic (identify the right universe of discourse, predicates before attempting each question. Think twice and do not oversimplify the problem)

(a) The fundamental law of nature is change.

(b) We cannot help everyone, but everyone can help someone.

(c) Power does not corrupt people, people corrupt power.

(d) It is nice of somebody to do something.

(e) No one who has no complete knowledge of himself will ever have a true understanding of another.

(f) Thought or thinking is what set human beings apart from other living things.

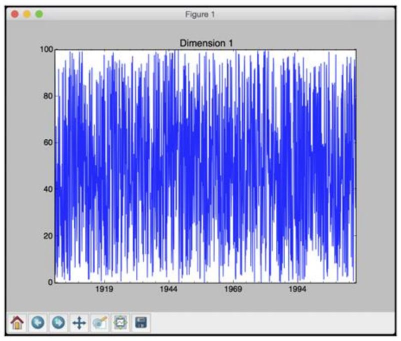
# 9. Exercises on Probabilistic Reasoning for Sequential Data

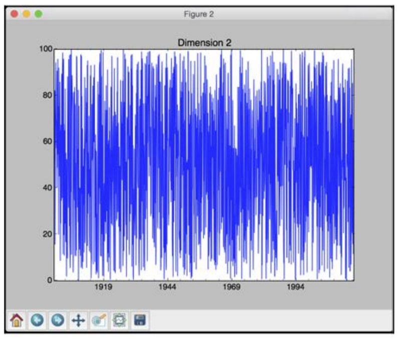
## 9.1. Handling Time-Series Data with Pandas

Time-series data analysis is used extensively in financial analysis, sensor data analysis, speech recognition, economics, weather forecasting, manufacturing, and many more. Explore a variety of scenarios where we encounter time-series data and see how a solution can be built. Create a python file and learn how to handle time-series data in Pandas.

**Input:** Time-series dataset

**Output:** Time-Series data with different dimensions like:





### Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

/* Define a function to read the data from the input file. */

/* Define a lambda function to convert strings to Pandas date format: */

/* Use this lambda function to get the start date from the first line in the input
file: */

/* Create a list of indices with dates using the start and end dates with a monthly
frequency: */

/* Create pandas data series using the timestamps: */


/* Define the main function and specify the input file:, Specify the columns that
contain the data:*/


/* Iterate through the columns and read the data in each column and Plot the time-
series data. */
```

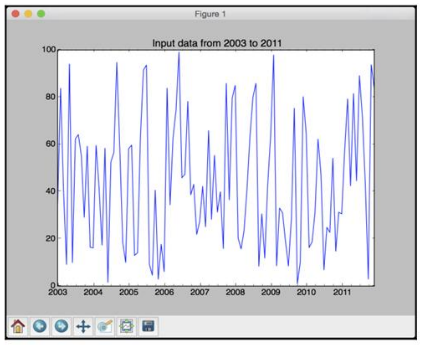
**Try:** Revise the above code to analyze and visualize the time-series data for three and more dimensions.

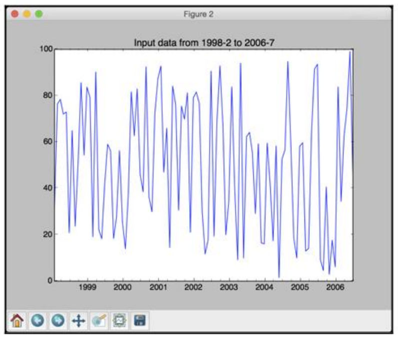
## 9.2. Slicing Time-Series Data

The process of slicing refers to dividing the data into various sub-intervals and extracting relevant information. This is very useful when you are working with time-series datasets. Instead of using indices, we will use timestamp to slice our data. Develop a python code to analyze the time-series data and visualize the same at different intervals.

**Input:** Time-series dataset

**Output:** Visualize the Time-Series data with different levels of granularity.





### Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

/* Define a function to read the data from the input file. */

/* Define a lambda function to convert strings to Pandas date format: */

/* Use this lambda function to get the start date from the first line in the input
file: */

/* Create a list of indices with dates using the start and end dates with a monthly
frequency: */

/* Create pandas data series using the timestamps: */


/* Define the main function and specify the input file:, Specify the columns that
contain the data:*/
```

```
/* Iterate through the columns and read the data in each column and Plot the time-
series data. */
```

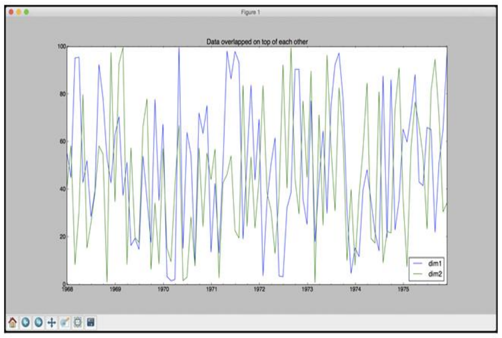
**Try:** Revise the above code to analyze and visualize the time-series data for different level of granularities.

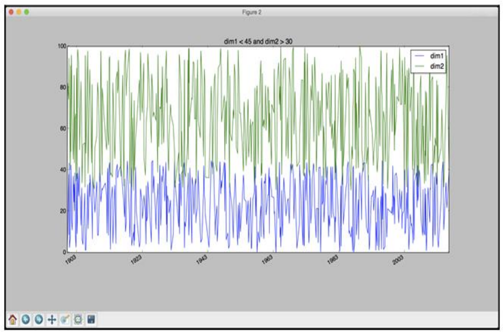
## 9.3. Operating on Time-Series Data

Pandas allow us to operate on time-series data efficiently and perform various operations like filtering and addition. You can simply set some conditions and Pandas will filter the dataset and return the right subset. Develop a python code that can allow to build various similar applications without having to reinvent the wheel.

**Input:** Time-series dataset

**Output:** Visualize the Time-Series data with different data frames.





### Hints

```
/* Create a python file and import the following packages */

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from timeseries import read_data

/* Define the input filename and load the third and fourth columns into separate
variables. */

/* Create a Pandas dataframe object by naming the two dimensions and plot the data
by specifying the start and end years */

/* Filter the data using conditions and then display it. */

/* Display the summarized results also */
```

**Try:** Add two series in Pandas and different dimensions between the given start and end dates.

# 10. Exercises on Expert Systems

## 10.1 Identify Animals

The goal is to create an expert system that can identify animals. We can use the rules of inference that we have learned about animals to perform this task. These rules serve as a starting point for developing an expert system, and they show the importance of having input from the users. The goal of an expert system is to provide useful information based on its users' inputs.

**Input**:

- If it has a tawny color and has dark spots, then the animal is a cheetah.
- If it has a tawny color and has black stripes, then the animal is a tiger.
- If it has a long neck and has long legs, then the animal is a giraffe.
- If it has black stripes, then the animal is a zebra.
- If it does not fly and has long neck, then the animal is an ostrich.
- If it does not fly, swims, black and white in color, then the animal is penguin.
- If it appears in story ancient mariner and flys well, then the animal is albatross.

