SUPHIYA
19131A05N0
CSE-4

WEEK 1

Write a program using python to:
1. a) Generate a Calendar for the given month and year?

```
import calendar
y=int(input("Enter year: "))
m=int(input("Enter month: "))

#to print calendar of given month
print(calendar.month(y,m))

#to print calendar of given year
print(calendar.calendar(y))
cal=calendar.Calendar(firstweekday=1)
for i in cal.iterweekdays():
  print(i)
```

OUTPUT:

```
Enter year: 2022
Enter month: 8
    August 2022
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31


                                  2022

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
                1  2         1  2  3  4  5  6         1  2  3  4  5  6
 3  4  5  6  7  8  9       7  8  9 10 11 12 13       7  8  9 10 11 12 13
10 11 12 13 14 15 16      14 15 16 17 18 19 20      14 15 16 17 18 19 20
17 18 19 20 21 22 23      21 22 23 24 25 26 27      21 22 23 24 25 26 27
24 25 26 27 28 29 30      28                        28 29 30 31
31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
             1  2  3                        1                 1  2  3  4  5
 4  5  6  7  8  9 10       2  3  4  5  6  7  8       6  7  8  9 10 11 12
11 12 13 14 15 16 17       9 10 11 12 13 14 15      13 14 15 16 17 18 19
18 19 20 21 22 23 24      16 17 18 19 20 21 22      20 21 22 23 24 25 26
25 26 27 28 29 30         23 24 25 26 27 28 29      27 28 29 30
```

```
        July                August              September
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
            1  2  3    1  2  3  4  5  6  7             1  2  3  4
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    5  6  7  8  9 10 11
11 12 13 14 15 16 17   15 16 17 18 19 20 21   12 13 14 15 16 17 18
18 19 20 21 22 23 24   22 23 24 25 26 27 28   19 20 21 22 23 24 25
25 26 27 28 29 30 31   29 30 31              26 27 28 29 30

      October             November            December
Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su   Mo Tu We Th Fr Sa Su
               1  2       1  2  3  4  5  6             1  2  3  4
 3  4  5  6  7  8  9    7  8  9 10 11 12 13    5  6  7  8  9 10 11
10 11 12 13 14 15 16   14 15 16 17 18 19 20   12 13 14 15 16 17 18
17 18 19 20 21 22 23   21 22 23 24 25 26 27   19 20 21 22 23 24 25
24 25 26 27 28 29 30   28 29 30              26 27 28 29 30 31
31

1
2
3
4
5
6
0
```

b). Implement a Simple Calculator program?

```
def add(x,y):
  return x+y

def sub(x,y):
  return x-y

def multiplicate(x,y):
  return x*y

def divide(x,y):
  return x/y

def module(x,y):
  return x%y

print("Enter two numbers: ")
x=int(input())
y=int(input())
print("1.Addition\n2.Subtraction\n3.Multiplication\n4.Division\
n5.Modular div")
choice= int(input("enter choice:"))
if(choice==1):
  print("Addition is ",add(x,y))
elif(choice==2):
  print("Subtraction is",sub(x,y))
```

```
elif(choice==3):
  print("Multiplication is",multiplicate(x,y))
elif(choice==4):
  print("Division is ",divide(x,y))
elif(choice==5):
  print("Modular division ",module(x,y))
else:
  print("Choose correct option")
```

OUTPUT:

```
Enter two numbers:
2
8
1.Addition
2.Subtraction
3.Multiplication
4.Division
5.Modular div
enter choice:4
Division is  0.25
```

