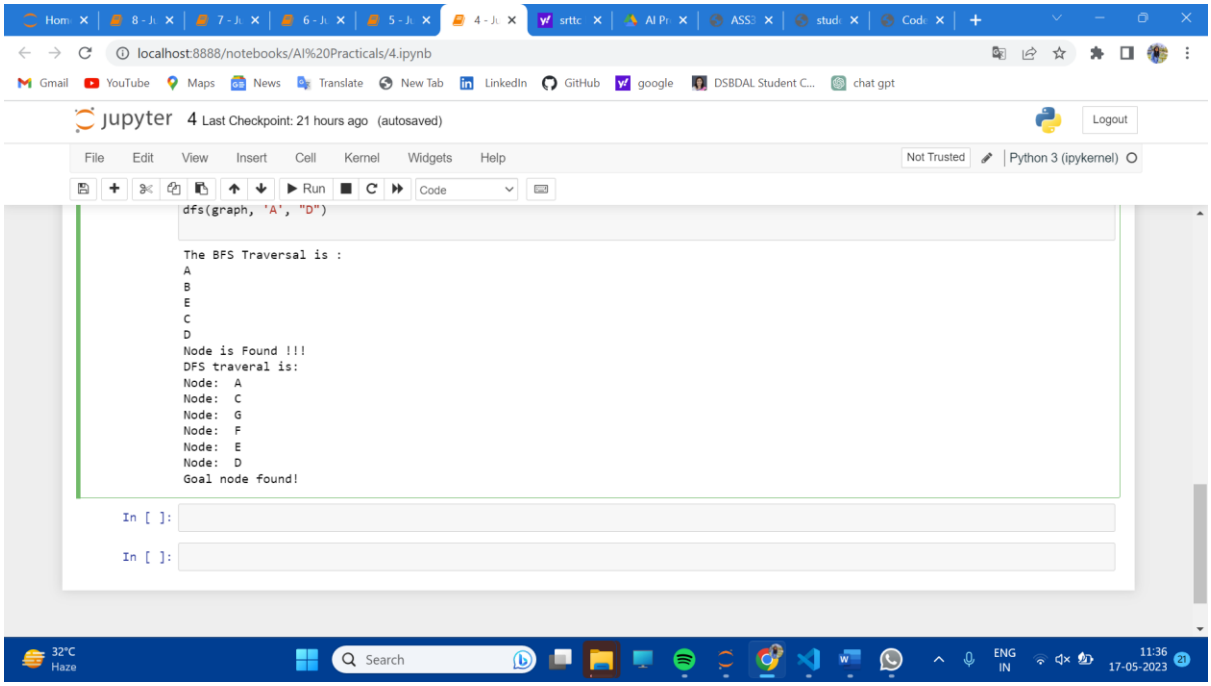
Code:

```
graph = {'A':['B', 'E', 'C'],
        'B':['A', 'D', 'E'],
        'D':['B', 'E'],
        'E':['A', 'D', 'B'],
        'C':['A', 'F', 'G'],
        'F':['C'],
        'G':['C']
        }
visited = []
queue = []

def bfs(visited, graph, start_node, goal_node):
    visited.append(start_node)
    queue.append(start_node)
    while queue:
        m = queue.pop(0)
        print(m)
        if m == goal_node:
            print("Node is Found !!! ")
            break
        else:
            for n in graph[m]:
                if n not in visited:
                    visited.append(n)
                    queue.append(n)

print("The BFS Traversal is : ")
bfs(visited, graph, 'A', 'D')

#DFS
```

```
graph = {'A':['B', 'E', 'C'],
        'B':['A', 'D', 'E'],
        'D':['B', 'E'],
        'E':['A', 'D', 'B'],
        'C':['A', 'F', 'G'],
        'F':['C'],
        'G':['C']
        }

visited = []
stack = []
def dfs(graph, start, goal):
    print("DFS traveral is: ")
    stack.append(start)
    visited.append(start)
    while stack:
        node = stack[-1]
        stack.pop()
        print("Node: ", node)
        if node == goal:
            print("Goal node found!")
            return
        for n in graph[node]:
            if n not in visited:
                visited.append(n)
                stack.append(n)

dfs(graph, 'A', "D")
```

