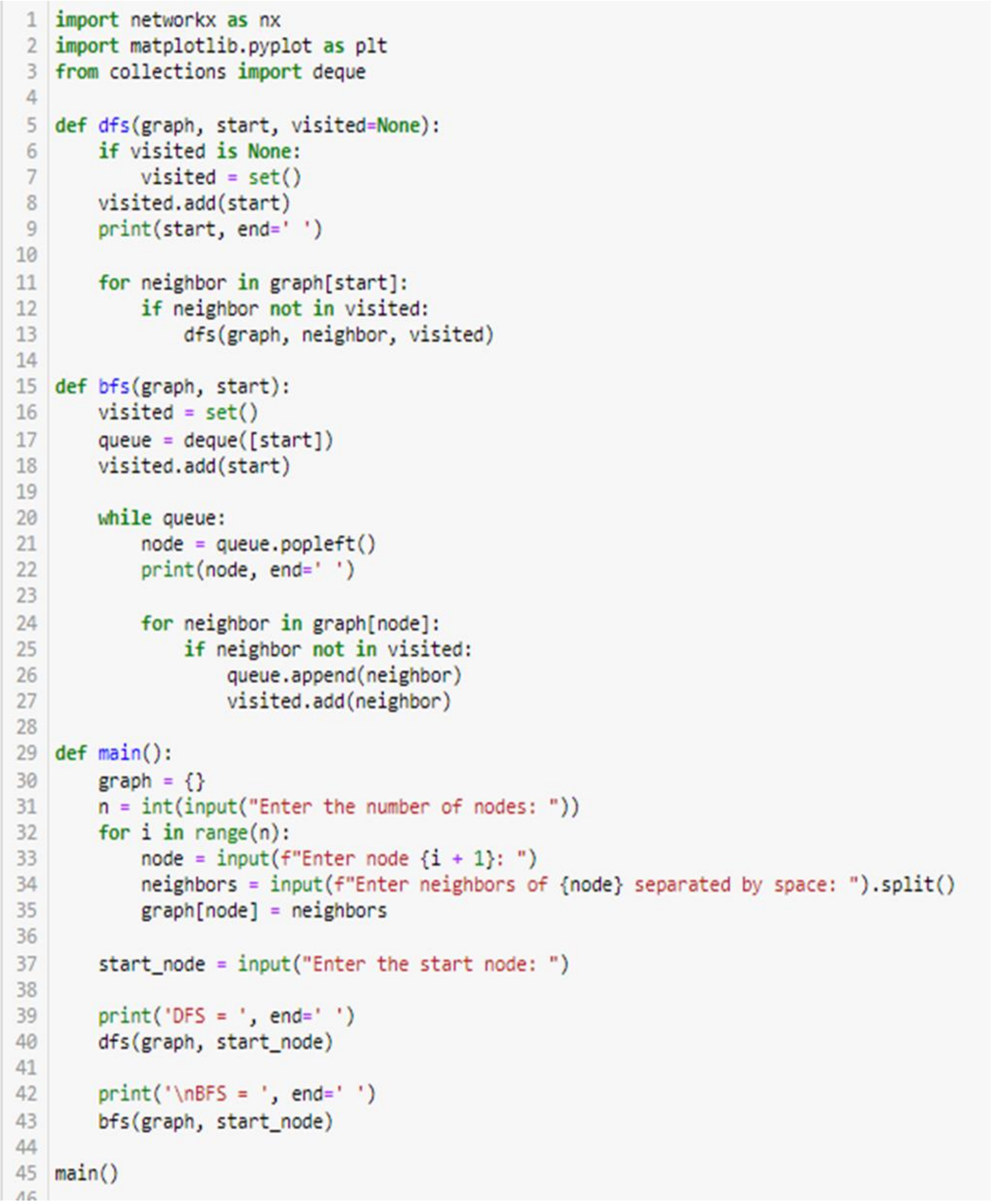
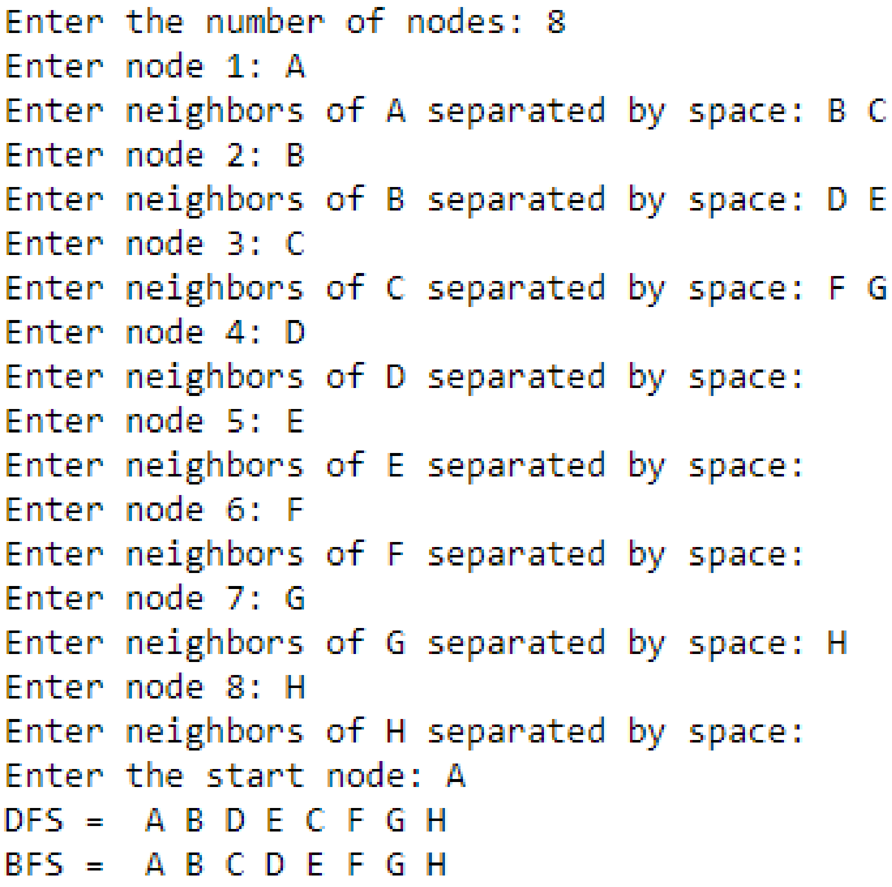
DFS & BFS code:

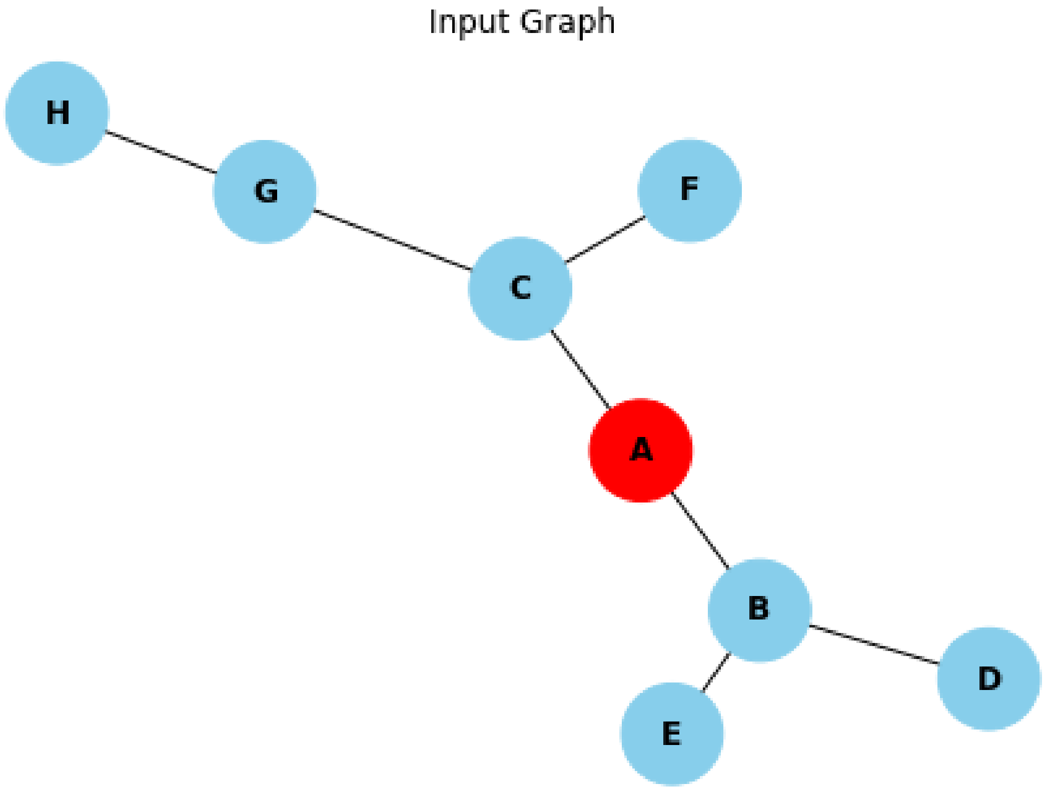


```python
import networkx as nx
import matplotlib.pyplot as plt
from collections import deque

def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add(start)
    print(start, end=' ')

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        node = queue.popleft()
        print(node, end=' ')

        for neighbor in graph[node]:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)

def main():
    graph = {}
    n = int(input("Enter the number of nodes: "))
    for i in range(n):
        node = input(f"Enter node {i + 1}: ")
        neighbors = input(f"Enter neighbors of {node} separated by space: ").split()
        graph[node] = neighbors

    start_node = input("Enter the start node: ")

    print('DFS = ', end=' ')
    dfs(graph, start_node)

    print('\nBFS = ', end=' ')
    bfs(graph, start_node)

main()
```