WEEK 2

2. Design of Intelligent systems. (Suggested exercise: to control the VACUUM Cleaner moves)

CODE:

```
goal = {'A':0, 'B':0}
cost = 0
location = input("Enter the location of vacuum cleaner: ")
status_1 = input('Enter the status of '+location+' : ')
status_2 = input("Enter the status of another location: ")
if location == 'A':
  print('Vacuum Cleaner is in location A')
  if status_1 == '1':
    print('A is Dirty')
    goal['A'] = '0'
    cost += 1
    print('Location A is cleaned. The cost for cleaning is :',cost
)
    if status_2 == '1':
      print('Location B is dirty')
      print("Moving RIGHT")
      cost += 1
```

```
      print('Cost for moving RIGHT: ',cost)
      goal['B'] = '0'
      cost += 1
      print("Location B is cleaned. The cost of cleaning B is: ",c
ost)
    else:
      print("Location B is cleaned. So no Action needed")
  else:
    print("Location A is cleaned. So no Action needed.")
    if status_2 == '1':
      print('Location B is dirty')
      print("Moving RIGHT")
      cost += 1
      print('Cost for moving RIGHT: ',cost)
      goal['B'] = '0'
      cost += 1
      print("Location B is cleaned. The cost of cleaning B is: ",c
ost)
    else:
      print("Location B is cleaned. So no Action needed")
else:
  print('Vacuum Cleaner is in location B')
  if status_1 == '1':
    print('Location B is Dirty')
    goal['B'] = '0'
    cost += 1
    print('Location B is cleaned. The cost for cleaning is :',cost
)
    if status_2 == '1':
      print('Location A is dirty')
      print("Moving LEFT")
      cost += 1
      print('Cost for moving LEFT: ',cost)
      goal['A'] = '0'
      cost += 1
      print("Location A is cleaned. The cost of cleaning A is: ",c
ost)
    else:
      print("Location A is cleaned. So no Action needed")
  else:
    print("Location B is cleaned. So no Action needed.")
    if status_2 == '1':
      print('Location A is dirty')
      print("Moving LEFT")
      cost += 1
      print('Cost for moving LEFT: ',cost)
      goal['A'] = '0'
      cost += 1
      print("Location A is cleaned. The cost of cleaning A is: ",c
ost)
    else:
      print("Location A is cleaned. So no Action needed")
```

```
print(goal)
print('The total Cost: ',cost)
```

OUTPUT:

```
Enter the location of vacuum cleaner: A
Enter the status of A : 1
Enter the status of another location: 0
Vacuum Cleaner is in location A
A is Dirty
Location A is cleaned. The cost for cleaning is : 1
Location B is cleaned. So no Action needed
{'A': '0', 'B': 0}
The total Cost:  1
```

WEEK 3

3. Implement the production system and derive a solution for the real world AI problem. (Suggested exercise: Write a program to solve the following problem: You have two jugs, a 4- gallon and a 3-gallon. Neither of the jugs has markings on them. There is a pump that can be used to fill the jugs with water. How can you get exactly two gallons of water in the 4-gallon jug?).

CODE:

```
x=0
y=0
n=2
count=0
a=4
b=3
if(n<=a):
  while(x!=n):
    if(x==0):
      x=a
      print(x,y)
    elif(y==b):
      y=0
      print(x,y)
    else:
      t=min(b-y,x)
      y+=t
      x-=t
      print(x,y)
    count+=1

print("Task Completion Cost: ",count)
```