Output:


localhost:8888/notebooks/AI%20Practicals/4.ipynb
jupyter 4 Last Checkpoint: 21 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
dfs(graph, 'A', "D")
The BFS Traversal is :
A
B
E
C
D
Node is Found !!!
DFS traveral is:
Node: A
Node: C
Node: G
Node: F
Node: E
Node: D
Goal node found!
In [ ]:
In [ ]:
32°C
Haze
Search
ENG
IN
11:36
17-05-2023

Code:

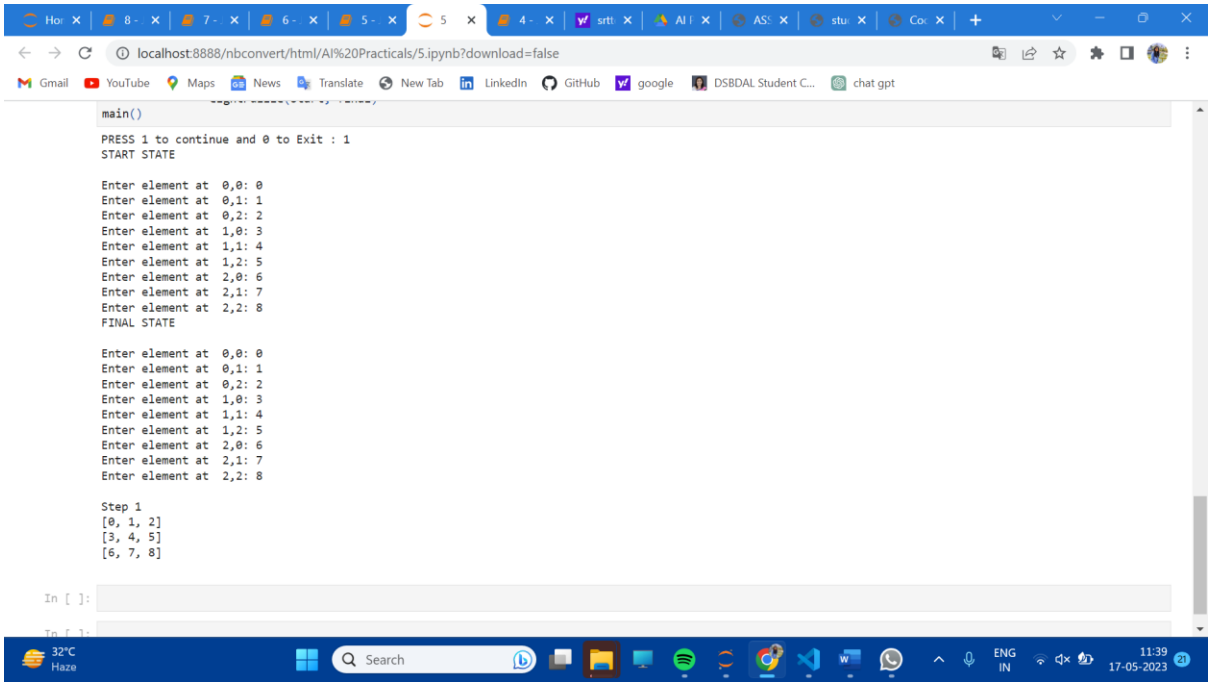
```
import copy

final = [[1,2,3],[4,5,6],[7,8,-1]]

initial = [[1,2,3],[-1,4,6],[7,5,8]]

#function to find heuristic cost

def gn(state, finalstate):
        count = 0
        for i in range(3):
                for j in range(3):
                        if(state[i][j]!=-1):
                                if(state[i][j] != finalstate[i][j]):
                                        count+=1
        return count
def findposofblank(state):
        for i in range(3):
                for j in range(3):
                        if(state[i][j] == -1):
                                return [i,j]
def move_left(state, pos):
        if(pos[1]==0):
                return None
        retarr = copy.deepcopy(state)
        retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]-1] = retarr[pos[0]][pos[1]-
1],retarr[pos[0]][pos[1]]
        return retarr
def move_up(state, pos):
        if(pos[0]==0):
                return None
        retarr = copy.deepcopy(state)
        #for i in state:
                #retarr.append(i)
```

```
        retarr[pos[0]][pos[1]],retarr[pos[0]-1][pos[1]] = retarr[pos[0]-
1][pos[1]],retarr[pos[0]][pos[1]]
        return retarr
def move_right(state, pos):
        if(pos[1]==2):
                return None
        retarr = copy.deepcopy(state)
        #for i in state:
                #retarr.append(i)
        retarr[pos[0]][pos[1]],retarr[pos[0]][pos[1]+1] =
retarr[pos[0]][pos[1]+1],retarr[pos[0]][pos[1]]
        return retarr
def move_down(state, pos):
        if(pos[0]==2):
                return None
        retarr = copy.deepcopy(state)
        retarr[pos[0]][pos[1]],retarr[pos[0]+1][pos[1]] =
retarr[pos[0]+1][pos[1]],retarr[pos[0]][pos[1]]
        return retarr
def printMatrix(matricesArray):
        print("")
        counter = 1
        for matrix in matricesArray:
                print("Step {}".format(counter))
                for row in matrix:
                        print(row)
                counter+=1
                print("")
def eightPuzzle(initialstate, finalstate):
        hn=0
        explored = []
        while(True):
```

```
            explored.append(initialstate)
            if(initialstate == finalstate):
                break
            hn+=1
            left = move_left(initialstate, findposofblank(initialstate))
            right = move_right(initialstate, findposofblank(initialstate))
            up = move_up(initialstate, findposofblank(initialstate))
            down = move_down(initialstate, findposofblank(initialstate))
            fnl=1000
            fnr=1000
            fnu=1000
            fnd=1000
            if(left!=None):
                fnl = hn + gn(left,finalstate)
            if(right!=None):
                fnr = hn + gn(right,finalstate)
            if(up!=None):
                fnu = hn + gn(up,finalstate)
            if(down!=None):
                fnd = hn + gn(down,finalstate)
            minfn = min(fnl, fnr, fnu, fnd)
            if((fnl == minfn) and (left not in explored)):
                initialstate = left
            elif((fnr == minfn) and (right not in explored)):
                initialstate = right
            elif((fnu == minfn) and (up not in explored)):
                initialstate = up
            elif((fnd == minfn) and (down not in explored)):
                initialstate = down
        printMatrix(explored)
#eightPuzzle(initial, final)
```

```
def main():
    while(True):
        ch = int(input("PRESS 1 to continue and 0 to Exit : "))
        if(not ch):
            break
        start = []
        print("START STATE\n")
        for i in range(3):
            arr=[]
            for j in range(3):
                a = int(input("Enter element at  {},{}: ".format(i,j)))
                arr.append(a)
            start.append(arr)
        final = []
        print("FINAL STATE\n")
        for i in range(3):
            arr=[]
            for j in range(3):
                a = int(input("Enter element at  {},{}: ".format(i,j)))
                arr.append(a)
            final.append(arr)
        eightPuzzle(start, final)
main()
```

