**Lab File**

**Artificial Intelligence**

**[CSE401]**

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING



**Submitted To: Submitted By:**

Dr. S. K. Dubey Shaurya Guliani Associate Professor A2305219086 CSE Department, ASET B. Tech (CSE)

6CSE-2X

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY

AMITY UNIVERSITY UTTAR PRADESH

NOIDA-201301

**INDEX**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Experiment No.** | **Category**  **of Assignment** | **Code** | **Name of Experiment** | **Date of Allotment of experiment** | **Date of Evaluation** | **Max.**  **Marks** | **Marks obtained** | **Sign.**  **of Faculty** |
|  | **Mandatory Experiment** | **LR (0)** | **Write a program to implement A\* algorithm in python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Write a program to implement Single Player Game** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Write a program to implement Tic-Tac-Toe game problem** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Implement Brute force solution to the Knapsack problem in Python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Implement Graph colouring problem using python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Write a program to implement BFS for water jug problem using Python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Write a program to implement DFS using Python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Tokenization of word and Sentences with the help of NLTK package** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Design an XOR truth table using Python** |  |  | **1** |  |  |
|  | **Mandatory Experiment** |  | **Study of SCIKIT fuzzy** |  |  | **1** |  |  |

## **EXPERIMENT 1**

**AIM**- Write a program to implement A\* Algorithm in Python.

**Software Used:** Visual Studio Code

**Theory:**

A\* algorithm is a computer [algorithm](https://brilliant.org/wiki/algorithm/" \t "_blank" \o "algorithm) that is widely used in path finding and [graph](https://brilliant.org/wiki/graphs/) traversal. The algorithm efficiently plots a walkable path between multiple nodes, (or points) on the graph

On a map with many obstacles, pathfinding from points A to B can be difficult. A robot, for instance, without getting much other direction, will continue until it encounters an obstacle. However, the A\* algorithm introduces a [heuristic](https://brilliant.org/wiki/heuristic/" \t "_blank" \o "heuristic) into a regular graph-searching algorithm, essentially planning ahead at each step so a more optimal decision is made.

**Algorithm :**

1. Firstly, Place the starting node into OPEN and find its f (n) value.
2. Then remove the node from OPEN, having the smallest f (n) value. If it is a goal node, then stop and return to success
3. Else remove the node from OPEN, and find all its successors.
4. Find the f (n) value of all the successors, place them into OPEN, and place the removed node into CLOSE.
5. Goto Step-2.
6. Exit.

**Code:**

from collections import deque

adjac\_lis = {

    'A':[('B',6),('F',3)],

    'B':[('C',3),('D',2)],

    'C':[('D',1),('E',5)],

    'D':[('E',8)],

    'E':[('I',5),('J',5)],

    'F':[('G',1),('H',7)],

    'G':[('I',3)],

    'H':[('I',2)],

    'I':[('J',3)]

    }

class Graph:

    def \_\_init\_\_(self, adjac\_lis):

        self.adjac\_lis = adjac\_lis

    def get\_neighbors(self, v):

        return self.adjac\_lis[v]

    def h(self, n):

        H = {

            'A':10,

            'B':8,

            'C':5,

            'D':7,

            'E':3,

            'G':5,

            'F':6,

            'I':1,

            'H':3,

            'J':0

            }

        return H[n]

    def a\_star\_algorithm(self, start, stop):

        open\_lst = set([start])

        closed\_lst = set([])

        poo = {}

        poo[start] = 0

        par = {}

        par[start] = start

        while len(open\_lst) > 0:

            n = None

            for v in open\_lst:

                if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):

                    n = v

            if n == None:

                print('Path does not exist!')

                return None

            if n == stop:

                reconst\_path = []

                cost=0

                while par[n] != n:

                    reconst\_path.append(n)

                    n = par[n]

                reconst\_path.append(start)

                reconst\_path.reverse()

                cost=0

                for i in range(len(reconst\_path)-1):

                    x=reconst\_path[i]

                    y=reconst\_path[i+1]

                    for j in adjac\_lis[x]:

                        if j[0]==y:

                            cost=cost+j[1]

                print('Path found: {}'.format(reconst\_path))

                print("cost:",cost)

                return reconst\_path

            for (m, weight) in self.get\_neighbors(n):

                if m not in open\_lst and m not in closed\_lst:

                    open\_lst.add(m)

                    par[m] = n

                    poo[m] = poo[n] + weight

                else:

                    if poo[m] > poo[n] + weight:

                        poo[m] = poo[n] + weight

                        par[m] = n

                        if m in closed\_lst:

                            closed\_lst.remove(m)

                            open\_lst.add(m)

            open\_lst.remove(n)

            closed\_lst.add(n)

        print('Path does not exist!')

        return None

print("Name: Shaurya Guliani\nEnrollment No: A2305219086\n")

adjac\_lis = {

    'A':[('B',6),('F',3)],

    'B':[('C',3),('D',2)],

    'C':[('D',1),('E',5)],

    'D':[('E',8)],

    'E':[('I',5),('J',5)],

    'F':[('G',1),('H',7)],

    'G':[('I',3)],

    'H':[('I',2)],

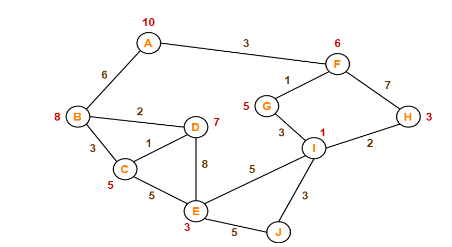
    'I':[('J',3)]

    }

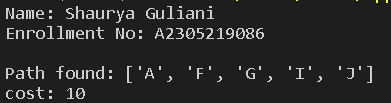
graph1 = Graph(adjac\_lis)

graph1.a\_star\_algorithm('A', 'J')

**Input:**



**Output:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 2**

**AIM**- Write a Python program to implement a single player game.

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

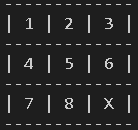
**8-puzzle game**

N-Puzzle or sliding puzzle is a popular puzzle that consists of N tiles where N can be 8, 15, 24 and so on. In our example N = 8. The puzzle is divided into sqrt(N+1) rows and sqrt(N+1) columns. E.g., 15-Puzzle will have 4 rows and 4 columns and an 8-Puzzle will have 3 rows and 3 columns. The puzzle consists of N tiles and one empty space where the tiles can be moved. Start and Goal configurations (also called state) of the puzzle are provided. The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal configuration.