**Output**:

a. Create an expert system that can identify the animal class using the inference rules.
b. Utilize the user inputs and predict the animal class based on the behaviors already learned by the expert system.

```
Welcome to the Animal Classifier Expert System!
Please answer the following questions with 'yes' or 'no':

Is the animal a vertebrate (has a backbone)? yes
Is the animal cold-blooded? yes
Does the animal have scales? yes

The animal class is: Fish
```

**Hint:**

```
/* Define a class to initialize an expert system with predefined inference rules. */

/* Define a class to predict the animal class based on user inputs. */

/* Define the features to ask the user and collect the user inputs. */

/* Create and use the expert systems. */

/* Display the results by predicting the animal class, other generate the results as
“unable to determine.” */
```

**Try**: Consider the if-then rules of figures and translate them into our rule notation. Propose extensions to the notation to handle certainty measures when needed.

*if*

1 the infection is primary bacteremia, and
2 the site of the culture is one of the sterilesites, and
3 the suspected portal of entry of the organism is the gastrointestinal tract

*then*

there is suggestive evidence (0.7) that the identity of the organism is bacteroides.

*if*

1 there is a hypothesis, $H$, that a plan $P$ succeeds, and
2 there are two hypotheses,
$H_1$, that a plan $R_1$ refutes plan $P$, and
$H_2$, that a plan $R_2$ refutes plan $P$, and
3 there are facts: $H_1$ *is false*, and $H_2$ *is false*

*then*

1 generate the hypothesis, $H_3$, that the combined plan '$R_1$ *or* $R_2$' refutes plan $P$, and
2 generate the fact: $H_3$ *implies not(H)*

## 10.2 Locating Failures in a Simple Electric Network

Create a knowledge base which can help locating failures in a simple electric network that consists of some electric devices and fuses. Such a network is shown in figure.

*if*

light1 is on *and*
light1 is not working *and*
fuse1 is proved intact

*then*

light1 is proved broken.

Another rule can be:

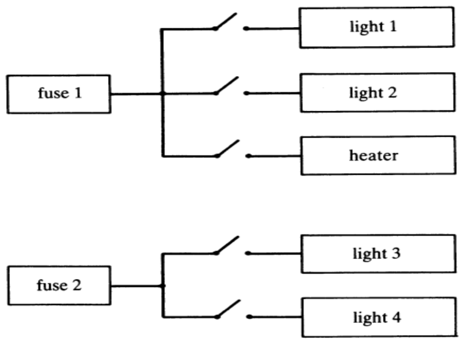
*if*

heater is working

*then*

fuse1 is proved intact.

Input:



**Hint:**

```
% A small knowledge base for locating faults in an electric network
% If a device is on and not working and its fuse is intact then the device is broken

broken_rule:
        if
                on(Device) and
                device(Device) and
                not working(Device) and
                connected(Device, Fuse) and
                proved(intact(Fuse))
        then
                proved(broken(Device))
% If a unit is working then its fuse is OK
fuse_ok_rule:
        if
                connected(Device,Fuse) and
                working(Device)
        then
                proved(intact(Fuse)).

% If two different devices are connected to a fuse and are both on and not working
% then the fuse has failed.
% NOTE: This assumes that at most one device is broken!

fused_rule:
        if
                connected(Device1, Fuse) and
                on(Device1) and
                not working(Device1) and
                samefuse(Device2, Device1) and
                on(Device2) and
                not working(Device2)
        then
                proved(failed(Fuse)).
same_fuse_rule:
        if
                connected(Device1, Fuse) and
                connected(device2, Fuse) and
                different(Device1, Device2)
        then
                samefuse(Device1, Device2).
fact: different(X, Y) :- not(X=Y).

fact: device(heater).
fact(device(light1).
fact(device(light2).
fact(device(light3).
fact(device(light4).
fact: connected(light1, fuse1).
fact: connected(light2, fuse1).
fact: connected(heater, fuse1).
fact: connected(light3, fuse2).
fact: connected(light4, fuse2).

askable(on(D), on('Device')).
askable(working(D), working('Device')).
```

**Try:** Think of some decision problem and try to formulate the corresponding knowledge in the form of if-then rules. You may consider choice of holiday, weather prediction, simple medical diagnosis, and treatment, etc.

## 11. Exercises on Expert Systems

### 11.1 Mental Health Disorder

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program with an idea of identifying the health disorder based on the database provide with mental health conditions.
**Input:** Set of logical rules with mental health conditions.