Output:


localhost:8888/nbconvert/html/AI%20Practicals/5.ipynb?download=false
main()
PRESS 1 to continue and 0 to Exit : 1
START STATE
Enter element at 0,0: 0
Enter element at 0,1: 1
Enter element at 0,2: 2
Enter element at 1,0: 3
Enter element at 1,1: 4
Enter element at 1,2: 5
Enter element at 2,0: 6
Enter element at 2,1: 7
Enter element at 2,2: 8
FINAL STATE
Enter element at 0,0: 0
Enter element at 0,1: 1
Enter element at 0,2: 2
Enter element at 1,0: 3
Enter element at 1,1: 4
Enter element at 1,2: 5
Enter element at 2,0: 6
Enter element at 2,1: 7
Enter element at 2,2: 8
Step 1
[0, 1, 2]
[3, 4, 5]
[6, 7, 8]
In [ ]:
32°C
Haze
Search
ENG
IN
11:39
17-05-2023

Code:

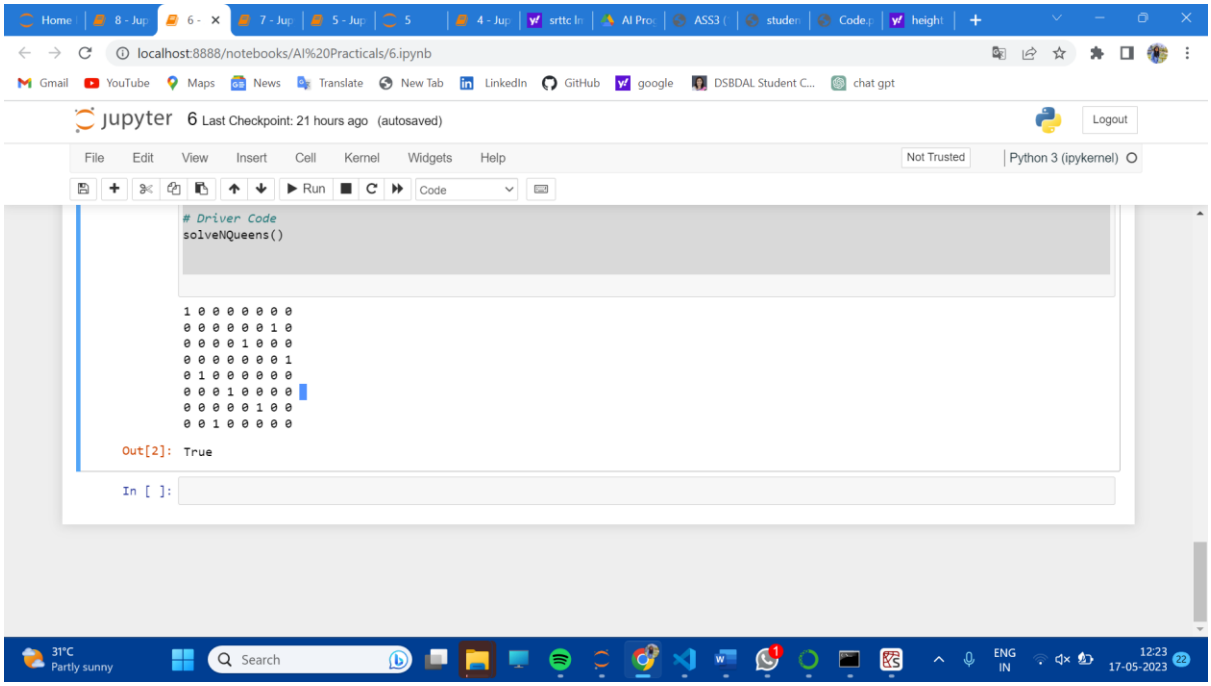
```
""" Python3 program to solve N Queen Problem
using Branch or Bound """

N = 8

""" A utility function to print solution """
def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end = " ")
        print()

""" A Optimized function to check if
a queen can be placed on board[row][col] """
def isSafe(row, col, slashCode, backslashCode,
           rowLookup, slashCodeLookup,
                                backslashCodeLookup):
    if (slashCodeLookup[slashCode[row][col]] or
        backslashCodeLookup[backslashCode[row][col]] or
        rowLookup[row]):
        return False
    return True

""" A recursive utility function
to solve N Queen problem """
def solveNQueensUtil(board, col, slashCode, backslashCode,
                                rowLookup, slashCodeLookup,
                                backslashCodeLookup):

    """ base case: If all queens are
```

```
placed then return True """
if(col >= N):
        return True
for i in range(N):
        if(isSafe(i, col, slashCode, backslashCode,
                        rowLookup, slashCodeLookup,
                        backslashCodeLookup)):

                """ Place this queen in board[i][col] """
                board[i][col] = 1
                rowLookup[i] = True
                slashCodeLookup[slashCode[i][col]] = True
                backslashCodeLookup[backslashCode[i][col]] = True

                """ recur to place rest of the queens """
                if(solveNQueensUtil(board, col + 1,