**Initial state:** can be any state with a random arrangement of numbers from 1 to 8 and blank tile.



**Goal state:** the state in which the numbers are in sequence and the blank tile is at the last



**Operations:**

**r**-X moves towards right

**l**-X moves towards left

**u**-X moves upwards

**d**-X moves downwards

**CODE:**

x=[[1,2,3],[4,5,6],[7,8,'X']]

y=[[1,4,7],[2,5,8],[3,6,'X']]

game=[[1,2,3],['X',4,6],[7,5,8]]

print("NAME: Shaurya Guliani\nENROLLMENT NUMBER: A2305219086")

def display():

    print(f'-------------\n| {game[0][0]} | {game[0][1]} | {game[0][2]} |\n-------------\n| {game[1][0]} | {game[1][1]} | {game[1][2]} |\n-------------\n| {game[2][0]} | {game[2][1]} | {game[2][2]} |\n-------------\n')

while(game!=x and game !=y):

    display()

    i=input("enter where you want to move X")

    if(game[0][0]=='X'):

        if(i=='d'):

            game[0][0]=game[1][0]

            game[1][0]='X'

        if(i=='r'):

            game[0][0]=game[0][1]

            game[0][1]='X'

    elif(game[0][2]=='X'):

        if(i=='d'):

            game[0][2]=game[1][2]

            game[1][2]='X'

        if(i=='l'):

            game[0][2]=game[0][1]

            game[0][1]='X'

    elif(game[2][2]=='X'):

        if(i=='u'):

            game[2][2]=game[1][2]

            game[1][2]='X'

        if(i=='l'):

            game[2][2]=game[2][1]

            game[2][1]='X'

    elif(game[2][0]=='X'):

        if(i=='u'):

            game[2][0]=game[1][0]

            game[1][0]='X'

        if(i=='r'):

            game[2][0]=game[2][1]

            game[2][1]='X'

    elif(game[0][1]=='X'):

        if(i=='l'):

            game[0][1]=game[0][0]

            game[0][0]='X'

        if(i=='r'):

            game[0][1]=game[0][2]

            game[0][2]='X'

        if(i=='d'):

            game[0][1]=game[1][1]

            game[1][1]='X'

    elif(game[2][1]=='X'):

        if(i=='l'):

            game[2][1]=game[2][0]

            game[2][0]='X'

        if(i=='r'):

            game[2][1]=game[2][2]

            game[2][2]='X'

        if(i=='u'):

            game[2][1]=game[1][1]

            game[1][1]='X'

    elif(game[1][0]=='X'):

        if(i=='u'):

            game[1][0]=game[0][0]

            game[0][0]='X'

        if(i=='r'):

            game[1][0]=game[1][1]

            game[1][1]='X'

        if(i=='d'):

            game[1][0]=game[2][0]

            game[2][0]='X'

    elif(game[1][2]=='X'):

        if(i=='u'):

            game[1][2]=game[0][2]

            game[0][2]='X'

        if(i=='l'):

            game[1][2]=game[1][1]

            game[1][1]='X'

        if(i=='d'):

            game[1][2]=game[2][2]

            game[2][2]='X'

    elif(game[1][1]=='X'):

        if(i=='u'):

            game[1][1]=game[0][1]

            game[0][1]='X'

        if(i=='r'):

            game[1][1]=game[1][2]

            game[1][2]='X'

        if(i=='d'):

            game[1][1]=game[2][1]

            game[2][1]='X'

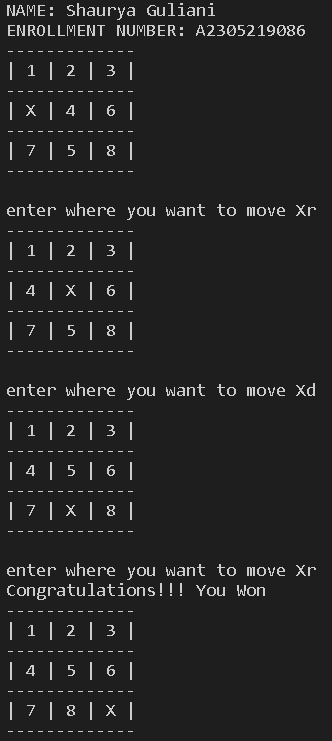
        if(i=='l'):

            game[1][1]=game[1][0]

            game[1][0]='X'

print("Congratulations!!! You Won")

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 3**

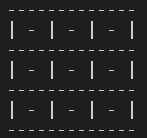
**AIM**- Write a program to implement the tic-tac-toe game problem.

**SOFTWARE USED:** Visual Studio Code

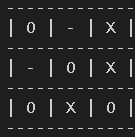
**THEORY:**

Tic-tac-toe who takes turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game. Players soon discover that the best play from both parties leads to a draw. Hence, tic-tac-toe is most often played by young children, who often have not yet discovered the optimal strategy. Because of the simplicity of tic-tac-toe, it is often used as a pedagogical tool for teaching the concepts of good sportsmanship and the branch of artificial intelligence that deals with the searching of game trees. It is straightforward to write a computer program to play tic-tac-toe perfectly or to enumerate the 765 essentially different positions (the state space complexity) or the 26,830 possible games up to rotations and reflections (the game tree complexity) on this space. The game can be generalized to an m,n,k-game in which two players alternate placing stones of their own colour on an m×n board, with the goal of getting k of their own colour in a row. Tic-tac-toe is the (3,3,3)-game. Tic-tac-toe is the game where n equals 3 and d equals 2. If played properly, the game will end in a draw, making tic-tac-toe a futile game. 

**Initial state:** an empty 3X3 Grid



**Goal state:** Any state in which either 3 ‘X’ or 3 ‘O’ are in a line.



**Operations:** Specify the row and column for inserting pattern (X or O) in an empty cell.

**CODE:**

print("NAME: Shaurya Guliani\nENROLLMENT NUMBER: A2305219086")

game=[['-','-','-'],['-','-','-'],['-','-','-']]

def display():

    for i in range(3):

        print("-------------")

        for j in range(3):

            print(f"| {game[i][j]} ",end="")

        print("|")

    print("-------------")

def checkO():

    if(game[0]==['O','O','O'] or game[1]==['O','O','O'] or game[2]==['O','O','O'] or game[0][0]=='O' and game[1][0]=='O' and game[2][0]=='O' or game[0][1]=='O' and game[1][1]=='O' and game[2][1]=='O' or game[0][2]=='O' and game[1][2]=='O' and game[2][2]=='O' or game[0][0]==game[1][1]==game[2][2]=='O' or game[2][0]==game[1][1]==game[0][2]=='O' ):

        return 1

    else:

        return 0

def checkX():

    if(game[0]==['X','X','X'] or game[1]==['X','X','X'] or game[2]==['X','X','X'] or game[0][0]=='X' and game[1][0]=='X' and game[2][0]=='X' or game[0][1]=='X' and game[1][1]=='X' and game[2][1]=='X' or game[0][2]=='X' and game[1][2]=='X' and game[2][2]=='X' or game[0][0]==game[1][1]==game[2][2]=='X' or game[2][0]==game[1][1]==game[0][2]=='X' ):

        return 1

    else:

        return 0

turn ='o'

t=0

display()

while((checkO()==0) and (checkX() == 0) and (t<9)):

    if(turn=='o'):

        print("player O enter row and column")

        x,y=input().split()

        x=int(x)

        y=int(y)

        game[x-1][y-1]='O'

        display()

        turn='x'

    elif(turn=='x'):

        print("player X enter row and column")

        x,y=input().split()

        x=int(x)

        y=int(y)

        game[x-1][y-1]='X'

        display()

        turn='o'

    t=t+1

if(checkO()==1):

    print("congratulations O you won!!!")

elif(checkX()==1):