**Output:** Model the disorder of the patient health.

**Hints**

```
diagnose :-
```

```
 write('This is an expert system for dignosis of mental disorders.'), nl,
 write('There are several questions you need to answer for dignosis of mental
disorders.'), nl, nl,
 disorder(X),
 write('Condition was diagnosed as '),
 write(X),
 write('.').
diagnose :-
    write('The diagnose was not found.').


%The question predicate will have to determine from the user
%whether or not a given attribute-value pair is true
question(Attribute, Value):-
 retract(yes, Attribute, Value), !.
question(Attribute, Value):-
 retract(_, Attribute, Value),  !, fail.
question(Attribute, Value):-
 write('Is the '),
 write(Attribute),
 write(' - '),
 write(Value),
 write('?'),
 read(Y),
 asserta(retract(Y, Attribute, Value)),
 Y == yes.

%question with additional argument which contains
%a list of possible values for the attribute.
questionWithPossibilities(Attribute, Value, Possibilities) :-
 write('What is the patient`s '), write(Attribute), write('?'), nl,
 write(Possibilities), nl,
 read(X),
 check_val(X, Attribute, Value, Possibilities),
 asserta( retract(yes, Attribute, X) ),
 X == Value.

check_val(X, _, _, Possibilities) :-  member(X, Possibilities),
 !.
check_val(X, Attribute, Value, Possibilities) :-
 write(X), write(' is not a legal value, try again.'), nl,
 questionWithPossibilities(Attribute, Value, Possibilities).

%retract equips this system with a memory that remembers the facts that are already
%known because they were already entered by the user at some point during the
interaction.
:- dynamic(retract/3).

%. The program needs to be modified to specify which attributes are askable
food_amount(X) :- question(food_amount,X).
symptom(X) :- question(symptom,X).
mentality(X) :- question(mentality,X).
cause(X) :- question(cause, X).
indication(X) :- question(indication,X).
social_skill(X) :- question(social_skill,X).
```

```
condition(X) :- question(condition, X).
consequence(X) :- question(consquence,X).
specialty(X) :- question(specialty,X).
face_features(X) :- question(face_features,X).
ears_features(X) :- question(ears_features,X).
brain_function(X) :- question(brain_function,X).
perceptions(X) :- question(perceptions, X).
behavior(X)  :-  questionWithPossibilities(behavior,  X,  [repetitive_and_restricted,
narcissistic, aggresive]).


disorder(anorexia_nervosa) :- type(eating_disorder),
                               consequence(low_weight),
                               food_amount(food_restriction).

disorder(bulimia_nervosa) :- type(eating_disorder),
                              consequence(purging),
                     food_amount(binge_eating).

disorder(asperger_syndrome) :- type(neurodevelopmental_disorder),
                                specialty(psychiatry),
                                social_skill(low),
                                behavior(repetitive_and_restricted).

disorder(dyslexia) :- type(neurodevelopmental_disorder),
                      social_skill(normal),
                      perceptions(low),
                      symptom(trouble_reading).

disorder(autism) :- type(neurodevelopmental_disorder),
                    social_skill(low),
                    symptom(impaired_communication).


disorder(tourettes_syndrome) :- type(neurodevelopmental_disorder),
                                 social_skill(normal),
                                 specialty(neurology),
                                 symptom(motor_tics).

disorder(bipolar_disorder) :- type(psychotic_disorder),
                               indication(elevated_moods).

disorder(schizophrenia) :- type(psychotic_disorder),
                            indication(hallucinations).

disorder(down_syndrome) :- type(genetic_disorder),
                            symptom(delayed_physical_growth),
                            face_features(long_and_narrow),
                            ears_features(large),
                            brain_function(intellectual_disability).

disorder(fragile_X_syndrome) :- type(genetic_disorder),
                                 face_features(small_chin_and_slanted_eyes),
                                 brain_function(intellectual_disability).
```

```
type(eating_disorder) :- symptom(abnormal_eating_habits),
                          mentality(strong_desire_to_be_thin).

type(neurodevelopmental_disorder) :-  condition(affected_nervous_system),
                                       brain_function(abnormal),
                                       cause(genetic_and_enviromental).

type(psychotic_disorder) :- symptom(false_beliefs),
                             mentality(manic_depressive),
                             cause(genetic_and_enviromental).

type(genetic_disorder) :- cause(abnormalities_in_genome).
```

**Try:** Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

## 11.2 War Crimes Explorer

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program with an idea of:

- In-browser learning about genocide, war crimes, crimes against humanity, and aggression.
- Interactively enter facts and discover the laws that might have been broken.
- Possibly submit the information to the ICC (International Criminal Court) as a witness statement.

**Input:** Set of logical rules with laws and crimes within the jurisdiction of the International Criminal Court.

**Output:** Model the legal statutes.

### Hints

```
/*
 * Crimes within the jurisdiction of the International Criminal Court.
 *
 *
https://world.public.law/rome_statute/article_5_crimes_within_the_jurisdiction_of_t
he_court
 */
crime(genocide).
crime(war_crime).
```

~~~
crime(crime_against_humanity).
crime(crime_of_aggression).


/*
 * A first, simple attempt at Protected Persons under
 * the Geneva Conventions of 1949.
 */
protected_by_geneva_convention(P) :- civilian(P).
protected_by_geneva_convention(P) :- prisoner_of_war(P).
protected_by_geneva_convention(P) :- medical_personnel(P).
protected_by_geneva_convention(P) :- religious_personnel(P).

/*
 * D = Defendant
 * V = Victim
 */

/*
 * Genocide
 * https://world.public.law/rome_statute/article_6_genocide
 */
criminal_liability(genocide, Statute, D, V) :-
        elements(Statute, D, V).

/*
 * War crimes
 * https://world.public.law/rome_statute/article_8_war_crimes
 */
criminal_liability(war_crime, Statute, D, V) :-
        protected_by_geneva_convention(V),
        international_conflict(D, V),
        elements(Statute, D, V).


elements(article_8_2_a_i, D, V) :-
        act(D, killed, V).

elements(article_8_2_a_ii, D, V) :-
        act(D, tortured, V).
```
~~~

**Try:** Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

## 11.3 DP Film Expert System

Create an expert system in prolog to improve the understanding of declarative programming paradigm based on the logic rules. Develop the program in such a way that it allows you to get film recommendations based on your answer and logic rules with a little film database.

**Input:** Set of logical rules with some film database.

**Output:** Recommend a file based on the persons name, mood, sex, time they have, and type of films interested.

### Hints

```
# Sample database
/** DRAMA */
film('Zielona  mila','Frank  Darabont',1999,'drama',  'others',    188,'USA','Tom Hanks',8.7,719).
film('Pif Paf! Jestes trup', 'Guy Ferland', 2002, 'drama', 'others',  87, 'Kanada', 'Ben Foster', 7.7, 16).
film('Dogville', 'Lars von Trier', 2003, 'drama', 'others', 178, 'Dania', 'Nicole Kidman', 7.7, 45).
film('Z dystansu', 'Ton Kayne', 2011, 'drama', 'others', 100, 'USA', 'Adrien Brody', 7.9, 30).
film('Lista Schindlera', 'Steven Spielberg', 1993, 'drama', 'others', 195, 'USA', 'Adrien Brody', 8.4, 259).
film('Requiem dla snu', 'Darren Aronofsky', 2000, 'drama', 'others', 102, 'USA', 'Jared Leto', 7.9, 520).
film('Biutiful', 'Alejandro Gonzalez Inarritu', 2010, 'drama', 'others', 148, 'Hiszpania', 'Javier Bardem', 7.6, 12).
film('Czarny labedz', 'Darren Aronofsky', 2010, 'drama', 'others', 108, 'USA', 'Natalie Portman', 7.7, 248).
film('Gladiator', 'Ridley Scott', 2000, 'drama', 'others', 155, 'USA', 'Russell Crowe', 8.1, 552).
film('Dzien swira', 'Marek Koterski', 2002, 'drama', 'others', 123, 'Polska', 'Marek Kondrat', 7.8, 438).
film('Pianista', 'Roman Polanski', 2002, 'drama', 'others', 150, 'Polska', 'Adrien Brody', 8.3, 410).