OUTPUT:

```
4 0
1 3
1 0
0 1
4 1
2 3
Task Completion Cost:  6
```

WEEK 4

4. Implement A* algorithm. (Suggested exercise: to find the shortest path).

CODE:

```
import sys
# Function to find out which of the unvisited node
# needs to be visited next

def to_be_visited():
  global visited_and_distance
  v = -10

  # Choosing the vertex with the minimum distance
  for index in range(number_of_vertices):
    if visited_and_distance[index][0] == 0 \
      and (v < 0 or visited_and_distance[index][1] <= \
      visited_and_distance[v][1]):
        v = index
  return v

# Creating the graph as an adjacency matrix
vertices = [[0, 1, 1, 0],
            [0, 0, 1, 0],
            [0, 0, 0, 1],
            [0, 0, 0, 0]]
edges =  [[0, 3, 4, 0],
          [0, 0, 0.5, 0],
          [0, 0, 0, 1],
          [0, 0, 0, 0]]

number_of_vertices = len(vertices[0])

# The first element of the lists inside visited_and_distance
# denotes if the vertex has been visited.
# The second element of the lists inside the visited_and_distance
# denotes the distance from the source.
```

```
visited_and_distance = [[0, 0]]
for i in range(number_of_vertices-1):
  visited_and_distance.append([0, sys.maxsize])
for vertex in range(number_of_vertices):
  # Finding the next vertex to be visited.
  to_visit = to_be_visited()
  for neighbor_index in range(number_of_vertices):
    # Calculating the new distance for all unvisited neighbours
    # of the chosen vertex.
    if vertices[to_visit][neighbor_index] == 1 and \
     visited_and_distance[neighbor_index][0] == 0:
      new_distance = visited_and_distance[to_visit][1] \
      + edges[to_visit][neighbor_index]
      # Updating the distance of the neighbor if its current dista
nce
      # is greater than the distance that has just been calculated
      if visited_and_distance[neighbor_index][1] > new_distance:
        visited_and_distance[neighbor_index][1] = new_distance
  # Visiting the vertex found earlier
  visited_and_distance[to_visit][0] = 1
i = 0
# Printing out the shortest distance from the source to each verte
x
for distance in visited_and_distance:
  print("The shortest distance of ",chr(ord('a') + i),\
  " from the source vertex a is:",distance[1])
  i = i + 1
```

OUTPUT:

```
The shortest distance of  a  from the source vertex a is: 0
The shortest distance of  b  from the source vertex a is: 3
The shortest distance of  c  from the source vertex a is: 3.5
The shortest distance of  d  from the source vertex a is: 4.5
```

WEEK 5

5. Implement the Constraint Specific Problem. (Suggested exercise: a crossword puzzle).
CODE:

```
sols=[]
for s in range(9,-1,-1):
  for e in range(9,-1,-1):
    for n in range(9,-1,-1):
      for d in range(9,-1,-1):
```

```
for m in range(9,-1,-1):
  for o in range(9,-1,-1):
    for r in range(9,-1,-1):
      for y in range(9,-1,-1):
        if len(set([s,e,n,d,m,o,r,y]))==8:
          send = (1000*s + 100*e + 10*n + d)
          more = (1000*m + 100*o + 10*r + e)
          money =(10000*m + 1000*o + 100*n + 10*e + y)
          if (send+more==money):
            print(send,more,money)
```

OUTPUT:

```
9567 1085 10652
8542 915 9457
8432 914 9346
8324 913 9237
7649 816 8465
7643 826 8469
7539 815 8354
7534 825 8359
7531 825 8356
7429 814 8243
7316 823 8139
6853 728 7581
6851 738 7589
6524 735 7259
6419 724 7143
6415 734 7149
5849 638 6487
5732 647 6379
5731 647 6378
3829 458 4287
3821 468 4289
3719 457 4176
3712 467 4179
2819 368 3187
2817 368 3185
```