                                                slashCode, backslashCode,
                                                rowLookup, slashCodeLookup,
                                                backslashCodeLookup)):
                        return True

                """ If placing queen in board[i][col]
                doesn't lead to a solution,then backtrack """

                """ Remove queen from board[i][col] """
                board[i][col] = 0
                rowLookup[i] = False
                slashCodeLookup[slashCode[i][col]] = False
                backslashCodeLookup[backslashCode[i][col]] = False

""" If queen can not be place in any row in
```

```
        this column col then return False """
        return False

""" This function solves the N Queen problem using
Branch or Bound. It mainly uses solveNQueensUtil()to
solve the problem. It returns False if queens
cannot be placed,otherwise return True or
prints placement of queens in the form of 1s.
Please note that there may be more than one
solutions,this function prints one of the
feasible solutions."""
def solveNQueens():
        board = [[0 for i in range(N)]
                                for j in range(N)]

        # helper matrices
        slashCode = [[0 for i in range(N)]
                                        for j in range(N)]
        backslashCode = [[0 for i in range(N)]
                                                for j in range(N)]

        # arrays to tell us which rows are occupied
        rowLookup = [False] * N

        # keep two arrays to tell us
        # which diagonals are occupied
        x = 2 * N - 1
        slashCodeLookup = [False] * x
        backslashCodeLookup = [False] * x

        # initialize helper matrices
```

```
    for rr in range(N):
        for cc in range(N):
            slashCode[rr][cc] = rr + cc
            backslashCode[rr][cc] = rr - cc + 7

    if(solveNQueensUtil(board, 0, slashCode, backslashCode,
                                        rowLookup, slashCodeLookup,
                                        backslashCodeLookup) == False):
        print("Solution does not exist")
        return False

    # solution found
    printSolution(board)
    return True

# Driver Code
solveNQueens()
```