/** COMEDY */
film('Seksmisja', 'Juliusz Machulski', 1984, 'comedy', 'others', 118, 'Polska', 'Jerzy Stuhr', 7.9, 420).
film('Forrest Gump', 'Robert Zemeckis', 1994, 'comedy', 'others', 144, 'USA', 'Tom Hanks', 8.6, 697).
film('Kac Vegas', 'Todd Phillips', 2009, 'comedy', 'others', 100, 'USA', 'Bradley Cooper', 7.3, 537).
film('Notykalni', 'Olivier Nakache', 2011, 'comedy', 'others', 112, 'Francja', 'Francois Cluzet', 8.7, 393).
film('Truman Show', 'Peter Weir', 1998, 'comedy', 'others', 103, 'USA', 'Jim Carrey', 7.4, 383).
film('Kiler', 'Juliusz Machulski', 1997, 'comedy', 'others', 104, 'Polska', 'Cezary Pazura', 7.7, 315).
film('Kevin sam w domu', 'Chris Columbus', 1990, 'comedy', 'others', 103, 'USA', 'Macaulay Culkin', 7.1, 297).
film('Mis', 'Stanislaw Bareja', 1980, 'comedy', 'others', 111, 'Polska', 'Stanislaw
Tym', 7.8, 261).
film('Diabel ubiera sie u Prady', 'David Frankel', 2006, 'comedy', 'others', 109,
'USA', 'Meryl Streep', 6.9, 227).
film('Jak rozpetalem druga wojne swiatowa', 'Tadeusz Chmielewski', 1969, 'comedy',
'others', 236, 'Polska', 'Marian Kociniak', 7.9, 195).

# Write the predicates related to actors and create some helper functions

# Write the code to design an expert system asking some questions related to the
mode, time available, gendre etc.
```

**Try:** Create an Expert System suggesting the medical support for the diagnosis of kidney diseases. Include the features like:

1. Forward chaining reasoning
2. History and questions management
3. Backtrack and facts **revocation.**
4. Management of uncertainty through the CF approach proposed for the first time in the MyCIN expert **system.**
5. Explanation and translation form of the technical glossary

# 12. Reinforcement Learning Techniques and Its Application

## 12.1 Working of RL Algorithm

In some situations, there is a lot of data available out there. However, algorithms aren't available to teach machines the logic to arrive at the desired output. This is where learning comes to the rescue. Reinforcement learning is the technology that, given the inputs and the desired outputs, will arrive at the logic or the algorithm to predict the output for an unforeseen or new input. The goal of this exercise is to develop the code to implement the RL algorithm.

**Input:** The state of the agent, environment, and the actions to be performed.

**Output:** Rewards accumulated by the agent in each step up to 20 steps which is the upper limit defined.

**Explanation:**

In a nutshell, RL is the branch of machine learning in which a machine learns from experience and takes proper decisions to maximize its reward or, in other words, to get the best reward possible. The machine is called the agent here. For every action it takes, it receives an award if it was the right action, failing which; it receives a punishment if it was the wrong action.

The best and the most common example of RL is how pet dogs are trained to get the stick and come back to their master! Every time the dog fails to get the stick or gets the wrong stick, it will not get its treat otherwise it will be rewarded with its delicious treats. The dog, quite obviously, aims to maximize the number of treats it gets because it loves enjoying its food! In this case, the dog is referred to as the agent.

**Agent:**

To check if the action taken by the agent was correct or wrong, logic will be involved. But, here, let's choose one of the rewards randomly using the random package. Let's begin by importing it:

```
import random
```

With the above understanding, let us define the environment class as follows:

**Hint:**

```
#create Environment class

class MyEnvironment:
def __init__(self):
self.remaining_steps=20

def get_observation(self):
return [1.0,2.0,1.0]

def get_actions(self):
return [-1,1]

def check_is_done(self):
return self.remaining_steps==0

def action(self,int):
if self.check_is_done():
raise Exception("Game over")
self.remaining_steps-=1

return random.random()
```

**myAgent:**

With this knowledge, the agent class can be defined as follows:

**Hint:**

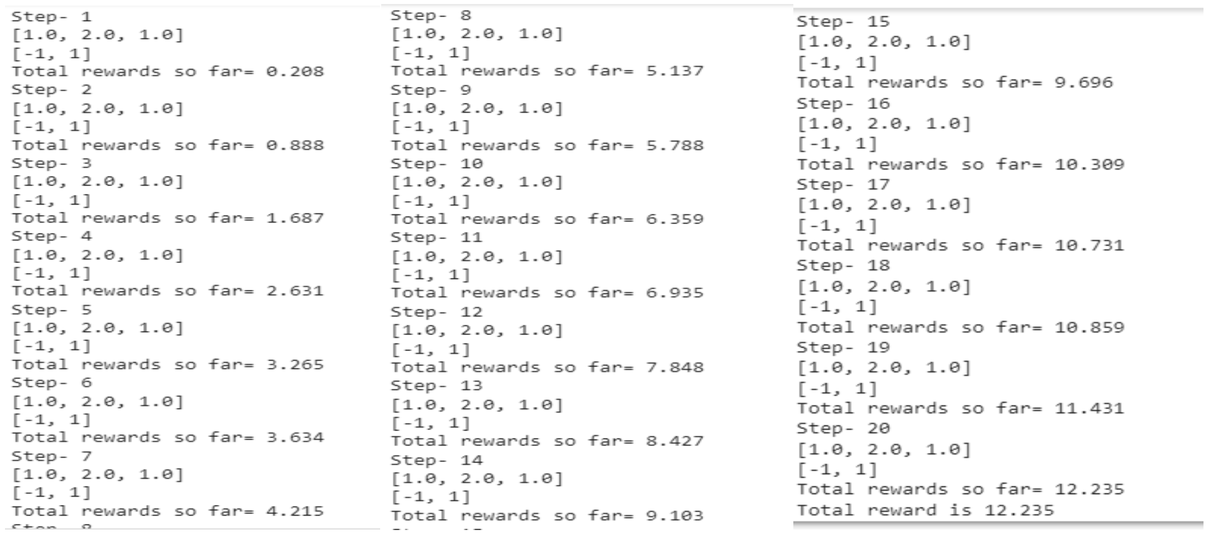
```
class myAgent:
   def __init__(self):
     self.total_rewards=0.0
   def step(self,ob:MyEnvironment):
     curr_obs=ob.get_observation()
     print(curr_obs)
     curr_action=ob.get_actions()
     print(curr_action)
     curr_reward=ob.action(random.choice(curr_action))
     self.total_rewards+=curr_reward
     print("Total rewards so far= %.3f "%self.total_rewards)
```

Finally, create objects of the above classes and execute as follows:

```
if __name__=='__main__':
        obj=MyEnvironment()
        agent=myAgent()
        step_number=0

while not obj.check_is_done():
        step_number+=1
        print("Step-",step_number)
        agent.step(obj)
print("Total reward is %.3f "%agent.total_rewards)
```

**Results**

Running the above code, we will get the rewards accumulated by the agent in each step up to 20 steps which is the upper limit defined by us. Here is a snapshot of what I got:



```
Step- 1
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 0.208
Step- 2
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 0.888
Step- 3
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 1.687
Step- 4
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 2.631
Step- 5
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 3.265
Step- 6
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 3.634
Step- 7
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 4.215
```

```
Step- 8
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 5.137
Step- 9
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 5.788
Step- 10
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 6.359
Step- 11
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 6.935
Step- 12
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 7.848
Step- 13
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 8.427
Step- 14
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 9.103
```