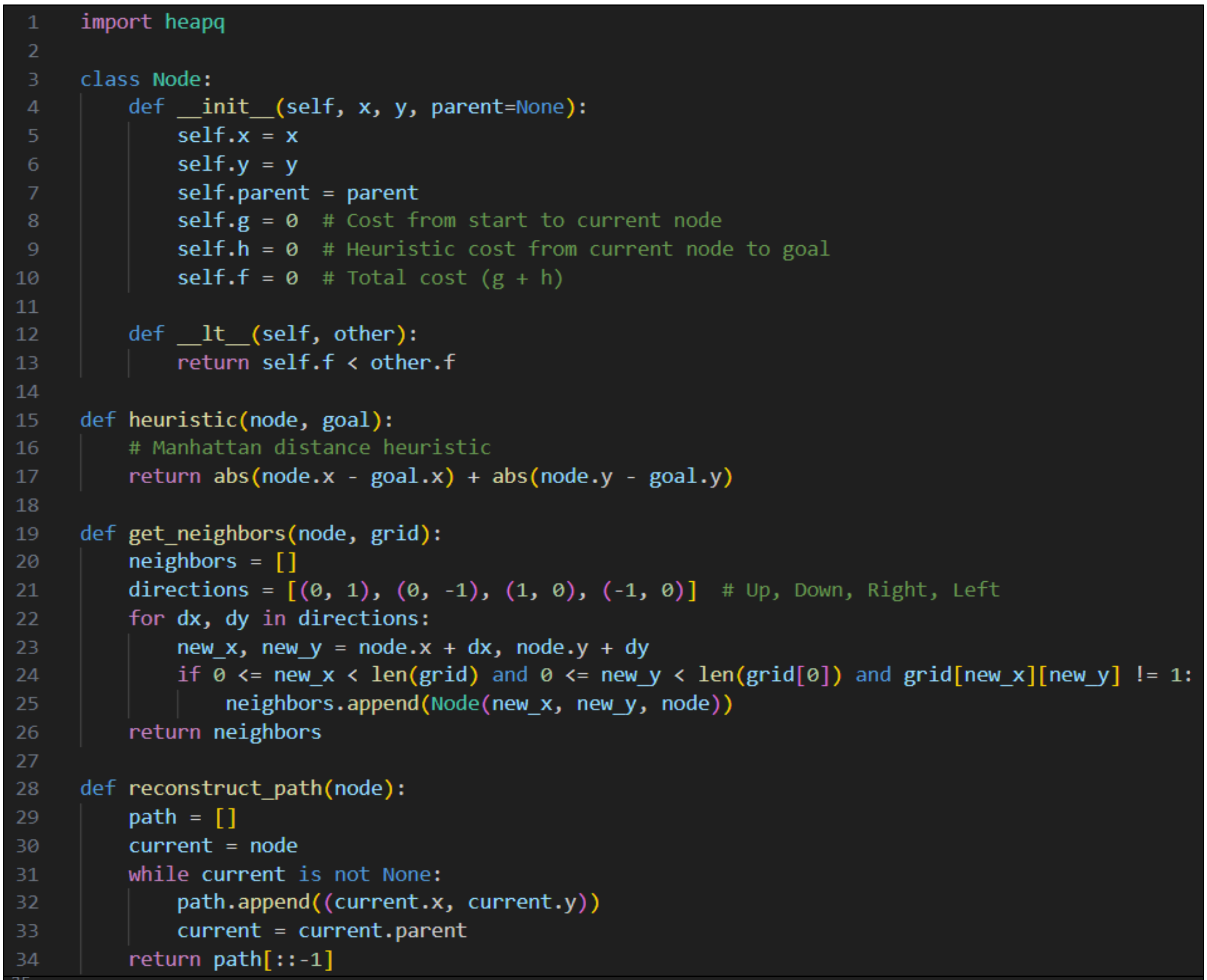
Output:



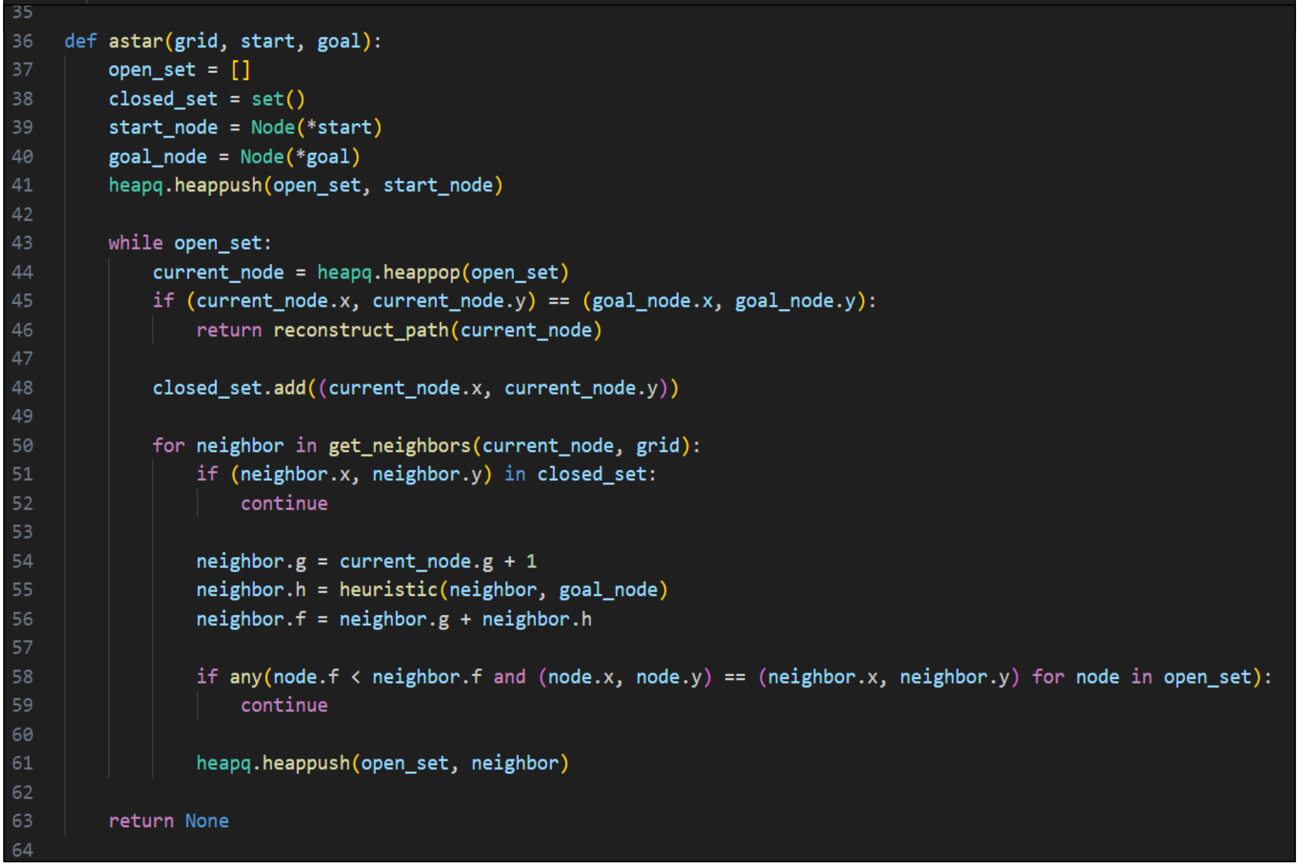
```
Enter the number of nodes: 8
Enter node 1: A
Enter neighbors of A separated by space: B C
Enter node 2: B
Enter neighbors of B separated by space: D E
Enter node 3: C
Enter neighbors of C separated by space: F G
Enter node 4: D
Enter neighbors of D separated by space:
Enter node 5: E
Enter neighbors of E separated by space:
Enter node 6: F
Enter neighbors of F separated by space:
Enter node 7: G
Enter neighbors of G separated by space: H
Enter node 8: H
Enter neighbors of H separated by space:
Enter the start node: A
DFS =  A B D E C F G H
BFS =  A B C D E F G H
```