Output:


jupyter 6 Last Checkpoint: 21 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
# Driver Code
solveNQueens()
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
Out[2]: True
In [ ]:

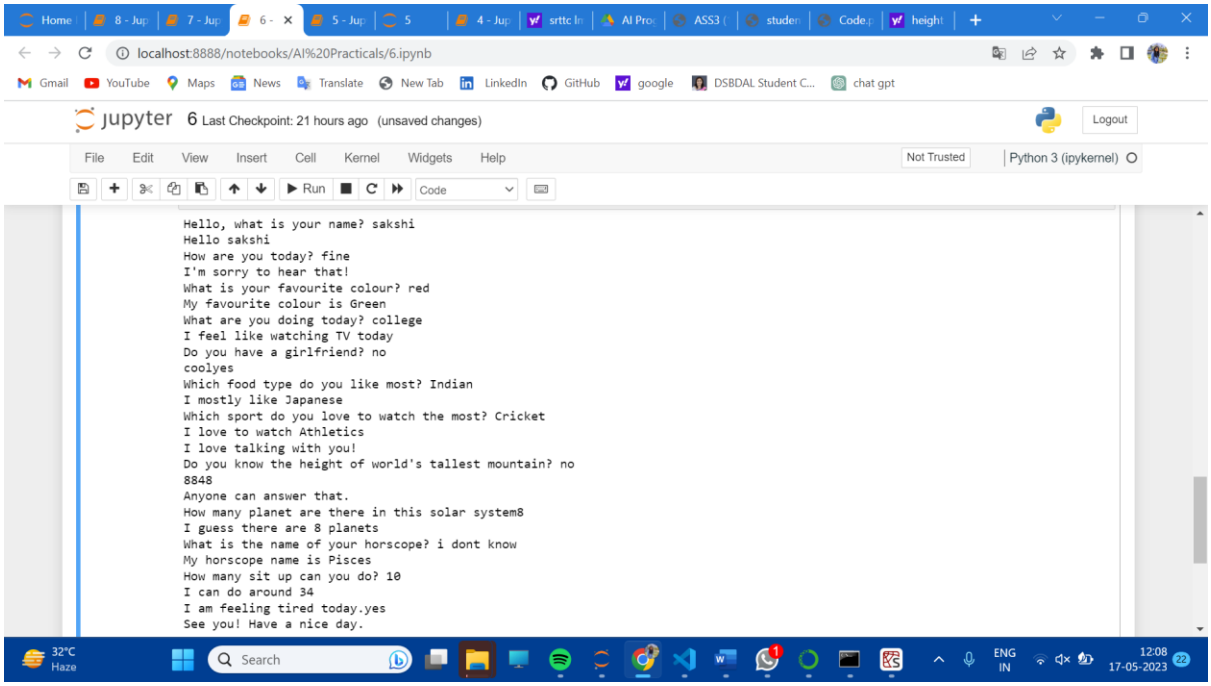
Code:

```
import time
import random

name = input("Hello, what is your name? ")

time.sleep(2)
print("Hello " + name)

feeling = input("How are you today? ")

time.sleep(1)
if "good" in feeling:
    print("I'm feeling good too!")
else:
    print("I'm sorry to hear that!")

time.sleep(1)
favcolour = input("What is your favourite colour? ")

colours = ["Red","Green","Blue"]

time.sleep(1)
print("My favourite colour is " + random.choice(colours))

mood = input("What are you doing today? ")
mood = ["dancing", "reading", "playing", "watching TV", "hiking"]
time.sleep(1)
print("I feel like " + random.choice(mood) + " today")

love=input("Do you have a girlfriend? ")
```

```
love = ["Yes", "No"]
time.sleep(1)
input("cool")

food = input("Which food type do you like most? ")
food=["Asian", "Japanese", "Italian", "Chinese", "Latin"]
time.sleep(1)
print("I mostly like " + random.choice(food))

sport = input("Which sport do you love to watch the most? ")
sport=["cirket", "Baseball", "Soccer", "Rugby", "Athletics"]
time.sleep(1)
print("I love to watch " + random.choice(sport))
time.sleep(1)
print("I love talking with you!")


peak = input("Do you know the height of world's tallest mountain? ")
time.sleep(1)
print("8848")
print("Anyone can answer that.")

planet = input("How many planet are there in this solar system")
time.sleep(1)
print("I guess there are 8 planets")

horscope = input("What is the name of your horscope? ")
horscope=["Leo", "Cancer", "Capricon", "Pisces", "Virgo"]
time.sleep(1)
print("My horscope name is " + random.choice(horscope))
```

```
exe = input("How many sit up can you do? ")
exe=["15", "34", "25", "40", "50"]
time.sleep(1)
print("I can do around " + random.choice(exe))

tired = input("I am feeling tired today.")
time.sleep(1)
print("See you! Have a nice day.")
```