```
Step- 15
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 9.696
Step- 16
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 10.309
Step- 17
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 10.731
Step- 18
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 10.859
Step- 19
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 11.431
Step- 20
[1.0, 2.0, 1.0]
[-1, 1]
Total rewards so far= 12.235
Total reward is 12.235
```

**Try:** Implement the reinforcement learning technique with an OpenAI's gym, specially with MountainCar-v0 environment.

## 12.2 Q-Learning Technique

Most of the learning algorithms are trained based on the training dataset and show their efficiency by understanding the unseen data. Reinforcement Learning is a type of learning paradigm in which a learning algorithm is trained not on preset data but rather based on a feedback system. The goal of this exercise is to implement a basic Reinforcement Learning algorithm which is called the Q-Learning technique. In this exercise, we attempt to teach a bot to reach its destination using the Q-Learning technique.

**Input:** A state or an input state

**Output:** The most efficient path to reach its destination by a bot

**Explanation:**

Training an RL model is an iterative process because the agent keeps on learning from its experience. It keeps exploring the environment. Here, the agent faces a trade-off between experience and exploration: At a given time, should the agent explore the environment and decide its next action, or should it decide its next action based on its previous experience?

While training an RL model, firstly, scores are assigned to all the grids in the environment. The agent explores all the possible paths and learns from experience, again, aiming to maximize this total score it achieves by choosing among the grids.
The agent keeps exploring until it gets a negative reward. It stops at this point, realizing,”Oh! I was not supposed to go this way. I was wrong.”

Step 1: Importing the required libraries.

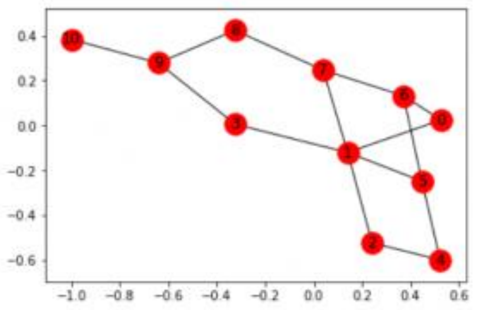
**Hint:**

```
import numpy as np
import pylab as pl
import networkx as nx
```

Step 2: Defining and visualizing the graph.

**Hint:**

```
edges = [(0, 1), (1, 5), (5, 6), (5, 4), (1, 2),
         (1, 3), (9, 10), (2, 4), (0, 6), (6, 7),
         (8, 9), (7, 8), (1, 7), (3, 9)]
goal = 10
G = nx.Graph()
G.add_edges_from(edges)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos)
pl.show()
```



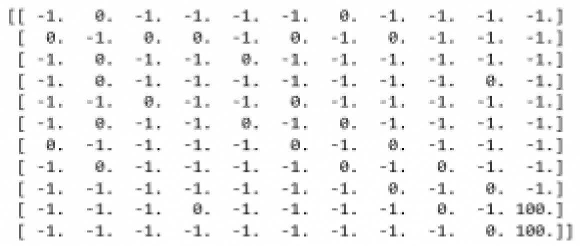
**Note:** The above graph may not look the same on reproduction of the code because the networkx library in python produces a random graph from the given edges.

Step 3: Defining the reward the system for the bot

**Hint:**

```
MATRIX_SIZE = 11
M = np.matrix(np.ones(shape =(MATRIX_SIZE, MATRIX_SIZE)))
M *= -1
```

```
for point in edges:
    print(point)
    if point[1] == goal:
        M[point] = 100
    else:
        M[point] = 0

    if point[0] == goal:
        M[point[::-1]] = 100
    else:
        M[point[::-1]]= 0
        # reverse of point
M[goal, goal]= 100
print(M)
# add goal point round trip
```



```
[[ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.   0.   0.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.]
 [ -1.  -1.   0.  -1.  -1.   0.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.  -1.]
 [  0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.  -1.]
 [ -1.   0.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.  -1.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.   0.  -1.   0.  -1.]
 [ -1.  -1.  -1.   0.  -1.  -1.  -1.  -1.   0.  -1. 100.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.   0. 100.]]
```

**Step 4: Defining some utility functions to be used in the training.**

**Hint:**

```
Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))
gamma = 0.75
# learning parameter
initial_state = 1
# Determines the available actions for a given state
def available_actions(state):
    current_state_row = M[state, ]
    available_action = np.where(current_state_row >= 0)[1]
    return available_action
 available_action = available_actions(initial_state)
# Chooses one of the available actions at random

# Updates the Q-Matrix according to the path chosen
 update(initial_state, action, gamma)
```

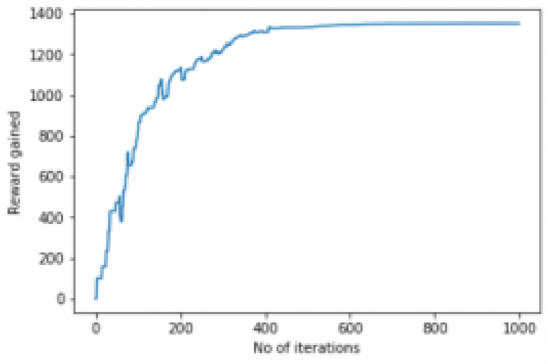
**Step 5: Training and evaluating the bot using the Q-Matrix**

**Hint:**

```
scores = []
```

```
for i in range(1000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_action = available_actions(current_state)
    action = sample_next_action(available_action)
    score = update(current_state, action, gamma)
    scores.append(score)
# print("Trained Q matrix:")
# print(Q / np.max(Q)*100)
# You can uncomment the above two lines to view the trained Q matrix
# Testing
current_state = 0
steps = [current_state]
while current_state != 10:
     next_step_index = np.where(Q[current_state, ] == np.max(Q[current_state,
]))[1]
    if next_step_index.shape[0] > 1:
        next_step_index = int(np.random.choice(next_step_index, size = 1))
    else:
        next_step_index = int(next_step_index)
    steps.append(next_step_index)
    current_state = next_step_index
print("Most efficient path:")
print(steps)
pl.plot(scores)
pl.xlabel('No of iterations')
pl.ylabel('Reward gained')
pl.show()
```

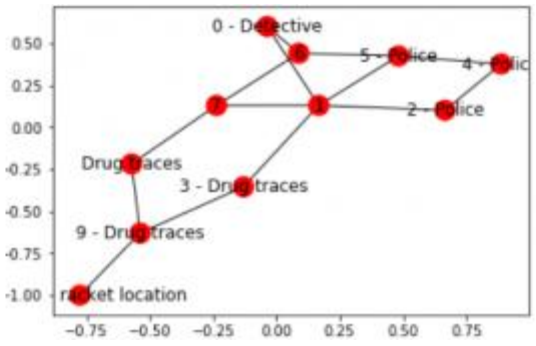
```
Most efficient path:
[0, 1, 3, 9, 10]
```



**Step 6: Defining and visualizing the new graph with the environmental clues.**

**Hint:**