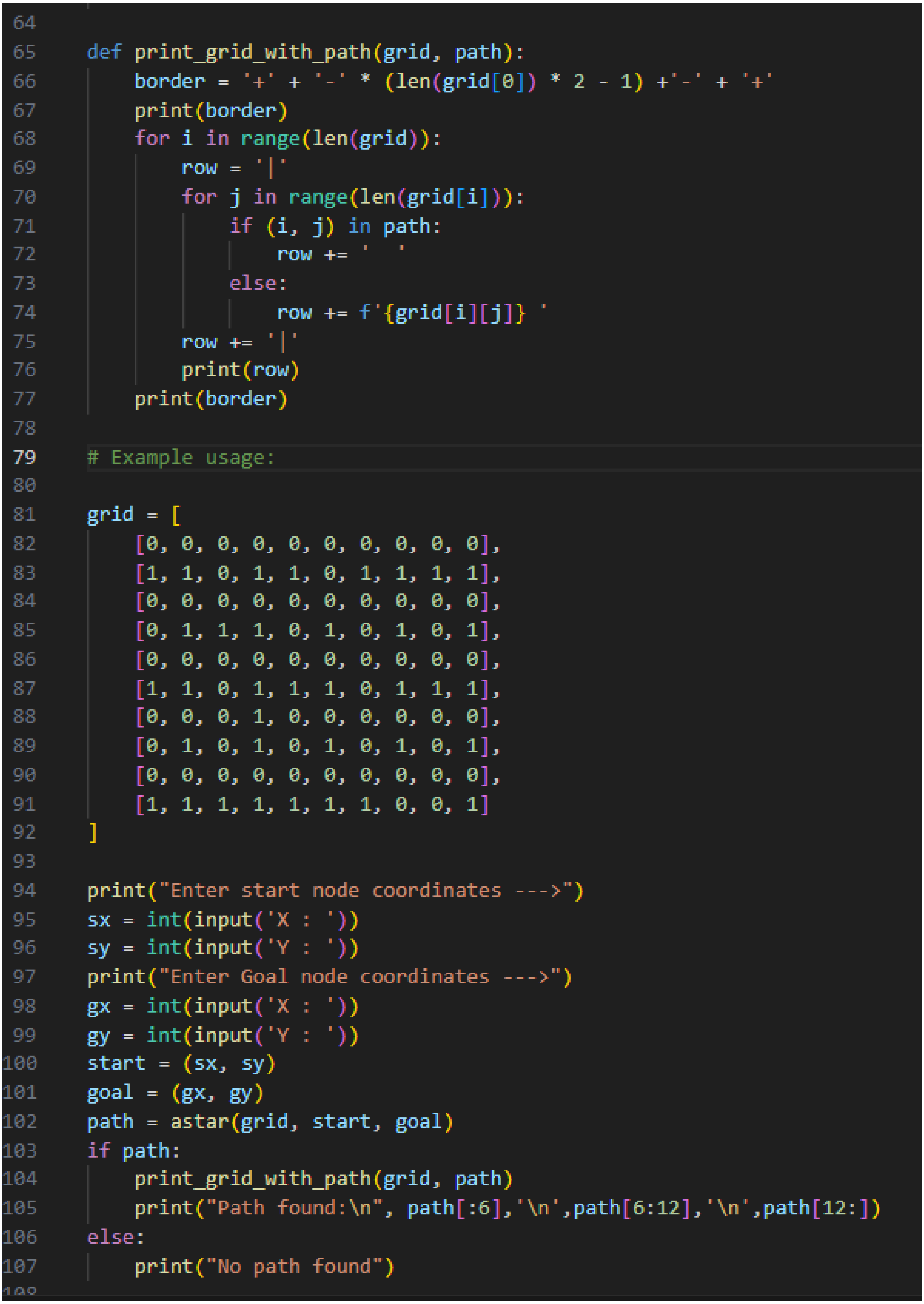
A* Search Algorithm code:



```python
import heapq

class Node:
    def __init__(self, x, y, parent=None):
        self.x = x
        self.y = y
        self.parent = parent
        self.g = 0  # Cost from start to current node
        self.h = 0  # Heuristic cost from current node to goal
        self.f = 0  # Total cost (g + h)

    def __lt__(self, other):
        return self.f < other.f

def heuristic(node, goal):
    # Manhattan distance heuristic
    return abs(node.x - goal.x) + abs(node.y - goal.y)

def get_neighbors(node, grid):
    neighbors = []
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]  # Up, Down, Right, Left
    for dx, dy in directions:
        new_x, new_y = node.x + dx, node.y + dy
        if 0 <= new_x < len(grid) and 0 <= new_y < len(grid[0]) and grid[new_x][new_y] != 1:
            neighbors.append(Node(new_x, new_y, node))
    return neighbors

def reconstruct_path(node):
    path = []
    current = node
    while current is not None:
        path.append((current.x, current.y))
        current = current.parent
    return path[::-1]
```