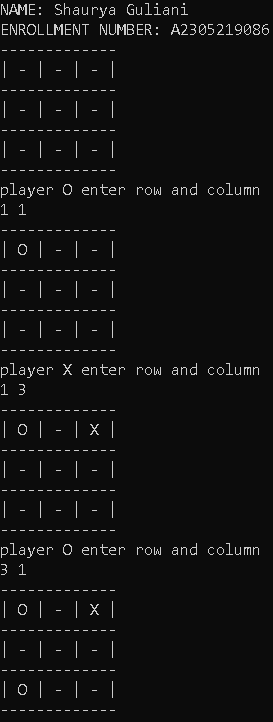
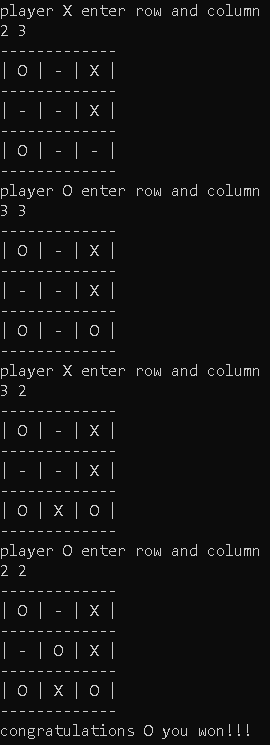
    print("congratulations X you won")

else:

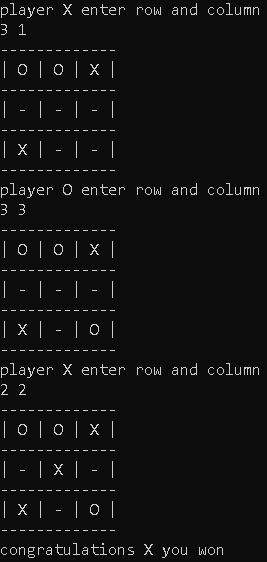
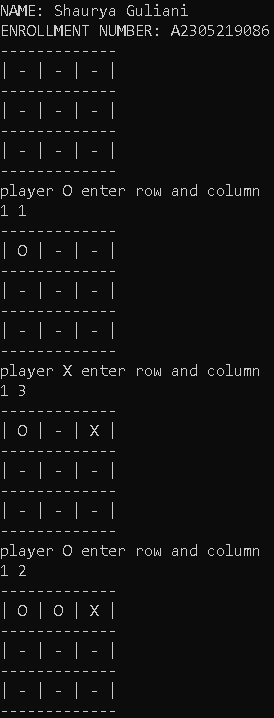
    print("Game Draw!!!!")

**OUTPUT:**

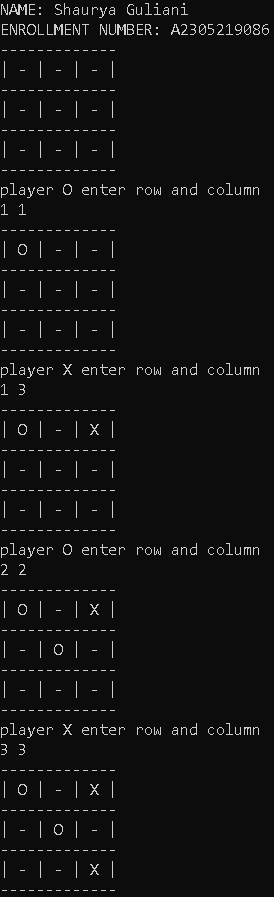
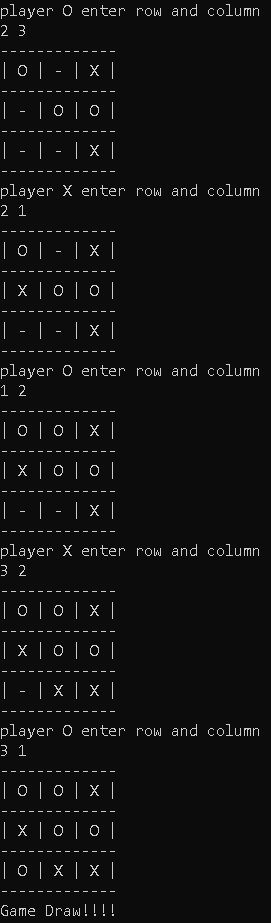
‘O’ wins:

‘X’ wins:



Game Draw:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 4**

**AIM**- Implement Brute Force solution to the Knapsack problem in Python.

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

Brute force is a straightforward approach to solving a problem, usually directly based on the problem’s statement and definitions of the concepts involved. If there are n items to choose from, then there will be 2 n possible combinations of items for the knapsack. An22 2 item is either chosen or not chosen. A bit string of 0’s and 1’s is generated which is of length n. If the I th symbol of a bit string is 0, then the i th item is not chosen and if it is 1, the i th item is chosen.

Therefore, the complexity of the Brute Force algorithm is O (n2 n ). Since the complexity of this algorithm grows exponentially, it can only be used for small instances of the KP. Otherwise, it does not require much programming effort in order to be implemented.

**CODE:**

import itertools

def findsubsets(s, n):

    return list(itertools.combinations(s, n))

print("\nNAME: SHAURYA GULIANI\nENROLLMENT NUMBER: A2305219086\n")

item=[0,1,2,3]

w=[2,5,10,5]

c=[20,30,50,10]

sub=[]

weight=[]

cost=[]

items=4

prof=0

capacity=12

subset=[]

for i in range(1,items+1):

    subset.append(findsubsets(item,i))

for i in subset:

    for j in i:

        sub.append(j)

        sumw=0

        sumc=0

        for k in j:

            sumw=sumw+w[k]

            sumc=sumc+c[k]

        weight.append(sumw)

        cost.append(sumc)

print("ITEM WEIGHTS: ",w,"\nITEM COSTS: ",c )

print("\nITEMS\t\tWEIGHT\t\tCOST\t\tVALIDITY ")

for i in range(len(sub)):

    if(len(sub[i])<3):

        print(sub[i],"\t\t",weight[i],"\t\t",cost[i],"\t\t",end="")

    else:

        print(sub[i],"\t",weight[i],"\t\t",cost[i],"\t\t",end="")

    if(weight[i]<=capacity):

        print('Valid')

    else:

        print('Invalid')

    if(weight[i]<=capacity and cost[i]>prof):

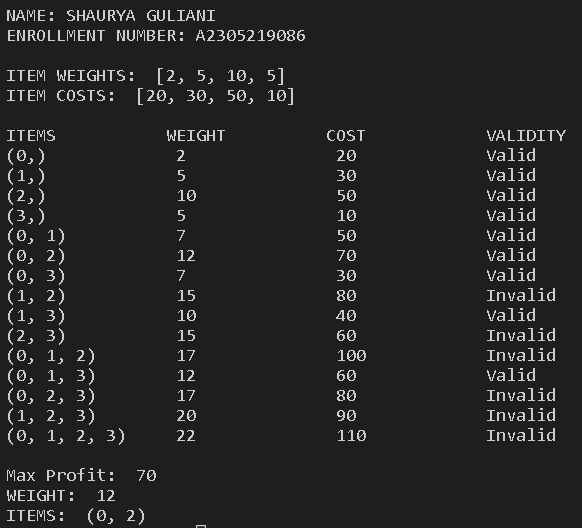
        prof=cost[i]

        profweight=weight[i]

        profsub=sub[i]

print("\nMax Profit: ",prof,"\nWEIGHT: ",profweight,"\nITEMS: ",profsub)

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 5**

**AIM**- Implement Graph Colouring problem using python.

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

In graph theory, graph colouring is a special case of graph labelling; it is an assignment of labels traditionally called "colours" to elements of a graph subject to certain constraints. In its simplest form, it is a way of colouring the vertices of a graph such that no two adjacent vertices are of the same colour; this is called a vertex colouring. Similarly, an edge colouring assigns a colour to each edge so that no two adjacent edges are of the same colour, and a face colouring of a planar graph assigns a colour to each face or region so that no two faces that share a boundary have the same colour. Vertex colouring is the starting point of graph colouring. Other colouring problems can be transformed into a vertex version. For example, an edge colouring of a graph is just a vertex colouring of its line graph, and a face colouring of a plane graph is just a vertex colouring of its dual.