```
# Defining the locations of the police and the drug traces
```



**Note:** The above graph may look a bit different from the previous graph but they, in fact, are the same graphs. This is due to the random placement of nodes by the networkx library.

**Step 7: Defining some utility functions for the training process**

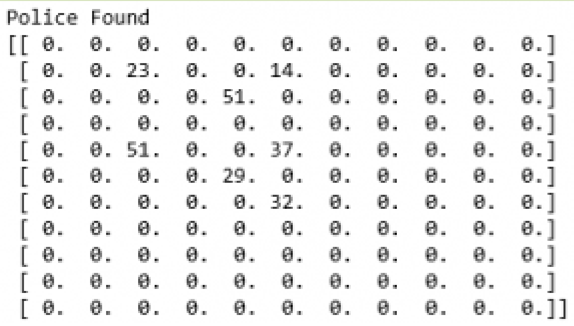
**Hint:**

```
Q = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))
env_police = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))
env_drugs = np.matrix(np.zeros([MATRIX_SIZE, MATRIX_SIZE]))
initial_state = 1
# Same as above
def available_actions(state):
    current_state_row = M[state, ]
    av_action = np.where(current_state_row >= 0)[1]
    return av_action
# Same as above
def sample_next_action(available_actions_range):
    next_action = int(np.random.choice(available_action, 1))
    return next_action
# Exploring the environment
def collect_environmental_data(action):
    found = []
    if action in police:
        found.append('p')
    if action in drug_traces:
        found.append('d')
    return (found)
available_action = available_actions(initial_state)
action = sample_next_action(available_action)
def update(current_state, action, gamma):
  max_index = np.where(Q[action, ] == np.max(Q[action, ]))[1]
  if max_index.shape[0] > 1:
      max_index = int(np.random.choice(max_index, size = 1))
  else:
      max_index = int(max_index)
  max_value = Q[action, max_index]
  Q[current_state, action] = M[current_state, action] + gamma * max_value
  environment = collect_environmental_data(action)
  if 'p' in environment:
    env_police[current_state, action] += 1
  if 'd' in environment:
    env_drugs[current_state, action] += 1
  if (np.max(Q) > 0):
    return(np.sum(Q / np.max(Q)*100))
  else:
    return (0)
# Same as above
update(initial_state, action, gamma)
def available_actions_with_env_help(state):
    current_state_row = M[state, ]
    av_action = np.where(current_state_row >= 0)[1]
# if there are multiple routes, dis-favor anything negative
    env_pos_row = env_matrix_snap[state, av_action]
    if (np.sum(env_pos_row < 0)):
# can we remove the negative directions from av_act?
        temp_av_action = av_action[np.array(env_pos_row)[0]>= 0]
        if len(temp_av_action) > 0:
            av_action = temp_av_action
    return av_action
# Determines the available actions according to the environment
```

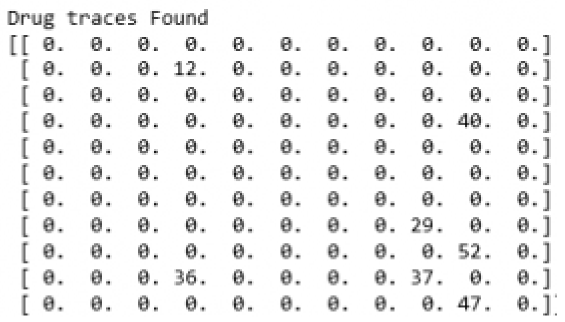
**Step 8: Visualizing the Environmental matrices**

**Hint:**

```
scores = []
for i in range(1000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_action = available_actions(current_state)
    action = sample_next_action(available_action)
    score = update(current_state, action, gamma)
# Print environmental matrices
```



```
Police Found
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 23.  0.  0. 14.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 51.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. 51.  0.  0. 37.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 29.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 32.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

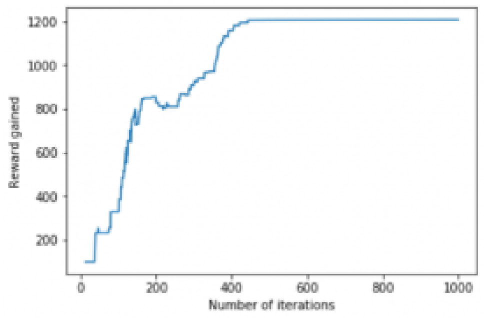


```
Drug traces Found
[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 12.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 40.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 29.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 52.  0.]
 [ 0.  0.  0. 36.  0.  0.  0.  0. 37.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 47.  0.]]
```

**Step 9: Training and evaluating the model**

**Hint:**

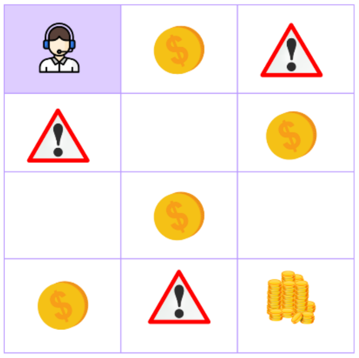
```
scores = []
for i in range(1000):
    current_state = np.random.randint(0, int(Q.shape[0]))
    available_action = available_actions_with_env_help(current_state)
    action = sample_next_action(available_action)
    score = update(current_state, action, gamma)
    scores.append(score)
pl.plot(scores)
pl.xlabel('Number of iterations')
pl.ylabel('Reward gained')
pl.show()
```



**Try:** Implement the Q Algorithm and Agent (Q Learning) and build Q Table by including all the possible discrete states.

## 12.3 Deep Q-Networks

Consider, that there is an AI agent present within a maze environment, and its goal is to find a reward. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.



**Input:** 1. Install the random package and choose the rewards

2. Environment and Action classes

**Output:** Perform exactly 10 steps and make the agent to again as many rewards as possible.

**Explanation:**

The field of reinforcement learning is made up of several algorithms that each take different approaches. The differences are mainly due to their strategies for exploring their environments. Some of the important Reinforcement learning algorithms are listed as follows.

- **Q-learning**
- **Deep Q-Networks**

```
#Importing the random package
import random
```

We use two classes, Environment and Agent in our model.

The environment class represents the agent's environment. The class must have member functions to get the current observation or state where the agent is, what are the points for reward and punishment, and keep track of how many more steps are left that the agent can take before the game is over. In this example, consider a game that the agent must finish in at most ten steps.

**Hint:**

```
#Creating the Environment class

class Environment:
  def init(self):
    self.steps_left=10
  def get_observation(self):
    return [1.0,2.0,1.0]
  def get_actions(self):
    return [-1,1]
  def check_is_done(self):
    return self.steps_left==0
  def action(self,int):
    if self.check_is_done():
      raise Exception("Game over")
    self.steps_left-=1
    return random.random()
```

The agent class is simpler compared to the environment class. The agent collects rewards given to it by its environment and makes an action. For this, we will need a data member and a member function.