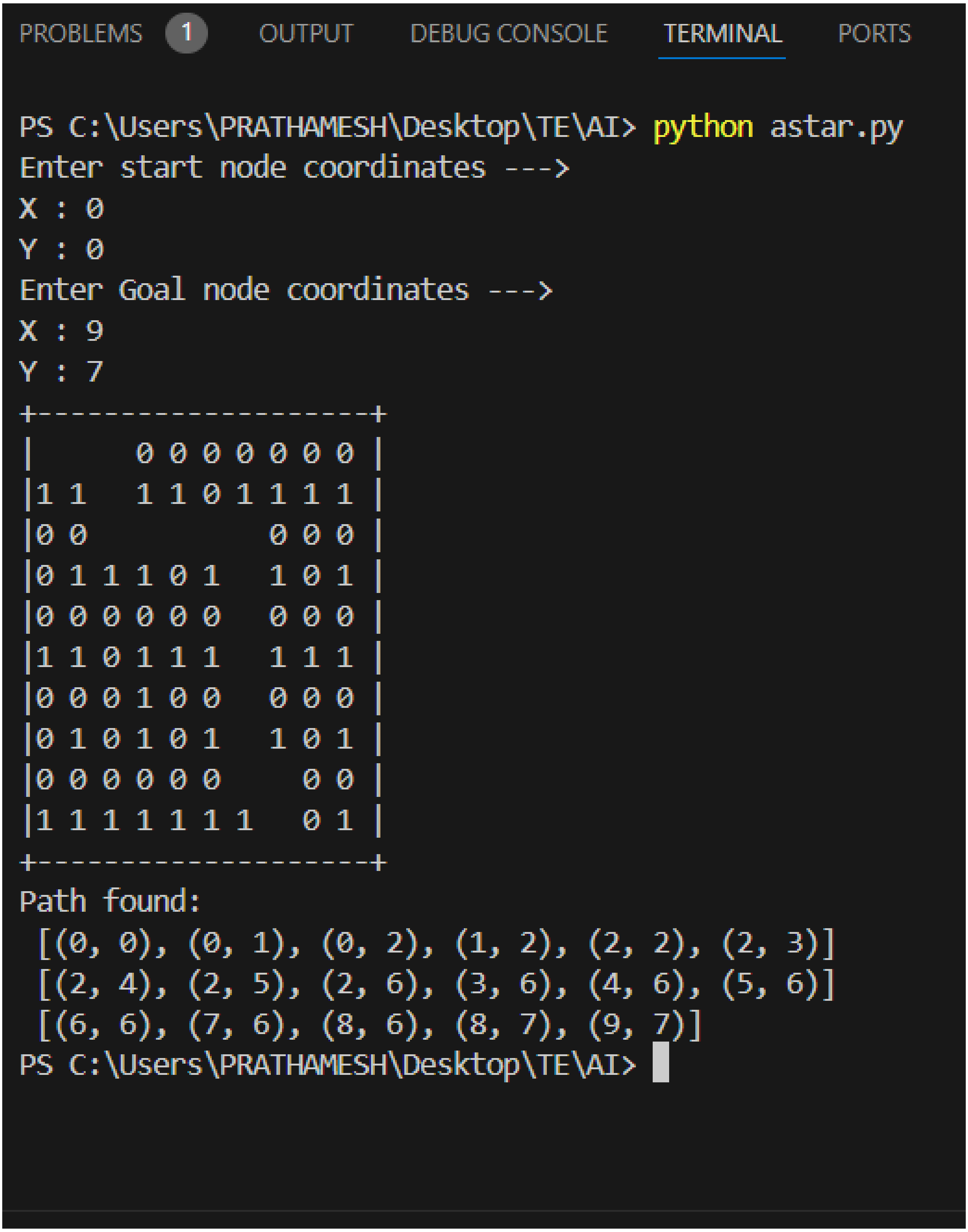


```python

def astar(grid, start, goal):
    open_set = []
    closed_set = set()
    start_node = Node(*start)
    goal_node = Node(*goal)
    heapq.heappush(open_set, start_node)

    while open_set:
        current_node = heapq.heappop(open_set)
        if (current_node.x, current_node.y) == (goal_node.x, goal_node.y):
            return reconstruct_path(current_node)

        closed_set.add((current_node.x, current_node.y))

        for neighbor in get_neighbors(current_node, grid):
            if (neighbor.x, neighbor.y) in closed_set:
                continue

            neighbor.g = current_node.g + 1
            neighbor.h = heuristic(neighbor, goal_node)
            neighbor.f = neighbor.g + neighbor.h

            if any(node.f < neighbor.f and (node.x, node.y) == (neighbor.x, neighbor.y) for node in open_set):
                continue

            heapq.heappush(open_set, neighbor)

    return None

```



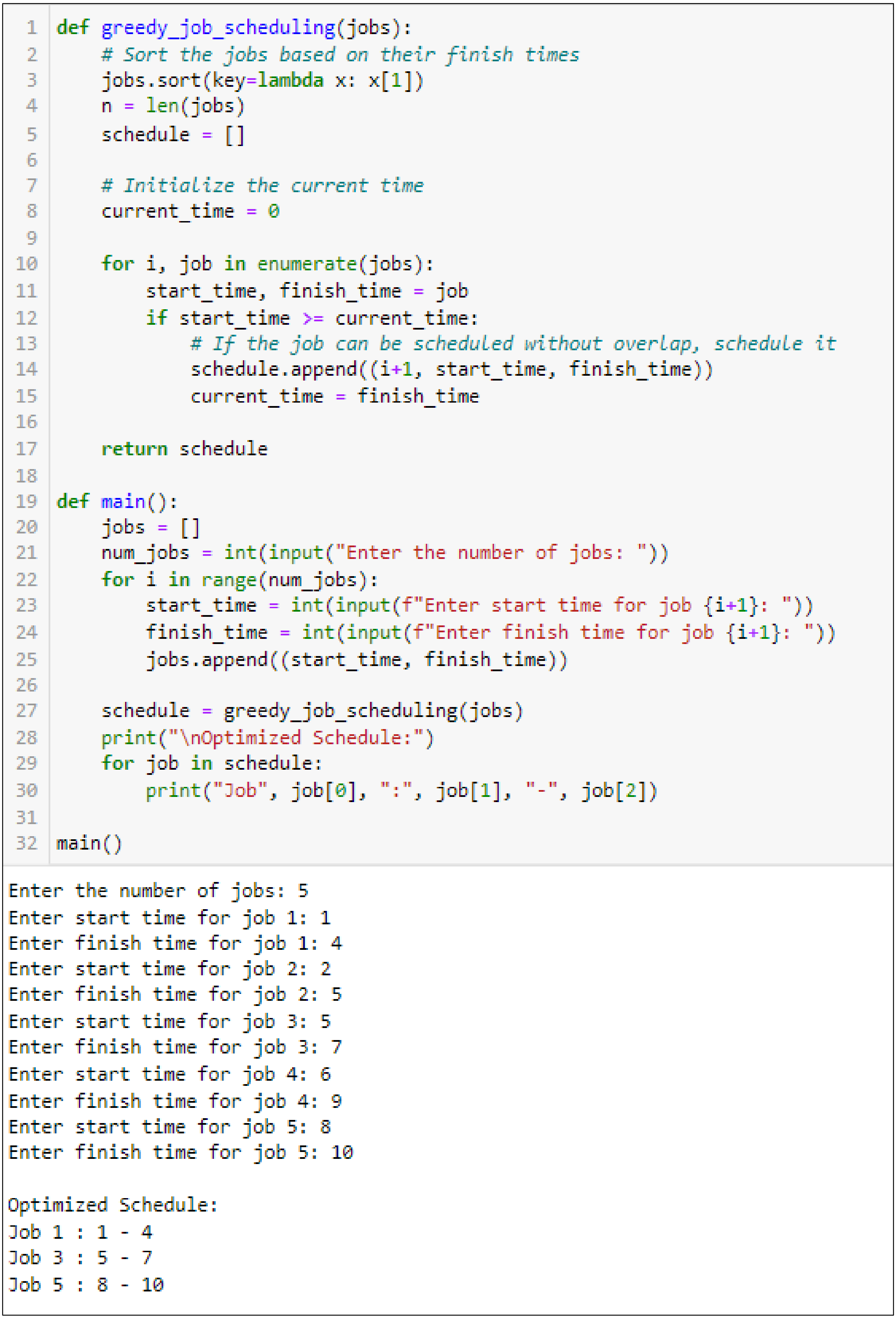
```python

def print_grid_with_path(grid, path):
    border = '+' + '-' * (len(grid[0]) * 2 - 1) +'-' + '+'
    print(border)
    for i in range(len(grid)):
        row = '|'
        for j in range(len(grid[i])):
            if (i, j) in path:
                row += '  '
            else:
                row += f'{grid[i][j]} '
        row += '|'
        print(row)
    print(border)

# Example usage:

grid = [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 1, 0, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 1, 1, 1, 0, 1, 0, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 0, 1, 1, 1, 0, 1, 1, 1],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
    [0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [1, 1, 1, 1, 1, 1, 1, 0, 0, 1]
]

print("Enter start node coordinates --->")
sx = int(input('X : '))
sy = int(input('Y : '))
print("Enter Goal node coordinates --->")
gx = int(input('X : '))
gy = int(input('Y : '))
start = (sx, sy)
goal = (gx, gy)
path = astar(grid, start, goal)
if path:
    print_grid_with_path(grid, path)
    print("Path found:\n", path[:6],'\n',path[6:12],'\n',path[12:])
else:
    print("No path found")
```

Output:



```
PS C:\Users\PRATHAMESH\Desktop\TE\AI> python astar.py
Enter start node coordinates --->
X : 0
Y : 0
Enter Goal node coordinates --->
X : 9
Y : 7
+--------------------+
|      0 0 0 0 0 0 0 |
|1 1   1 1 0 1 1 1 1 |
|0 0           0 0 0 |
|0 1 1 1 0 1   1 0 1 |
|0 0 0 0 0 0   0 0 0 |
|1 1 0 1 1 1   1 1 1 |
|0 0 0 1 0 0   0 0 0 |
|0 1 0 1 0 1   1 0 1 |
|0 0 0 0 0 0     0 0 |
|1 1 1 1 1 1 1   0 1 |
+--------------------+
Path found:
 [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3)]
 [(2, 4), (2, 5), (2, 6), (3, 6), (4, 6), (5, 6)]
 [(6, 6), (7, 6), (8, 6), (8, 7), (9, 7)]
PS C:\Users\PRATHAMESH\Desktop\TE\AI>
```