**CODE:**

colors=['red','pink','blue','green','black','purple','yellow','orange','white']

neighbours={}

chrome={}

x=int(input("Enter the number of vertices"))

for i in range(x):

    print("enter the neighbours of",i,"in single line")

    n=input()

    arr=[]

    arr=n.split(" ")

    for j in range(len(arr)):

        arr[j]=int(arr[j])

        chrome[arr[j]]='NULL'

    neighbours[i]=arr

print("\nNEIGHBOURS:")

print(neighbours,"\n")

count=1

for j in range(x):

    c=0

    for k in neighbours[j]:

        if(chrome[k]==colors[c]):

            c=c+1

    chrome[j]=colors[c]

    if(c>=count):

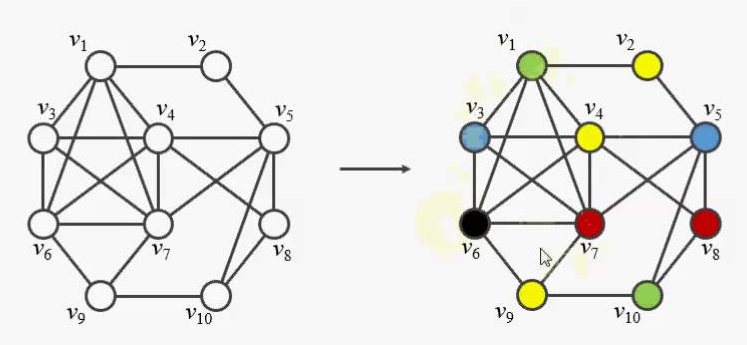
        count=count+1

print("\nCHROMATIC COLORING:")

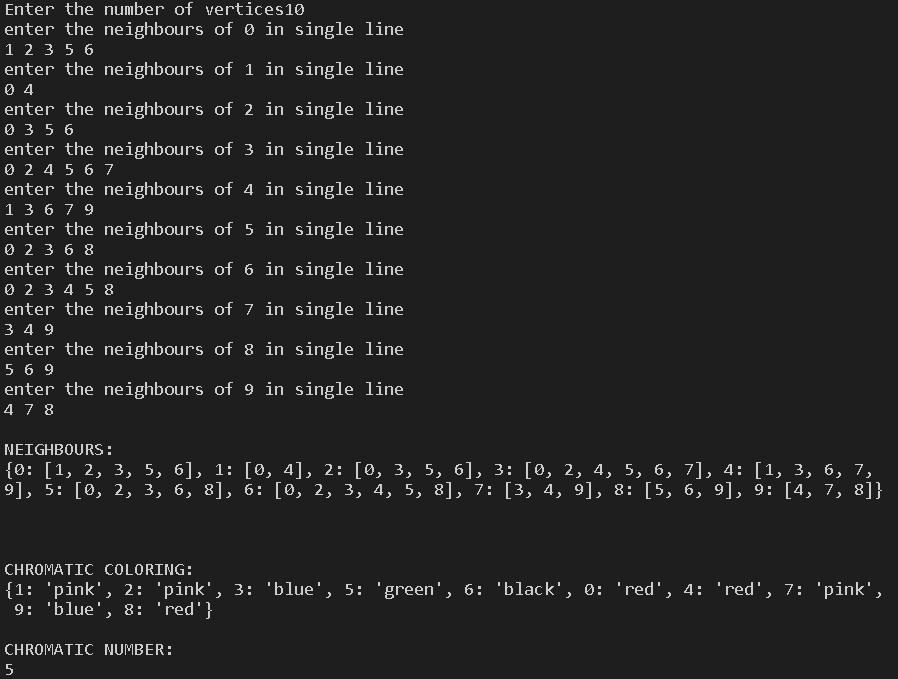
print(chrome)

print("\nCHROMATIC NUMBER:")

print(count)

**Input:**

**Output:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 6**

**AIM**- Write a Program to Implement BFS for water jug problem using python.

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

Problem Statement - A Water Jug Problem: You are given two jugs, a 4-gallon one and a 3-gallon one, a pump which has unlimited water which you can use to fill the jug, and the ground on which water may be poured. Neither jug has any measuring markings on it. How can you get exactly 2 gallons of water in the 4-gallon jug?

Breadth-first search is a graph traversal algorithm. It starts traversing the graph from the root node and explores all the neighbouring nodes. Then, it selects the nearest node and explores all the unexplored nodes. The algorithm continues the same process for each of the nearest nodes until it finds the goal.

**CODE:**

print('Shaurya Guliani\nA2305219086\n')

print ("Water Jug Problem Solution:")

x\_capacity = int(input("Enter capacity of jug 1"))

y\_capacity = int(input("Enter capacity of jug 2"))

end = int(input("Enter target volume:"))

tg=int(input("Enter target jug"))

def bfs(start, end, x\_capacity, y\_capacity,tg):

    path = []

    front = []

    front.append(start)

    visited = []

    visited.append(start)

    while(not (not front)):

        current = front.pop()

        x = current[0]

        y = current[1]

        path.append(current)

        if(y==end):

            if tg==1:

                print ("Found!")

                n=len(path)

                (a,b)=path[n-1]

                if(a!=end and b==end):

                    path.append([0,end])

                    visited.append([0,end])

                    path.append([end,0])

                    visited.append([end,0])

            print("jug-1  jug-2")

            for i in path:

                print(" ",i[0],"    ",i[1])

            return

        elif(x==end):

            if tg == 2:

                print ("Found!")

                n=len(path)

                (a,b)=path[n-1]

                if(a==end and b!=end):

                    path.append([end,0])

                    visited.append([end,0])

                    path.append([0,end])

                    visited.append([0,end])

            print("jug-1  jug-2")

            for i in path:

                print("  ",i[0],"    ",i[1])

            return

        # rule 1

        if current[0] < x\_capacity and ([x\_capacity, current[1]] not in visited):

            front.append([x\_capacity, current[1]])

            visited.append([x\_capacity, current[1]])

        # rule 2

        if current[1] < y\_capacity and ([current[0], y\_capacity] not in visited):

            front.append([current[0], y\_capacity])

            visited.append([current[0], y\_capacity])

        # rule 3

        if current[0] > 0 and ([0, current[1]] not in visited):

            front.append([0, current[1]])

            visited.append([0, current[1]])