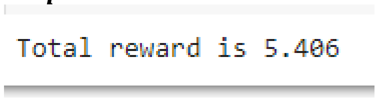
**Hint:**

```
#Creating the Agent class
class Agent:
  def init(self):
  self.total_rewards=0.0
  def step(self,ob:Environment):
    curr_obs=ob.get_observation()
    #print(curr_obs,end=" ")
    curr_action=ob.get_actions()
    #print(curr_action)
    curr_reward=ob.action(random.choice(curr_action))
    self.total_rewards+=curr_reward
    #print("Total rewards so far= %.3f "%self.total_rewards)
```

Until the game is not over, which is checked by the while loop, the agent takes an action by invoking the step function of the Agent class by passing obj which refers to the agent's environment. The reward here can be positive(in case of 1) or negative(in case of -1) and will be added to the total rewards of the agent.

```
if name=='main':
  obj=Environment()
  agent=Agent()
  step_number=0
  while not obj.check_is_done():
    step_number+=1

#print("Step-",step_number, end=" ")
    agent.step(obj)
  print("Total reward is %.3f "%agent.total_rewards)
```

***Output:***



```
Total reward is 5.406
```

On executing the code, we will get the rewards accumulated by the agent in each and every step up to 10 steps. The output differs with each time we play the game.

**Try:** Perform the sequence of actions that will eventually generate the maximum total reward using Markov Decision Process.

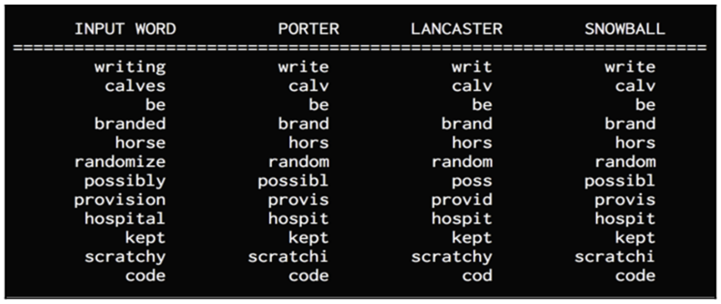
# 13. Exercises on Natural Language Processing

## 13.1 Converting words to their base forms using stemming

Working with text has a lot of variations included in it. We have to deal with different forms of the same word and enable the computer to understand that these different words have the same base form. For example, the word sing can appear in many forms such as sang, singer, singing, singer, and so on. We just saw a set of words with similar meanings. Humans can easily identify these base forms and derive context. Develop the python code to analyze the text and identify the base forms and derive the context.

**Input:** Bag of some input words like writing, calves, branded etc.

**Output:** Extract useful statistics to analyze the input text.



```
      INPUT WORD           PORTER        LANCASTER        SNOWBALL
======================================================================
         writing            write             writ           write
          calves             calv             calv            calv
              be               be               be              be
         branded            brand            brand           brand
           horse             hors             hors            hors
       randomize           random           random          random
        possibly          possibl             poss         possibl
       provision           provis           provid          provis
        hospital           hospit           hospit          hospit
            kept             kept             kept            kept
        scratchy         scratchi         scratchy        scratchi
            code             code              cod            code
```

**Hint**

```
# Create a new python file and import the following packages.

from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer

# Define some input words

# Create objects for Porter, Lancaster, and Snowball stemmers.

# Create a list of names for table display and format the output text accordingly.

# Iterate through the words and stem them using the three stemmers
```

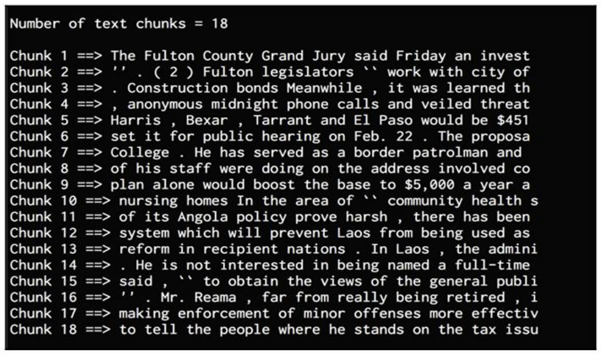
**Try**: Implement the code to understand the three stemming algorithms to achieve the same goal. Converting words to their base forms using lemmatization.

## 13.2 Dividing text data into chunks

Text data usually needs to be divided into pieces for further analysis. This process is known as chunking. This is used frequently in text analysis. The conditions that are used to divide the text into chunks can vary based on the problem at hand. This is not the same as tokenization where we also divide text into pieces. During chunking, we do not adhere to any constraints and the output chunks need to be meaningful. Develop the python code to divide the text into chunks to extract meaningful information.

Input: A large text document named brown.

Output: Divide the input text into chunks and display the output.



```
Number of text chunks = 18

Chunk 1 ==> The Fulton County Grand Jury said Friday an invest
Chunk 2 ==> '' . ( 2 ) Fulton legislators `` work with city of
Chunk 3 ==> . Construction bonds Meanwhile , it was learned th
Chunk 4 ==> , anonymous midnight phone calls and veiled threat
Chunk 5 ==> Harris , Bexar , Tarrant and El Paso would be $451
Chunk 6 ==> set it for public hearing on Feb. 22 . The proposa
Chunk 7 ==> College . He has served as a border patrolman and
Chunk 8 ==> of his staff were doing on the address involved co
Chunk 9 ==> plan alone would boost the base to $5,000 a year a
Chunk 10 ==> nursing homes In the area of `` community health s
Chunk 11 ==> of its Angola policy prove harsh , there has been
Chunk 12 ==> system which will prevent Laos from being used as
Chunk 13 ==> reform in recipient nations . In Laos , the admini
Chunk 14 ==> . He is not interested in being named a full-time
Chunk 15 ==> said , `` to obtain the views of the general publi
Chunk 16 ==> '' . Mr. Reama , far from really being retired , i
Chunk 17 ==> making enforcement of minor offenses more effectiv
Chunk 18 ==> to tell the people where he stands on the tax issu
```

### Hint

```
# Create a new python file and import the following packages.

import numpy as np
from nltk.corpus import brown

# Define a function to divide the input text into chunks.

# Iterate through the words and divide them into chunks using the input parameter.

# Define the main function and read the input data using the Brown corpus.

# Define the number of words in each chunk:

# Divide the input text into chunks and display the output:
```

**Try:** Implement the code to understand the three stemming algorithms to achieve the same goal. Converting words to their base forms using lemmatization.