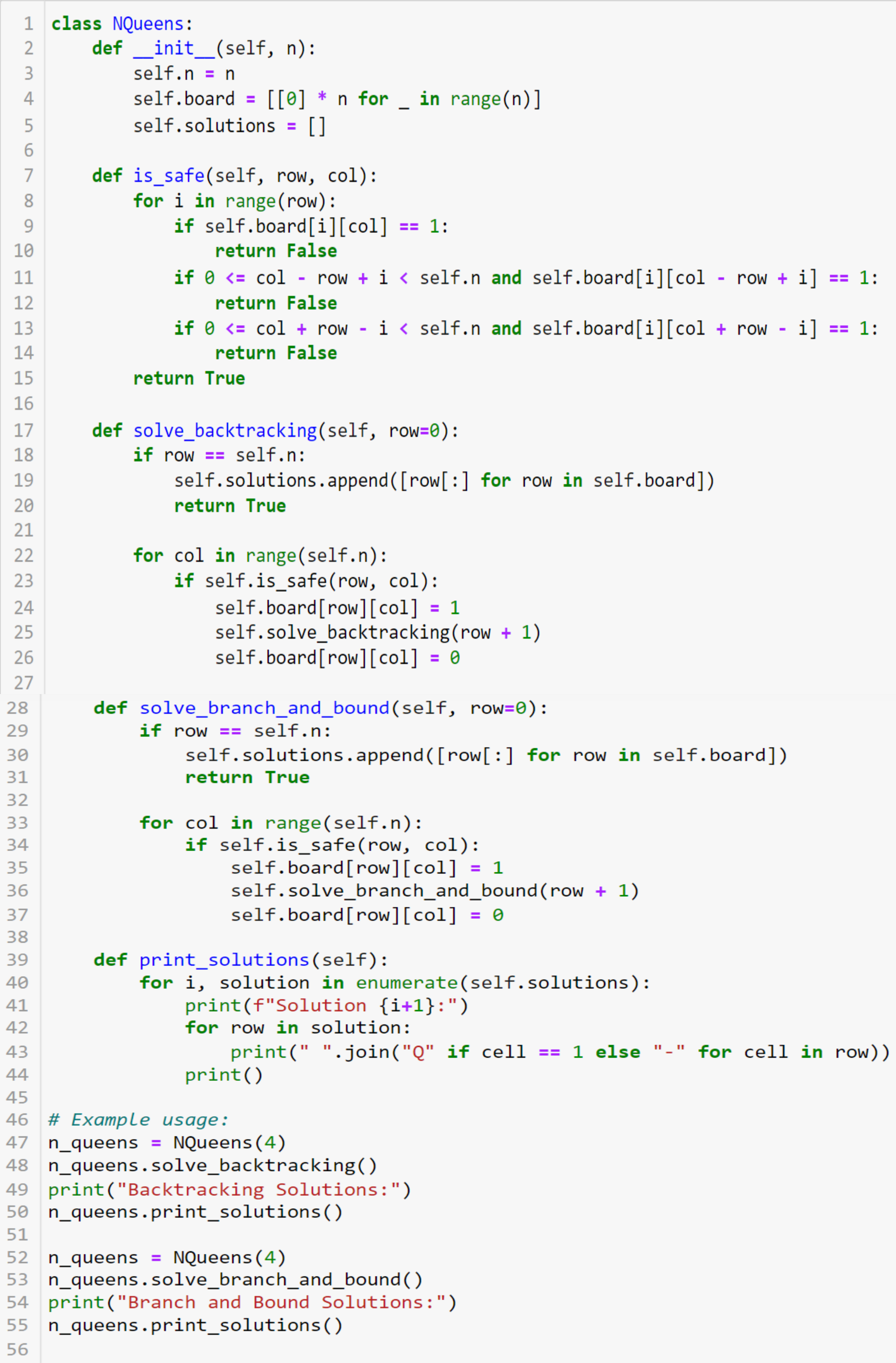
Greedy Search – job scheduing problem



```python
def greedy_job_scheduling(jobs):
    # Sort the jobs based on their finish times
    jobs.sort(key=lambda x: x[1])
    n = len(jobs)
    schedule = []

    # Initialize the current time
    current_time = 0

    for i, job in enumerate(jobs):
        start_time, finish_time = job
        if start_time >= current_time:
            # If the job can be scheduled without overlap, schedule it
            schedule.append((i+1, start_time, finish_time))
            current_time = finish_time

    return schedule

def main():
    jobs = []
    num_jobs = int(input("Enter the number of jobs: "))
    for i in range(num_jobs):
        start_time = int(input(f"Enter start time for job {i+1}: "))
        finish_time = int(input(f"Enter finish time for job {i+1}: "))
        jobs.append((start_time, finish_time))

    schedule = greedy_job_scheduling(jobs)
    print("\nOptimized Schedule:")
    for job in schedule:
        print("Job", job[0], ":", job[1], "-", job[2])

main()
```

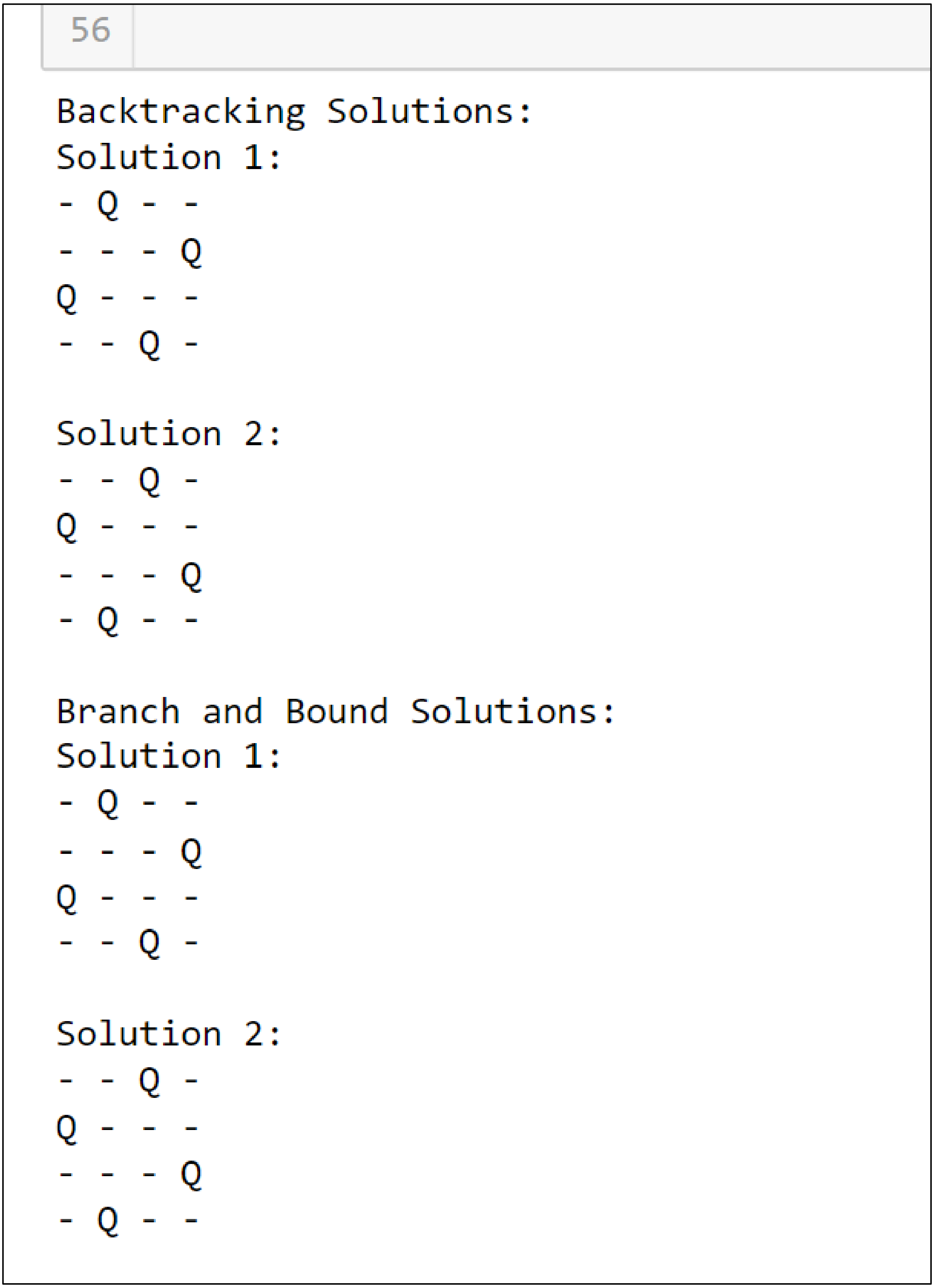
```
Enter the number of jobs: 5
Enter start time for job 1: 1
Enter finish time for job 1: 4
Enter start time for job 2: 2
Enter finish time for job 2: 5
Enter start time for job 3: 5
Enter finish time for job 3: 7
Enter start time for job 4: 6
Enter finish time for job 4: 9
Enter start time for job 5: 8
Enter finish time for job 5: 10

Optimized Schedule:
Job 1 : 1 - 4
Job 3 : 5 - 7
Job 5 : 8 - 10
```