        # rule 4

        if current[1] > 0 and ([current[0], 0] not in visited):

            front.append([current[0], 0])

            visited.append([current[0], 0])

        # rule 5

        if current[1] > 0 and current[0]+current[1]>=x\_capacity and (x\_capacity, current[1]-(x\_capacity-current[0]) not in visited):

            front.append([x\_capacity, current[1]-(x\_capacity-current[0])])

            visited.append([x\_capacity, current[1]-(x\_capacity-current[0])])

        # rule 6

        if current[0] > 0  and current[0]+current[1]>=y\_capacity and ([current[0]-(y\_capacity-current[1]),y\_capacity] not in visited):

            front.append([current[0]-(y\_capacity-current[1]),y\_capacity])

            visited.append([current[0]-(y\_capacity-current[1]),y\_capacity])

        # rule 7

        if current[1]>0 and current[0]+current[1]<=x\_capacity and ([current[0]+current[1],0] not in visited):

            front.append([current[0]+current[1],0])

            visited.append([current[0]+current[1],0])

        # rule 8

        if current[0]>0 and current[0]+current[1]<=y\_capacity and ([0,current[0]+current[1]] not in visited):

            front.append([0,current[0]+current[1]])

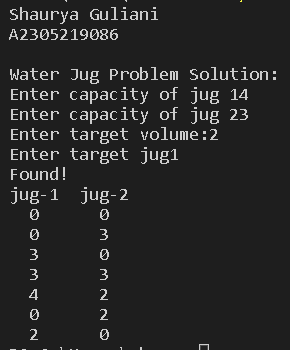
            visited.append([0,current[0]+current[1]])

    print ("Solution Not found")

start = [0, 0]

bfs(start, end, x\_capacity, y\_capacity,tg)

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

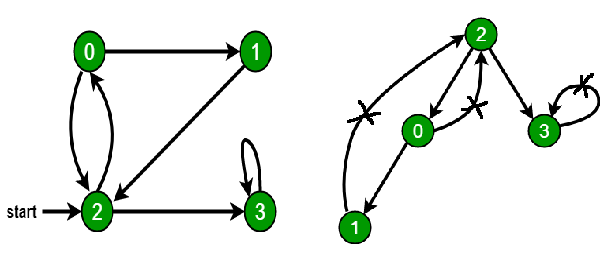
## **EXPERIMENT 7**

**AIM**- Write a program to implement DFS using Python

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

Depth First Traversal (or Search) for a graph is like Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. In the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don’t mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.



**CODE:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self):

        self.graph = defaultdict(list)

    def addEdge(self,u,v):

        self.graph[u].append(v)

    def DFSUtil(self,v,visited):

        visited[v]= True

        print (v)

        for i in self.graph[v]:

            if visited[i] == False:

                self.DFSUtil(i, visited)

    def DFS(self,v):

        visited = [False]\*(len(self.graph))

        self.DFSUtil(v,visited)

g = Graph()

g.addEdge(0, 1)

g.addEdge(0, 2)

g.addEdge(1, 2)

g.addEdge(2, 0)

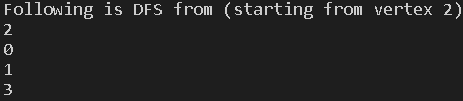
g.addEdge(2, 3)

g.addEdge(3, 3)

print ("Following is DFS from (starting from vertex 2)")

g.DFS(2)

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 8**

**AIM**- Tokenization of word and Sentences with the help of NLTK package

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](http://nltk.org/nltk_data/) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active [discussion forum](http://groups.google.com/group/nltk-users).

NLTK has been called “a wonderful tool for teaching, and working in, computational linguistics using Python,” and “an amazing library to play with natural language.”

[Natural Language Processing with Python](http://nltk.org/book) provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analysing linguistic structure, and more.

**CODE:**

from nltk.tokenize import word\_tokenize

newwords=[]

textw="John Cena is a professional wrestler, actor and television personality. Calling himself The Prototype, he captured the UPW title in 2000. In 2001, he signed a contract to work at Ohio Valley Wrestling. Cena claimed the OVW heavyweight title in February 2002, then made his WWE debut that June."

wordsw=word\_tokenize(textw)

for word in wordsw:

    if word.isalnum():

        newwords.append(word)

    else:

        continue

print("Words:", newwords)

print("No. of words: ",len(textw))

print("\n")

newwords=[]

textp="Mahatma Gandhi is known as the father of the nation. He was a great freedom fighter. His policy of non violence played an important role in India’s freedom. He inspired many civil rights movements in India and across the world. The title “Mahatma” was given to him in South Africa in 1914. Mahatma Gandhi was a lawyer by profession. He worked as a lawyer for Indians working in South Africa. He fought for their civil rights and against discrimination. He fought for the voting rights of Indians in Africa. Mahatma Gandhi returned to India in 1915 and became the most popular freedom fighter and political leader"

words=word\_tokenize(textp)

for word in words:

    if word.isalnum():

        continue

    else:

        newwords.append(word)

print("punctuations: ", newwords)

print("No. of Punctuations: ",len(newwords))

print("\n")

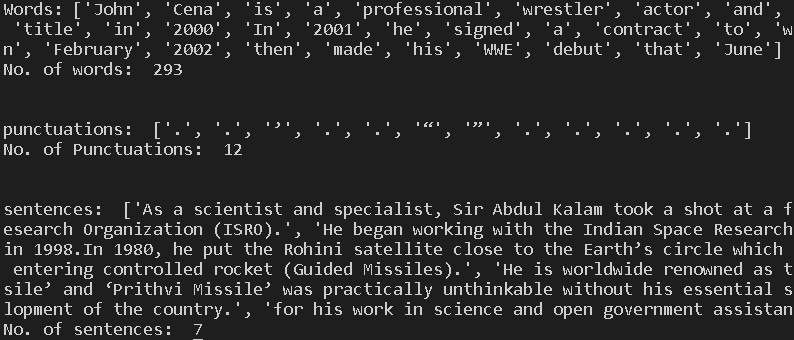
from nltk.tokenize import sent\_tokenize

texts="As a scientist and specialist, Sir Abdul Kalam took a shot at a few significant projects of the Defense Research and Development Organization (DRDO) and Indian Space Research Organization (ISRO). He began working with the Indian Space Research Organization (ISRO) since 1972. He was additionally a significant part of Pokhran II Nuclear Test in 1998.In 1980, he put the Rohini satellite close to the Earth’s circle which caused India to be a member from the International Club. He had structured the indigenous objective entering controlled rocket (Guided Missiles). He is worldwide renowned as the ‘Rocket Man of India’ for his extraordinary work in space science.The fruitful testing of ‘Agni Missile’ and ‘Prithvi Missile’ was practically unthinkable without his essential support. Kalam Ji consistently showed Indian youth to build up their ability and use it for the development of the country. for his work in science and open government assistance, he got the Bharat Ratna and different esteemed honours of India."

print("sentences: ",sent\_tokenize(texts))

print("No. of sentences: ",len(sent\_tokenize(texts)))