WEEK 6

6. Implement the alpha-beta pruning. (Suggested exercise: for a tic toc game).

CODE:

```
import random
class TicTacToe:

    def __init__(self):
        self.board = []

    def create_board(self):
        for i in range(3):
            row = []
            for j in range(3):
                row.append('-')
            self.board.append(row)

    def get_random_first_player(self):
        return random.randint(0, 1)

    def fix_spot(self, row, col, player):
        self.board[row][col] = player

    def is_player_win(self, player):
        win = None

        n = len(self.board)

        # checking rows
        for i in range(n):
            win = True
            for j in range(n):
                if self.board[i][j] != player:
                    win = False
                    break
            if win:
                return win

        # checking columns
        for i in range(n):
            win = True
            for j in range(n):
                if self.board[j][i] != player:
                    win = False
                    break
            if win:
                return win

        # checking diagonals
        win = True
        for i in range(n):
            if self.board[i][i] != player:
                win = False
                break
        if win:
            return win
```

```
        win = True
        for i in range(n):
            if self.board[i][n - 1 - i] != player:
                win = False
                break
        if win:
            return win
        return False

        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True

    def is_board_filled(self):
        for row in self.board:
            for item in row:
                if item == '-':
                    return False
        return True

    def swap_player_turn(self, player):
        return 'X' if player == 'O' else 'O'

    def show_board(self):
        for row in self.board:
            for item in row:
                print(item, end=" ")
            print()

    def start(self):
        self.create_board()

        player = 'X' if self.get_random_first_player() == 1 else '
O'
        while True:
            print(f"Player {player} turn")

            self.show_board()

            # taking user input
            row, col = list(
                map(int, input("Enter row and column numbers to fi
x spot: ").split()))
            print()

            # fixing the spot
            self.fix_spot(row - 1, col - 1, player)

            # checking whether current player is won or not
            if self.is_player_win(player):
```

```
                print(f"Player {player} wins the game!")
                break

            # checking whether the game is draw or not
            if self.is_board_filled():
                print("Match Draw!")
                break

            # swapping the turn
            player = self.swap_player_turn(player)

        # showing the final view of board
        print()
        self.show_board()


# starting the game
tic_tac_toe = TicTacToe()
tic_tac_toe.start()
```

OUTPUT:

```
Player X turn
- - -
- - -
- - -
Enter row and column numbers to fix spot: 2 3

Player O turn
- - -
- - X
- - -
Enter row and column numbers to fix spot: 1 2

Player X turn
- O -
- - X
- - -
Enter row and column numbers to fix spot: 1 3

Player O turn
- O X
- - X
- - -
Enter row and column numbers to fix spot: 2 2

Player X turn
- O X
- O X
- - -
Enter row and column numbers to fix spot: 3 3

Player X wins the game!

- O X
- O X
- - X
```

WEEK-7

7. Design a planning system using STRIPS. (Suggested exercise: an elevator problem to move a passenger from the 1st floor to the 4th floor in a building).

CODE:

```
n=int(input("Enter No. of Blocks: "))
clear=[True]*(n+1)
on=[0]*(n+1)
goalOn=[0]*(n+1)
```

```
def putOn(x,y):
  if not clear[x]:
    putOnTable(on.index(x))
  if not clear[y]:
    putOnTable(on.index(y))
  clear[y]=False
  clear[on[x]]=True
  on[x]=y
  print("Put block", x, "on", y)

def putOnTable(x):
  if not clear[x]:
    putOnTable(on.index(x))
  clear[on[x]]=True
  on[x]=0
  print("Put block", x, "on table")

print("Initial state: ")
print("Select position of each block: 1.On Table 2.On a Block")

for i in range(1,n+1):
  print("Block", i)
  pos=int(input())
  if pos==1:
    on[i]=0
  elif pos==2:
    y=int(input("On which block: "))
    clear[y]=False
    on[i]=y

for i in range(1,n+1):
  print(i, clear[i], on[i])

print("For Goal state: ")
print("Select position of each block: 1.On Table 2.On a Block")
for i in range(1,n+1):
  print("Block", i)
  pos=int(input())
  if pos==1:
    goalOn[i]=0
  elif pos==2:
    goalOn[i]=int(input("On which block: "))

base=[]

if on!=goalOn:
  for i in range(1,n+1):
    if goalOn[i]==0:
      if on[i]!=0:
        putOnTable(i)
      base.append(i)
  while on!=goalOn:
```

```
    b=base.pop(0)
    try:
      x=goalOn.index(b)
      if on[x]!=b:
        putOn(x,b)
      base.append(x)
    except:
      pass
```

OUTPUT:

```
Enter No. of Blocks: 3
Initial state:
Select position of each block: 1.On Table 2.On a Block
Block 1
1
Block 2
1
Block 3
2
On which block: 2
1 True 0
2 False 0
3 True 2
For Goal state:
Select position of each block: 1.On Table 2.On a Block
Block 1
```