## 13.3 Extracting the frequency of terms using a Bag of Words Model

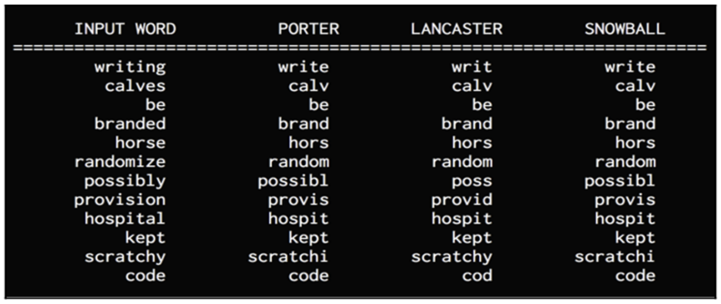
One of the main goals of text analysis is to convert text into numeric form so that we can use machine learning on it. Let's consider text documents that contain many millions of words. In order to analyze these documents, develop the python code to extract the text and convert it into a form of numeric representation.

**Input:** Consider the following sentences.

**Sentence 1:** The children are playing in the hall
**Sentence 2:** The hall has a lot of space
**Sentence 3:** Lots of children like playing in an open space

If you consider all the three sentences, we have the following nine unique words: the, children, are, playing, in, hall, has, a, lot, of, space, like, an, open.

**Output:** Extract useful statistics to analyze the input text.



```
      INPUT WORD           PORTER         LANCASTER         SNOWBALL
====================================================================
         writing            write              writ            write
          calves             calv              calv             calv
              be               be                be               be
         branded            brand             brand            brand
           horse             hors              hors             hors
       randomize           random            random           random
        possibly          possibl              poss          possibl
       provision           provis            provid           provis
        hospital           hospit            hospit           hospit
            kept             kept              kept             kept
        scratchy         scratchi          scratchy         scratchi
            code             code               cod             code
```

**Hint**

```
# Create a new python file and import the following packages.

import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import brown
from text_chunker import chunker

# Build a bag of words model in NLTK.

# Define the number of words in each chunk

# Divide the input text into chunks:

# Convert the chunks into dictionary items:

# Extract the document term matrix where we get the count of each word.

# Extract the vocabulary and display it and generate the names for display.

# Print the document term matrix.
```

**Try:** 1. Build a category predictor.
2. Construct a gender identifier.
3. Building a sentiment analyzer

# 14. Natural Language Processing

## 14.1 Sentiment Analysis

We have a dataset with tweets. Some of them are annotated with positive, negative, or neutral sentiment. Unfortunately annotating is time and cost intensive — we need to pay annotators for doing so and cross-check their answers for correctness. Therefore, most of the tweets are not labelled as it is relatively cheap and easy to download them, but not so cheap to annotate them. The goal of this exercise is to extract some useful features from these images which could help us in other tasks.

**Input**: piece of text
**Output**: The sentiment (positive, negative, or neutral) based on its polarity score.

**Explanation**:

defines a function analyze_sentiment() that takes a piece of text as input and returns the sentiment (positive, negative, or neutral) based on its polarity score.

```
pip install textblob
```

**Hint:**

```
from textblob import TextBlob
def analyze_sentiment(text):
    # Create a TextBlob object
    blob = TextBlob(text)

    # Get the sentiment polarity
    polarity = blob.sentiment.polarity

    # Classify sentiment
    if polarity > 0:
        return "Positive"
    elif polarity < 0:
        return "Negative"
    else:
        return "Neutral"

# Complete the code
```

**Try:** Explore what sentiment analysis encompasses and the various ways to implement it in Python.

## 14.2 Text Classification Using Naive Bayes

Text classification is the process of assigning predefined categories to textual data. In this experiment, you'll build a classifier (e.g., Spam vs. Ham) using the Naive Bayes algorithm, which works well with text due to its probabilistic nature and speed.

**Input:** SMS Spam dataset (e.g., from UCI repository or Kaggle), Format: label, text

**Output:**

Classification of messages as "spam" or "ham".

Accuracy, precision, recall, and confusion matrix.

**Hint:**

```
# import libraries
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

# Load dataset
df = pd.read_csv("spam.csv", encoding='latin-1')[['v1', 'v2']]
df.columns = ['label', 'text']

# Preprocessing

# Split & Train

# Predict and evaluate
```

**Try**: Modify the existing experiment to use TF-IDF vectorization instead of CountVectorizer. Then, compare the performance (accuracy, precision, recall) of both models using a bar plot. Which vectorization method works better for spam detection in your dataset, and why?

## 14.3 Named Entity Recognition using spaCy

NER is a sub-task of information extraction that seeks to locate and classify named entities in text into predefined categories such as person names, organizations, locations, etc.

**Input:** Any sample news article or paragraph.

**Output:**

List of named entities with labels.

Visual representation using spaCy's displacy.

**Hint:**

```
# import libraries
import spacy
from spacy import displacy

# Load English model
nlp = spacy.load("en_core_web_sm")

# Input text

# Process and visualize
```

**Try**: Modify the input to a longer news article or Wikipedia excerpt containing multiple types of entities (e.g., people, dates, locations, organizations). Extract and count the number of entities for each type (e.g., how many PERSON, ORG, GPE, etc.). Then, visualize the counts using a bar chart.

## 14.4 Text Summarization using Transformers (BART or T5)

Text summarization is the process of creating a shorter version of a long document while preserving its key information. Transformer-based models like BART and T5 are pre-trained for this task and can generate high-quality summaries using deep learning.

**Input:** A long news article or paragraph

**Output:** A concise summary of the input text

**Hint:**

```
# import libraries
from transformers import pipeline

# Load summarization pipeline
summarizer = pipeline("summarization", model="facebook/bart-large-cnn")

# Input Long text
text = """              """

# Generate summary
```

**Try**: Compare the results of BART and T5 for the same text. Adjust min_length and max_length to control the level of summarization.

## 15. Final Notes

The only way to learn programming is program, program, and program on challenging problems. The problems in this tutorial are certainly NOT challenging. There are tens of thousands of challenging problems available – used in training for various programming contests. Check out these sites:

1. Introduction to Artificial Intelligence with Python, Associated with Harvard University. CS50's Introduction to Artificial Intelligence with Python | Harvard University

2. NPTEL: An Introduction to Artificial Intelligence, https://nptel.ac.in/courses/106105077/

3. NPTEL: Artificial Intelligence Search Methods for Problem Solving, Artificial Intelligence Search Methods for Problem Solving - Course (nptel.ac.in).

4. IFACET (iitk.ac.in)

5. Introduction to Artificial Intelligence (AI) | Coursera in association with IBM.

6. http://www.ai.eecs.umich.edu

**Student must have any one of the following certifications:**

- Competitive Coding with AlphaCode Team - Competitive programming with AlphaCode (deepmind.com)

- IIIT Hyderabad Certification - Competitive programming with AlphaCode (deepmind.com)