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

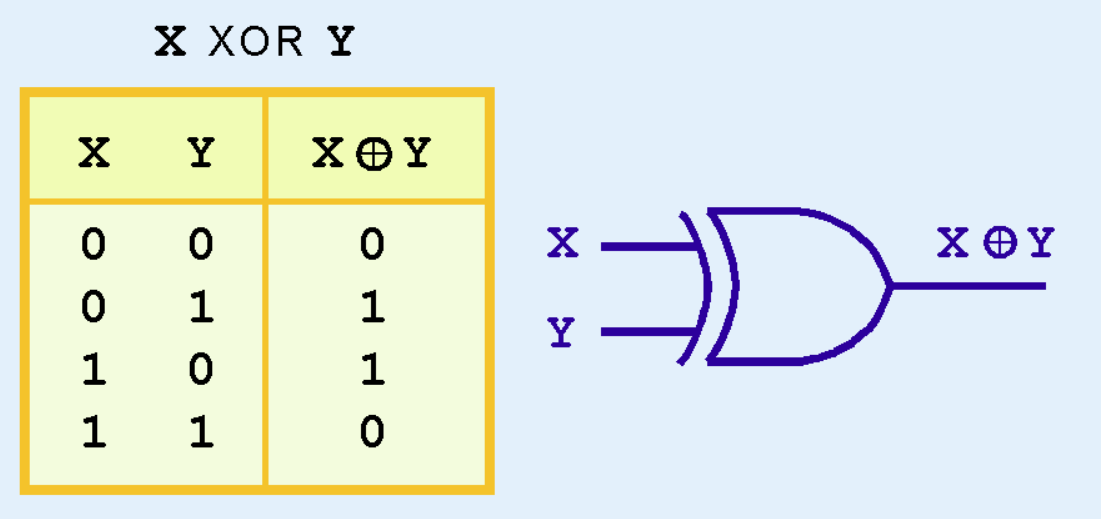
## **EXPERIMENT 9**

**AIM**- Design an XOR truth table using Python

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

A Logic gate is an elementary building block of any digital circuits. It takes one or two inputs and produces output based on those inputs. Outputs may be high (1) or low (0). Logic gates are implemented using diodes or transistors. The XOR gate gives an output of 1 if either both inputs are different, it gives 0 if they are same.



**CODE:**

print("X xor Y:")

print("------------------------")

print("|  X  |  Y  |  X xor Y |")

print("------------------------")

for X in [0,1]:

    for Y in [0,1]:

        print("| ",X," | ",Y," |    ",X^Y,"   |")

        print("------------------------")

print("\n X xor Y xor Z:")

print("---------------------------------")

print("|  X  |  Y  |  Z  |X xor Y xor Z|")

print("---------------------------------")

for X in [0,1]:

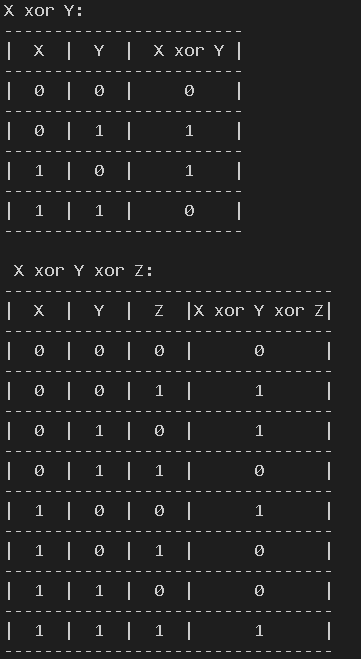
    for Y in [0,1]:

        for Z in [0,1]:

            print("| ",X," | ",Y," | ",Z," |     ",X^Y^Z,"     |")

            print("---------------------------------")

**OUTPUT:**



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

## **EXPERIMENT 10**

**AIM**- Study of SCIKIT fuzzy

**SOFTWARE USED:** Visual Studio Code

**THEORY:**

Scikit-fuzzy is also known as SKFuzzy. It is a Fuzzy logic toolbox for Python.This package implements many useful tools for projects involving fuzzy logic, also known as grey logic. Scikit-Fuzzy is a collection of fuzzy logic algorithms intended for use in the [SciPy](http://scipy.org/) Stack, written in the Python computing language. Scikit-fuzzy is a fuzzy logic Python package that works with numpy arrays.