Output:



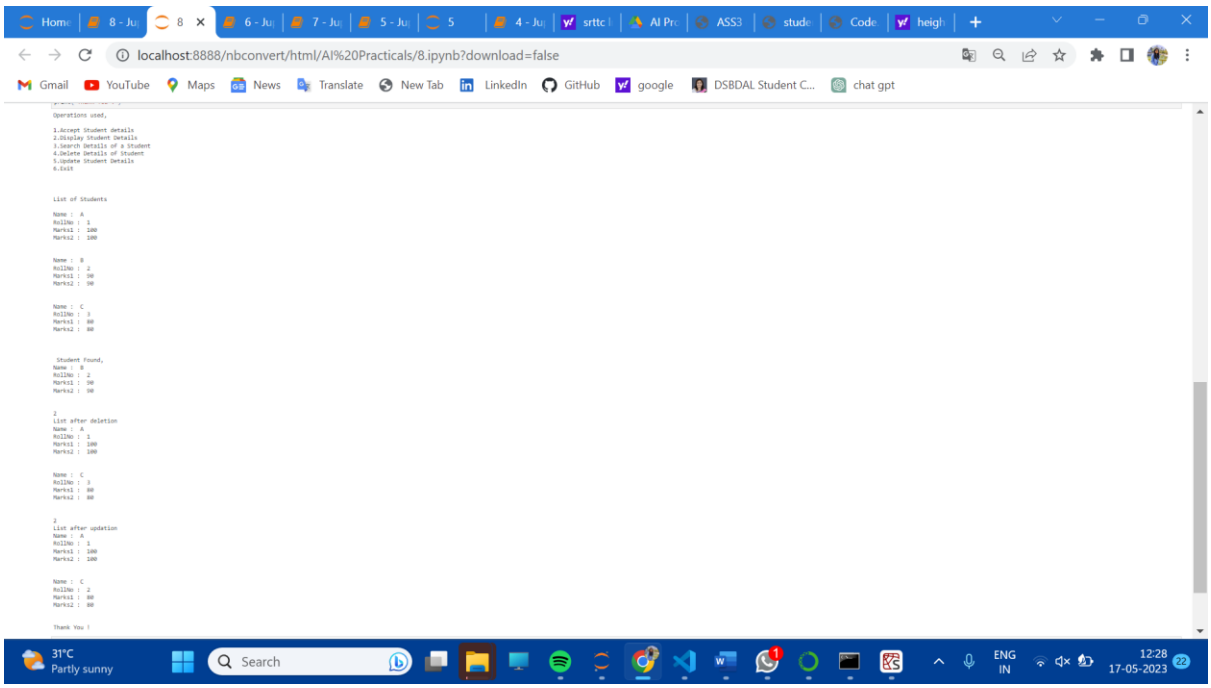
Code:

```
# This is simplest Student data management program in python


# Create class "Student"
class Student:


# Constructor
        def __init__(self, name, rollno, m1, m2):
                self.name = name
                self.rollno = rollno
                self.m1 = m1
                self.m2 = m2


        # Function to create and append new student
        def accept(self, Name, Rollno, marks1, marks2):


# use ' int(input()) ' method to take input from user
                ob = Student(Name, Rollno, marks1, marks2)
                ls.append(ob)


        # Function to display student details
        def display(self, ob):
                print("Name : ", ob.name)
                print("RollNo : ", ob.rollno)
                print("Marks1 : ", ob.m1)
                print("Marks2 : ", ob.m2)
                print("\n")


        # Search Function
        def search(self, rn):
                for i in range(ls.__len__()):
```

```
                    if(ls[i].rollno == rn):
                        return i

    # Delete Function
    def delete(self, rn):
        i = obj.search(rn)
        del ls[i]

    # Update Function
    def update(self, rn, No):
        i = obj.search(rn)
        roll = No
        ls[i].rollno = roll


# Create a list to add Students
ls = []
# an object of Student class
obj = Student('', 0, 0, 0)

print("\nOperations used, ")
print("\n1.Accept Student details\n2.Display Student Details\n3.Search Details of a
Student\n4.Delete Details of Student\n5.Update Student Details\n6.Exit")

# ch = int(input("Enter choice:"))
# if(ch == 1):
obj.accept("A", 1, 100, 100)
obj.accept("B", 2, 90, 90)
obj.accept("C", 3, 80, 80)

# elif(ch == 2):
```

```
print("\n")
print("\nList of Students\n")
for i in range(ls.__len__()):
        obj.display(ls[i])

# elif(ch == 3):
print("\n Student Found, ")
s = obj.search(2)
obj.display(ls[s])

# elif(ch == 4):
obj.delete(2)
print(ls.__len__())
print("List after deletion")
for i in range(ls.__len__()):
        obj.display(ls[i])

# elif(ch == 5):
obj.update(3, 2)
print(ls.__len__())
print("List after updation")
for i in range(ls.__len__()):
        obj.display(ls[i])

# else:
print("Thank You !")
```

Output:


localhost:8888/nbconvert/html/AI%20Practicals/8.ipynb?download=false
Operations used,
1.Accept Student details
2.Display Student Details
3.Search Details of a Student
4.Delete Details of Student
5.Update Student Details
6.Exit
List of Students
Name : A
RollNo : 1
Mark1 : 100
Mark2 : 100
Name : B
RollNo : 2
Mark1 : 90
Mark2 : 90
Name : C
RollNo : 3
Mark1 : 80
Mark2 : 80
Student Found,
Name : B
RollNo : 2
Mark1 : 90
Mark2 : 90
2
List after deletion
Name : A
RollNo : 1
Mark1 : 100
Mark2 : 100
Name : C
RollNo : 3
Mark1 : 80
Mark2 : 80
2
List after updation
Name : A
RollNo : 1
Mark1 : 100
Mark2 : 100
Name : C
RollNo : 2
Mark1 : 80
Mark2 : 80
Thank You !
31°C
Partly sunny
Search
ENG
IN
12:28
17-05-2023