CSP : N - Queens Problem



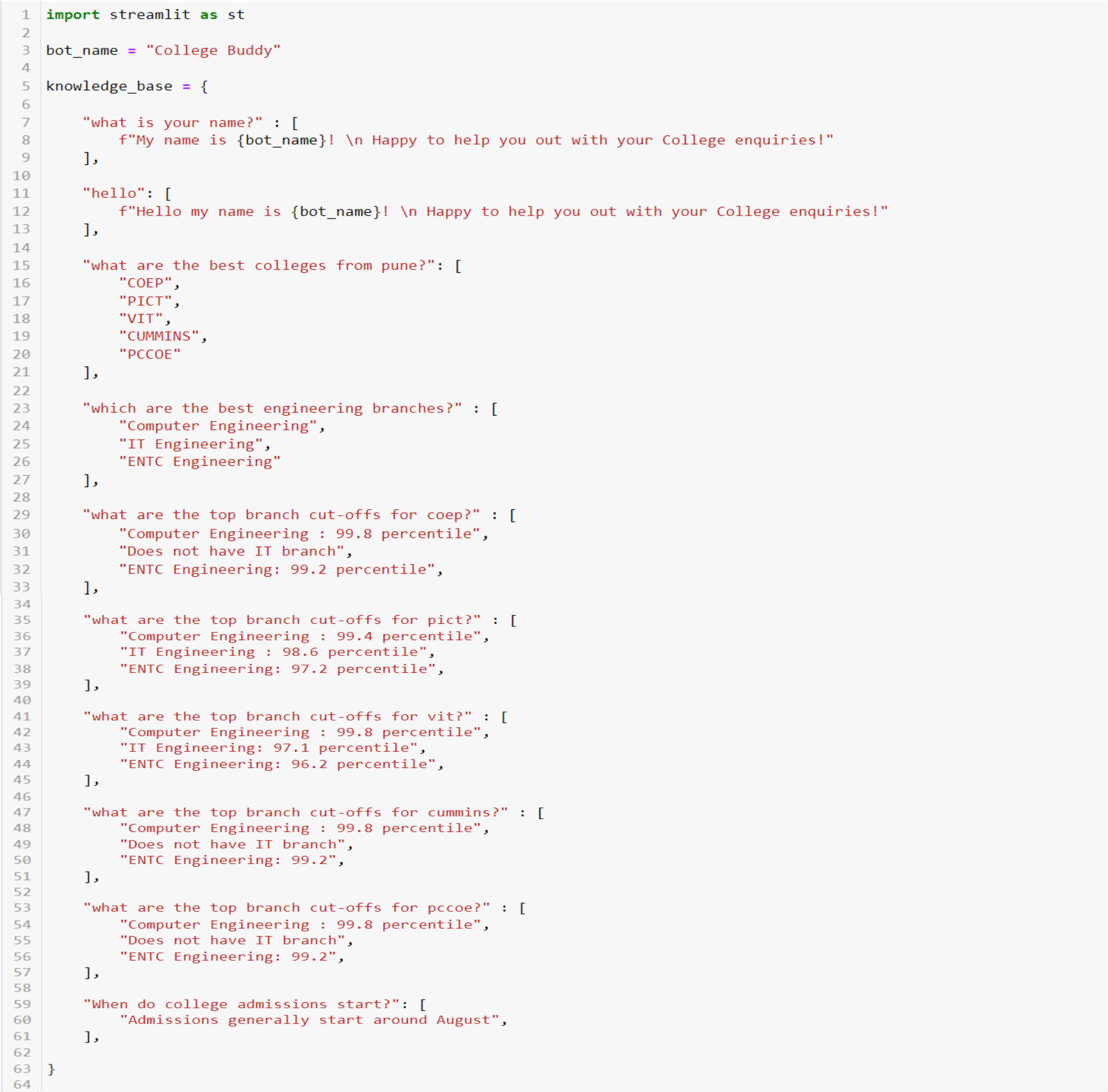
```python
class NQueens:
    def __init__(self, n):
        self.n = n
        self.board = [[0] * n for _ in range(n)]
        self.solutions = []

    def is_safe(self, row, col):
        for i in range(row):
            if self.board[i][col] == 1:
                return False
            if 0 <= col - row + i < self.n and self.board[i][col - row + i] == 1:
                return False
            if 0 <= col + row - i < self.n and self.board[i][col + row - i] == 1:
                return False
        return True

    def solve_backtracking(self, row=0):
        if row == self.n:
            self.solutions.append([row[:] for row in self.board])
            return True

        for col in range(self.n):
            if self.is_safe(row, col):
                self.board[row][col] = 1
                self.solve_backtracking(row + 1)
                self.board[row][col] = 0

    def solve_branch_and_bound(self, row=0):
        if row == self.n:
            self.solutions.append([row[:] for row in self.board])
            return True

        for col in range(self.n):
            if self.is_safe(row, col):
                self.board[row][col] = 1
                self.solve_branch_and_bound(row + 1)
                self.board[row][col] = 0

    def print_solutions(self):
        for i, solution in enumerate(self.solutions):
            print(f"Solution {i+1}:")
            for row in solution:
                print(" ".join("Q" if cell == 1 else "-" for cell in row))
            print()

# Example usage:
n_queens = NQueens(4)
n_queens.solve_backtracking()
print("Backtracking Solutions:")
n_queens.print_solutions()

n_queens = NQueens(4)
n_queens.solve_branch_and_bound()
print("Branch and Bound Solutions:")
n_queens.print_solutions()
```

OUTPUT:

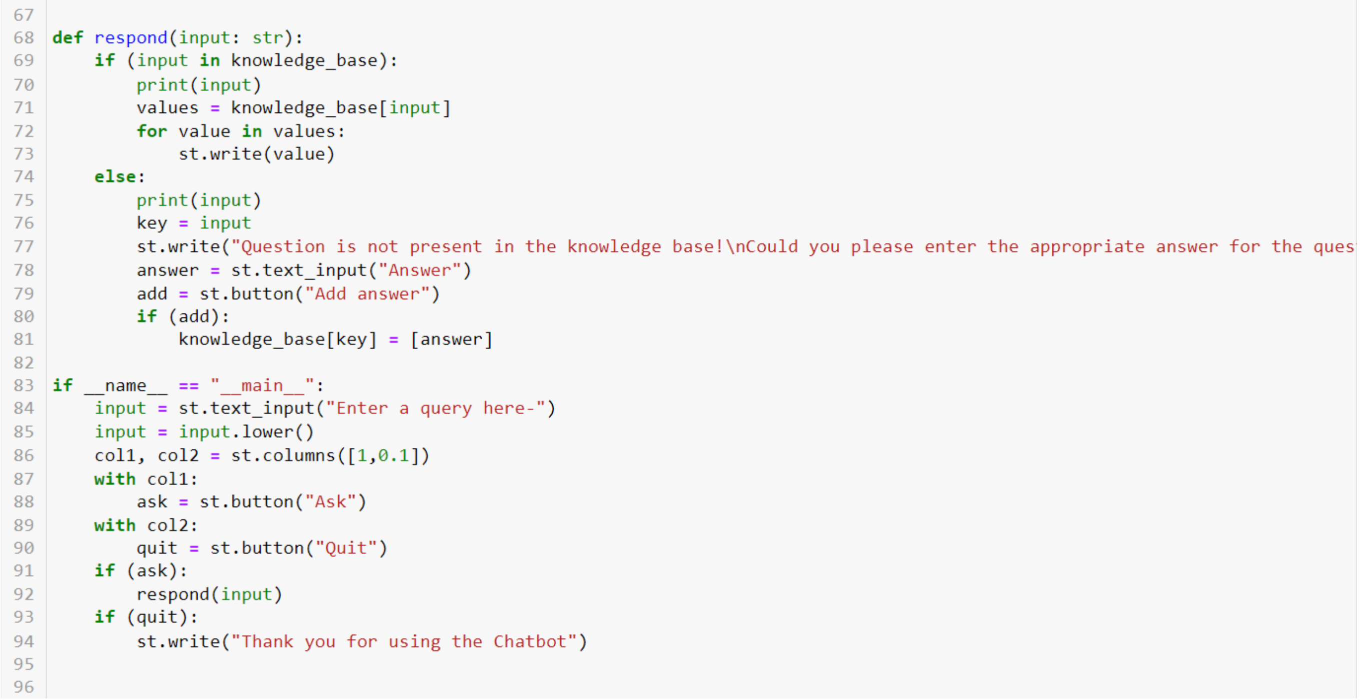


```
Backtracking Solutions:
Solution 1:
- Q - -
- - - Q
Q - - -
- - Q -

Solution 2:
- - Q -
Q - - -
- - - Q
- Q - -

Branch and Bound Solutions:
Solution 1:
- Q - -
- - - Q
Q - - -
- - Q -

Solution 2:
- - Q -
Q - - -
- - - Q
- Q - -
```

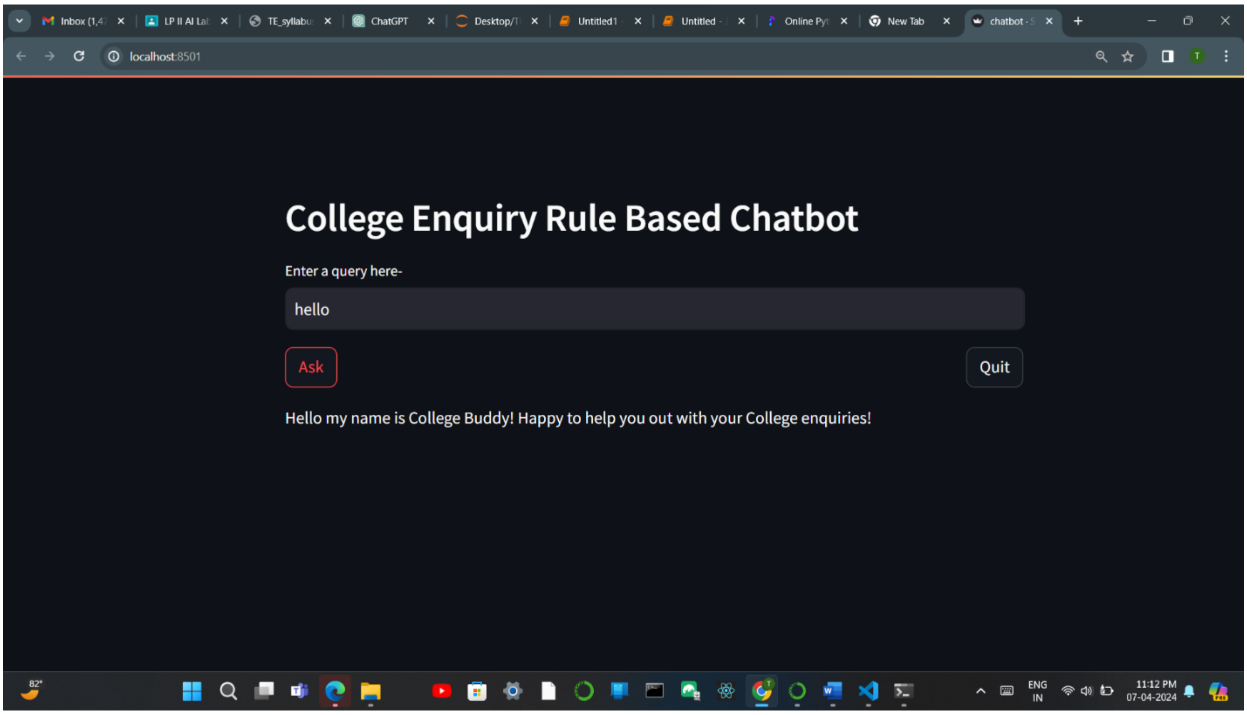
Chatbot code



```python
import streamlit as st

bot_name = "College Buddy"

knowledge_base = {

    "what is your name?" : [
        f"My name is {bot_name}! \n Happy to help you out with your College enquiries!"
    ],

    "hello": [
        f"Hello my name is {bot_name}! \n Happy to help you out with your College enquiries!"
    ],

    "what are the best colleges from pune?": [
        "COEP",
        "PICT",
        "VIT",
        "CUMMINS",
        "PCCOE"
    ],

    "which are the best engineering branches?" : [
        "Computer Engineering",
        "IT Engineering",
        "ENTC Engineering"
    ],

    "what are the top branch cut-offs for coep?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2 percentile",
    ],

    "what are the top branch cut-offs for pict?" : [
        "Computer Engineering : 99.4 percentile",
        "IT Engineering : 98.6 percentile",
        "ENTC Engineering: 97.2 percentile",
    ],

    "what are the top branch cut-offs for vit?" : [
        "Computer Engineering : 99.8 percentile",
        "IT Engineering: 97.1 percentile",
        "ENTC Engineering: 96.2 percentile",
    ],

    "what are the top branch cut-offs for cummins?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "what are the top branch cut-offs for pccoe?" : [
        "Computer Engineering : 99.8 percentile",
        "Does not have IT branch",
        "ENTC Engineering: 99.2",
    ],

    "When do college admissions start?": [
        "Admissions generally start around August",
    ],

}

```



```python

def respond(input: str):
    if (input in knowledge_base):
        print(input)
        values = knowledge_base[input]
        for value in values:
            st.write(value)
    else:
        print(input)
        key = input
        st.write("Question is not present in the knowledge base!\nCould you please enter the appropriate answer for the ques
        answer = st.text_input("Answer")
        add = st.button("Add answer")
        if (add):
            knowledge_base[key] = [answer]

if __name__ == "__main__":
    input = st.text_input("Enter a query here-")
    input = input.lower()
    col1, col2 = st.columns([1,0.1])
    with col1:
        ask = st.button("Ask")
    with col2:
        quit = st.button("Quit")
    if (ask):
        respond(input)
    if (quit):
        st.write("Thank you for using the Chatbot")