|  |  |
| --- | --- |
| [kfuzzy.addval](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.addval)(interval1, interval2) | Add intervals interval1 and interval2. |
| [skfuzzy.arglcut](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.arglcut)(ms, lambdacut) | Determines the subset of indices mi of the elements in an N-point resultant fuzzy membership sequence ms that have a grade of membership >= lambdacut. |
| [skfuzzy.cartadd](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.cartadd)(x, y) | Cartesian addition of fuzzy membership vectors using the algebraic method. |
| [skfuzzy.cartprod](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.cartprod)(x, y) | Cartesian product of two fuzzy membership vectors. |
| [skfuzzy.centroid](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.centroid)(x, mfx) | Defuzzification using centroid (center of gravity) method. |
| [skfuzzy.classic\_relation](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.classic_relation)(a, b) | Determine the classic relation matrix, R, between two fuzzy sets. |
| [skfuzzy.cmeans](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.cmeans)(data, c, m, error, maxiter[, ...]) | Fuzzy c-means clustering algorithm [1]. |
| [skfuzzy.cmeans\_predict](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.cmeans_predict)(test\_data, ...[, ...]) | Prediction of new data in given a trained fuzzy c-means framework [1]. |
| [skfuzzy.continuous\_to\_discrete](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.continuous_to_discrete)(a, b, ...) | Converts a continuous-time system to its equivalent discrete-time version. |
| [skfuzzy.contrast](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.contrast)(arr[, amount, split, normalize]) | General contrast booster or diffuser of normalized array-like data. |
| [skfuzzy.dcentroid](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dcentroid)(x, mfx, x0) | Defuzzification using a differential centroidal method about x0. |
| [skfuzzy.defocus\_local\_means](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.defocus_local_means)(im) | Defocusing non-normalized image im using local arithmatic mean. |
| [skfuzzy.defuzz](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.defuzz)(x, mfx, mode) | Defuzzification of a membership function, returning a defuzzified value of the function at x, using various defuzzification methods. |
| [skfuzzy.divval](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.divval)(interval1, interval2) | Divide interval2 into interval1, by inversion and multiplication. |
| [skfuzzy.dsigmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dsigmf)(x, b1, c1, b2, c2) | Difference of two fuzzy sigmoid membership functions. |
| [skfuzzy.dsw\_add](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dsw_add)(x, mfx, y, mfy, n) | Add two fuzzy variables together using the restricted DSW method [1]. |
| [skfuzzy.dsw\_div](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dsw_div)(x, mfx, y, mfy, n) | Divide one fuzzy variable by another using the restricted DSW method [1]. |
| [skfuzzy.dsw\_mult](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dsw_mult)(x, mfx, y, mfy, n) | Multiply two fuzzy variables using the restricted DSW method [1]. |
| [skfuzzy.dsw\_sub](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.dsw_sub)(x, mfx, y, mfy, n) | Subtract a fuzzy variable from another by the restricted DSW method [1]. |
| [skfuzzy.fire1d](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fire1d)(x[, l1, l2]) | 1-D filtering using Fuzzy Inference Ruled by Else-action (FIRE) [1]. |
| [skfuzzy.fire2d](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fire2d)(im[, l1, l2, fuzzyresolution]) | 2-D filtering using Fuzzy Inference Ruled by Else-action (FIRE) [1]. |
| [skfuzzy.fuzzy\_add](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_add)(x, a, y, b) | Add fuzzy set a to fuzzy set b. |
| [skfuzzy.fuzzy\_and](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_and)(x, mfx, y, mfy) | Fuzzy AND operator, a.k.a. |
| [skfuzzy.fuzzy\_compare](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_compare)(q) | Determine the comparison matrix, c, based on the fuzzy pairwise comparison matrix, q, using Shimura’s special relativity formula. |
| [skfuzzy.fuzzy\_div](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_div)(x, a, y, b) | Divide fuzzy set b into fuzzy set a. |
| [skfuzzy.fuzzy\_min](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_min)(x, a, y, b) | Find minimum between fuzzy set a fuzzy set b. |
| [skfuzzy.fuzzy\_mult](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_mult)(x, a, y, b) | Multiplies fuzzy set a and fuzzy set b. |
| [skfuzzy.fuzzy\_not](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_not)(mfx) | Fuzzy NOT operator, a.k.a. |
| [skfuzzy.fuzzy\_or](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_or)(x, mfx, y, mfy) | Fuzzy OR operator, a.k.a. |
| [skfuzzy.fuzzy\_similarity](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_similarity)(ai, b[, mode]) | The fuzzy similarity between set ai and observation set b. |
| [skfuzzy.fuzzy\_sub](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.fuzzy_sub)(x, a, y, b) | Subtract fuzzy set b from fuzzy set a. |
| [skfuzzy.gauss2mf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.gauss2mf)(x, mean1, sigma1, mean2, sigma2) | Gaussian fuzzy membership function of two combined Gaussians. |
| [skfuzzy.gaussmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.gaussmf)(x, mean, sigma) | Gaussian fuzzy membership function. |
| [skfuzzy.gbellmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.gbellmf)(x, a, b, c) | Generalized Bell function fuzzy membership generator. |
| [skfuzzy.inner\_product](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.inner_product)(a, b) | Inner product (dot product) of two fuzzy sets. |
| [skfuzzy.interp10](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.interp10)(x) | Utility function which conducts linear interpolation of any rank-1 array. |
| [skfuzzy.interp\_membership](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.interp_membership)(x, xmf, xx) | Find the degree of membership u(xx) for a given value of x = xx. |
| [skfuzzy.interp\_universe](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.interp_universe)(x, xmf, y) | Find interpolated universe value(s) for a given fuzzy membership value. |
| [skfuzzy.lambda\_cut](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.lambda_cut)(ms, lcut) | The crisp (binary) lambda-cut set of the membership sequence ms with membership >= lcut. |
| [skfuzzy.lambda\_cut\_boundaries](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.lambda_cut_boundaries)(x, mfx, lambdacut) | Find exact boundaries where mfx crosses lambdacut using interpolation. |
| [skfuzzy.lambda\_cut\_series](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.lambda_cut_series)(x, mfx, n) | Determine a series of lambda-cuts in a sweep from 0+ to 1.0 in n steps. |
| [skfuzzy.maxmin\_composition](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.maxmin_composition)(s, r) | The max-min composition t of two fuzzy relation matrices. |
| [skfuzzy.maxprod\_composition](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.maxprod_composition)(s, r) | The max-product composition t of two fuzzy relation matrices. |
| [skfuzzy.modus\_ponens](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.modus_ponens)(a, b, ap[, c]) | Generalized *modus ponens* deduction to make approximate reasoning in a rules-base system. |
| [skfuzzy.multval](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.multval)(interval1, interval2) | Multiply intervals interval1 and interval2. |
| [skfuzzy.nmse](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.nmse)(known, degraded) | Computes the percent normalized mean square error (NMSE %) between known and degraded arrays. |
| [skfuzzy.outer\_product](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.outer_product)(a, b) | Outer product of two fuzzy sets. |
| [skfuzzy.pad](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.pad)(array, pad\_width[, mode]) | Pads an array. |
| [skfuzzy.partial\_dmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.partial_dmf)(x, mf\_name, ...) | Calculate the *partial derivative* of a specified membership function. |
| [skfuzzy.piecemf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.piecemf)(x, abc) | Piecewise linear membership function (particularly used in FIRE filters). |
| [skfuzzy.pimf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.pimf)(x, a, b, c, d) | Pi-function fuzzy membership generator. |
| [skfuzzy.psigmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.psigmf)(x, b1, c1, b2, c2) | Product of two sigmoid membership functions. |
| [skfuzzy.relation\_min](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.relation_min)(a, b) | Determine fuzzy relation matrix R using Mamdani implication for the fuzzy antecedent a and consequent b inputs. |
| [skfuzzy.relation\_product](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.relation_product)(a, b) | Determine the fuzzy relation matrix, R, using product implication for the fuzzy antecedent a and the fuzzy consequent b. |
| [skfuzzy.scaleval](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.scaleval)(q, interval) | Multiply scalar q with interval interval. |
| [skfuzzy.sigmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.sigmf)(x, b, c) | The basic sigmoid membership function generator. |
| [skfuzzy.sigmoid](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.sigmoid)(x, power[, split]) | Intensify grayscale values in an array using a sigmoid function. |
| [skfuzzy.smf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.smf)(x, a, b) | S-function fuzzy membership generator. |
| [skfuzzy.subval](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.subval)(interval1, interval2) | Subtract interval interval2 from interval interval1. |
| [skfuzzy.test](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.test)([doctest, verbose]) | Run all unit tests. |
| [skfuzzy.trapmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.trapmf)(x, abcd) | Trapezoidal membership function generator. |
| [skfuzzy.trimf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.trimf)(x, abc) | Triangular membership function generator. |
| [skfuzzy.view\_as\_blocks](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.view_as_blocks)(arr\_in, block\_shape) | Block view of the input n-dimensional array (using re-striding). |
| [skfuzzy.view\_as\_windows](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.view_as_windows)(arr\_in, window\_shape) | Rolling window view of the input n-dimensional array. |
| [skfuzzy.zmf](https://pythonhosted.org/scikit-fuzzy/api/skfuzzy.html#skfuzzy.zmf)(x, a, b) | Z-function fuzzy membership generator. |

**ANALYSIS:**

A FuzzySetrequires the following parameters so that it can be initiated:

* **name —**the name of the set
* **minimum value —**the minimum value of the set
* **maximum value —**the maximum value of the set
* **resolution —**the number of steps between the minimum and maximum value

It is, therefore, possible to represent a fuzzy set by using two **numpy**arrays; one that will hold the domain values and one that will hold the degree-of-membership values. Initially, all degree-of-membership values will be all set to zero. It can be argued that if the minimum and maximum values are available together with the resolution of the set, the domain numpy array is not required as the respective values can be calculated. While this is perfectly true, a domain array was preferred in this example project so that the code is more readable and simple.

import numpy as np

def create\_triangular(cls, name, domain\_min, domain\_max, res, a, b, c):

  t1fs = cls(name, domain\_min, domain\_max, res)

  a = t1fs.\_adjust\_domain\_val(a)

  b = t1fs.\_adjust\_domain\_val(b)

  c = t1fs.\_adjust\_domain\_val(c)

  t1fs.\_dom = np.round(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (c-t1fs.\_domain)/(c-b)), 0), t1fs.\_precision)

In the context of a fuzzy variable, all the sets will have the same minimum, maximum and resolution values. As we are dealing with a discretized domain, it will be necessary to adjust any value used to set or retrieve the degree-of-membership to the closest value in the domain array.

def \_adjust\_domain\_val(self, x\_val):

  return self.\_domain[np.abs(self.\_domain-x\_val).argmin()]

Since the sets are based on numpy arrays, the equation above can be translated directly to code, as can be seen below. Sets having different shapes can be constructed using a similar method.

def create\_triangular(cls, name, domain\_min, domain\_max, res, a, b, c):

  t1fs = cls(name, domain\_min, domain\_max, res)

  a = t1fs.\_adjust\_domain\_val(a)

  b = t1fs.\_adjust\_domain\_val(b)

  c = t1fs.\_adjust\_domain\_val(c)

  t1fs.\_dom = np.round(np.maximum(np.minimum((t1fs.\_domain-a)/(b-a), (c-t1fs.\_domain)/(c-b)), 0), t1fs.\_precision)

The FuzzySetclass also contains union, intersection and negation operators that are necessary so that inferencing can take place. All operator methods return a new fuzzy set with the result of the operation that took place.

def union(self, f\_set):

        result = FuzzySet(f'({self.\_name}) union ({f\_set.\_name})', self.\_domain\_min, self.\_domain\_max, self.\_res)

        result.\_dom = np.maximum(self.\_dom, f\_set.\_dom)

        return result

Finally, we implemented the ability to obtain a crisp result from a fuzzy set using the centre-of-gravity method that is referred to in some detail in the previous article. It is important to mention that there is a large number of defuzzification methods are available in the literature. Still, as the centre-of-gravity method is overwhelmingly popular, it is used in this implementation.

def cog\_defuzzify(self):

  num = np.sum(np.multiply(self.\_dom, self.\_domain))

  den = np.sum(self.\_dom)

  return num/den

For input variables, fuzzification is carried out by retrieving the degree-of-membership of all the sets in the variable for a given domain value. The degree-of-membership is stored in the set as it will be required by the rules when they are evaluated.

def fuzzify(self, val):

    for set\_name, f\_set in self.\_sets.items():

        f\_set.last\_dom\_value = f\_set[val]

Therefore, output variables will require an additional FuzzySet attribute that will hold the output distribution for that variable, where the contribution that was resulting from each rule and added using the set union operator. The defuzzification result can then be obtained by calling the centre-of-gravity method for output distribution set.

class FuzzyOutputVariable(FuzzyVariable):

    def \_\_init\_\_(self, name, min\_val, max\_val, res):

        super().\_\_init\_\_(name, min\_val, max\_val, res)

        self.\_output\_distribution = FuzzySet(name, min\_val, max\_val, res)

    def add\_rule\_contribution(self, rule\_consequence):

        self.\_output\_distribution = self.\_output\_distribution.union(rule\_consequence)

    def get\_crisp\_output(self):

        return self.\_output\_distribution.cog\_defuzzify()

The rule will combine the degree-of-membership values from the various antecedent clauses using the min operator, obtaining the rule activation that is then used in conjunction with the consequent clauses to obtain the contribution of the rule to the output variables. This operation is a two-step process:

* The activation value is combined with the consequent FuzzySet using the **min** operator, that will act as a threshold to the degree-of-membership values of the **FuzzySet**.
* The resultant FuzzySet is combined with the FuzzySets obtained from the other rules using the **union** operator, obtaining the output distribution for that variable.

def evaluate\_antecedent(self):

    return self.\_set.last\_dom\_value

def evaluate\_consequent(self, activation):

    self.\_variable.add\_rule\_contribution(self.\_set.min\_scalar(activation))

The FuzzyRule class will, therefore, require two attributes:

* a list containing the antecedent clauses and
* a list containing the consequent clauses

During the execution of the FuzzyRule, the procedure explained above is carried out. The FuzzyRule coordinates all the tasks by utilizing all the various FuzzyClause*s* as appropriate.

def evaluate(self):

    rule\_activation = 1

    for ante\_clause in self.\_antecedent:

        rule\_activation = min(ante\_clause.evaluate\_antecedent(), rule\_activation)

    for consequent\_clause in self.\_consequent:

        consequent\_clause.evaluate\_consequent(rule\_activation)

A more user-friendly method is to provide the rule as a string and then parse that string to create the rule, but this seemed an unnecessary overhead for a demonstration application.

def add\_rule(self, antecedent\_clauses, consequent\_clauses):

        new\_rule = FuzzyRule()

        for var\_name, set\_name in antecedent\_clauses.items():

            var = self.get\_input\_variable(var\_name)

            f\_set = var.get\_set(set\_name)

            new\_rule.add\_antecedent\_clause(var, f\_set)

        for var\_name, set\_name in consequent\_clauses.items():

            var = self.get\_output\_variable(var\_name)

            f\_set = var.get\_set(set\_name)

            new\_rule.add\_consequent\_clause(var, f\_set)

        self.\_rules.append(new\_rule)

The execution of the inference process can be achieved with a few lines of code given this structure, where the following steps are carried out;

1. The output distribution sets of all the output variables are cleared.
2. The input values to the system are passed to the corresponding input variables so that each set in the variable can determine its degree-of-membership for that input value.
3. Execution of the Fuzzy Rules takes place, meaning that the output distribution sets of all the output variables will now contain the union of the contributions from each rule.
4. The output distribution sets are defuzzified using a centre-of-gravity defuzzifier to obtain the crisp result.

self.\_clear\_output\_distributions()

for input\_name, input\_value in input\_values.items():

    self.\_input\_variables[input\_name].fuzzify(input\_value)

for rule in self.\_rules:

    rule.evaluate()

output = {}

for output\_var\_name, output\_var in self.\_output\_variables.items():

    output[output\_var\_name] = output\_var.get\_crisp\_output()

return output

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP) | | | | |
| Programme | B. Tech CSE | | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | | Semester | 6 |
| Student Name | Shaurya Guliani | | Enrolment No. | A2305219086 |
| Marking Criteria | | | | |
| Criteria | Total Marks | Marks Obtained | | Comments |
| Concept (A) | 2 |  | |  |
| Implementation (B) | 2 |  | |  |
| Performance (C) | 2 |  | |  |
| Total | 6 |  | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Internal Assessment (Viva Component) Sheet for Lab Experiment Department of Computer Science & Engineering Amity University, Noida (UP)** | | | |
| Programme | B. Tech CSE | Course Name | Artificial Intelligence |
| Course Code | [CSE401] | Semester | 6 |
| Student Name | Shaurya Guliani | Enrollment No. | A2305219086 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Clarity of the Subject (H) | 4 |  |  |
| Quality of theoretical Discussion (I) | 6 |  |  |
| Total | 10 |  |  |