```

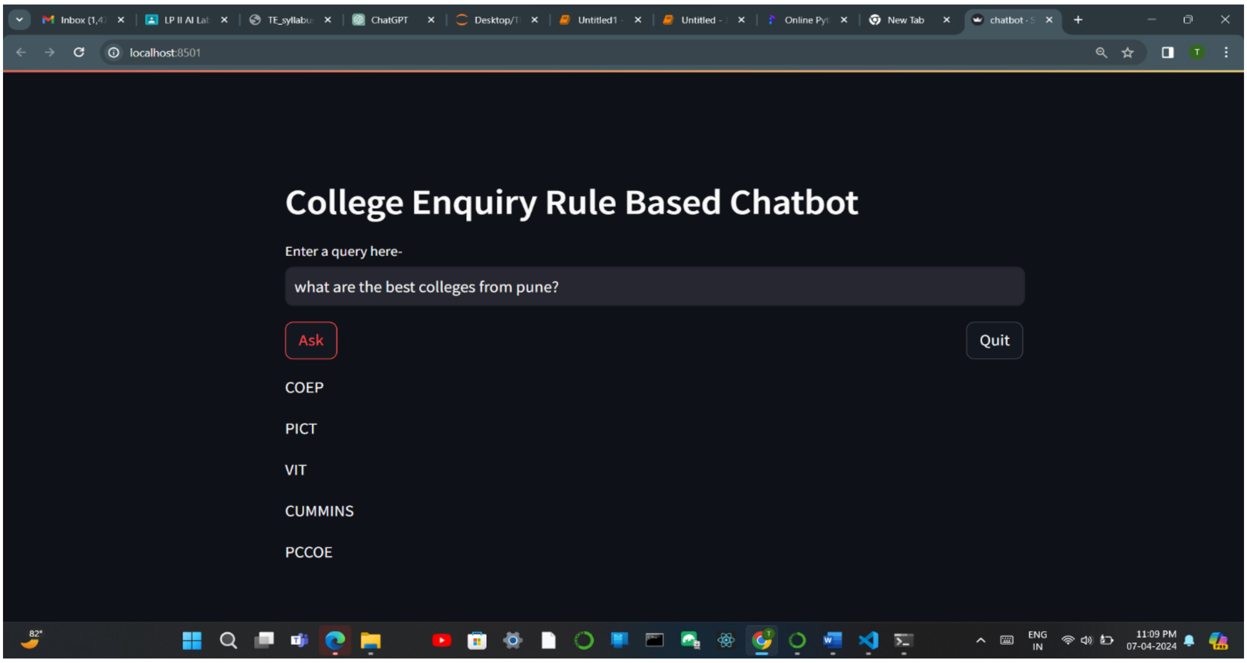


College Enquiry Rule Based Chatbot

Enter a query here-

hello

Hello my name is College Buddy! Happy to help you out with your College enquiries!



College Enquiry Rule Based Chatbot

Enter a query here-

what are the best colleges from pune?

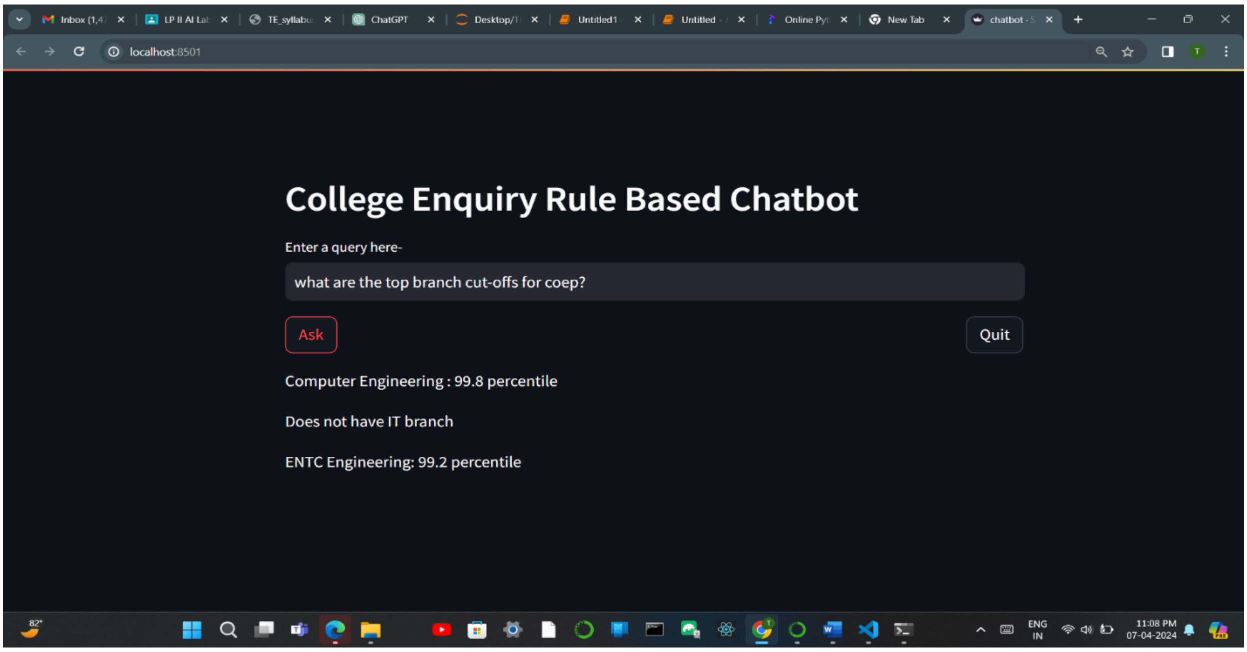
COEP

PICT

VIT

CUMMINS

PCCOE



College Enquiry Rule Based Chatbot

Enter a query here-

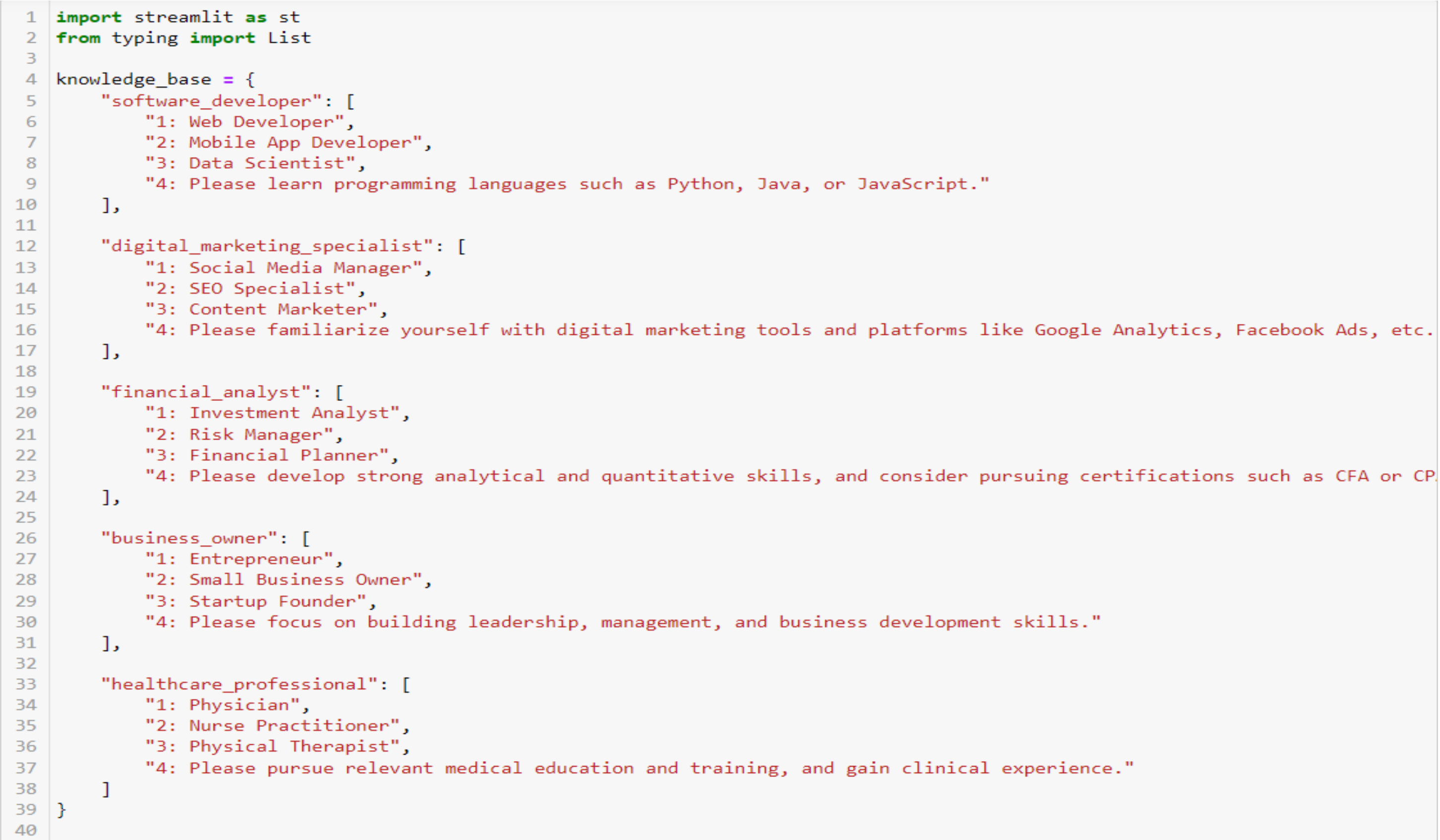
what are the top branch cut-offs for coep?

Computer Engineering : 99.8 percentile

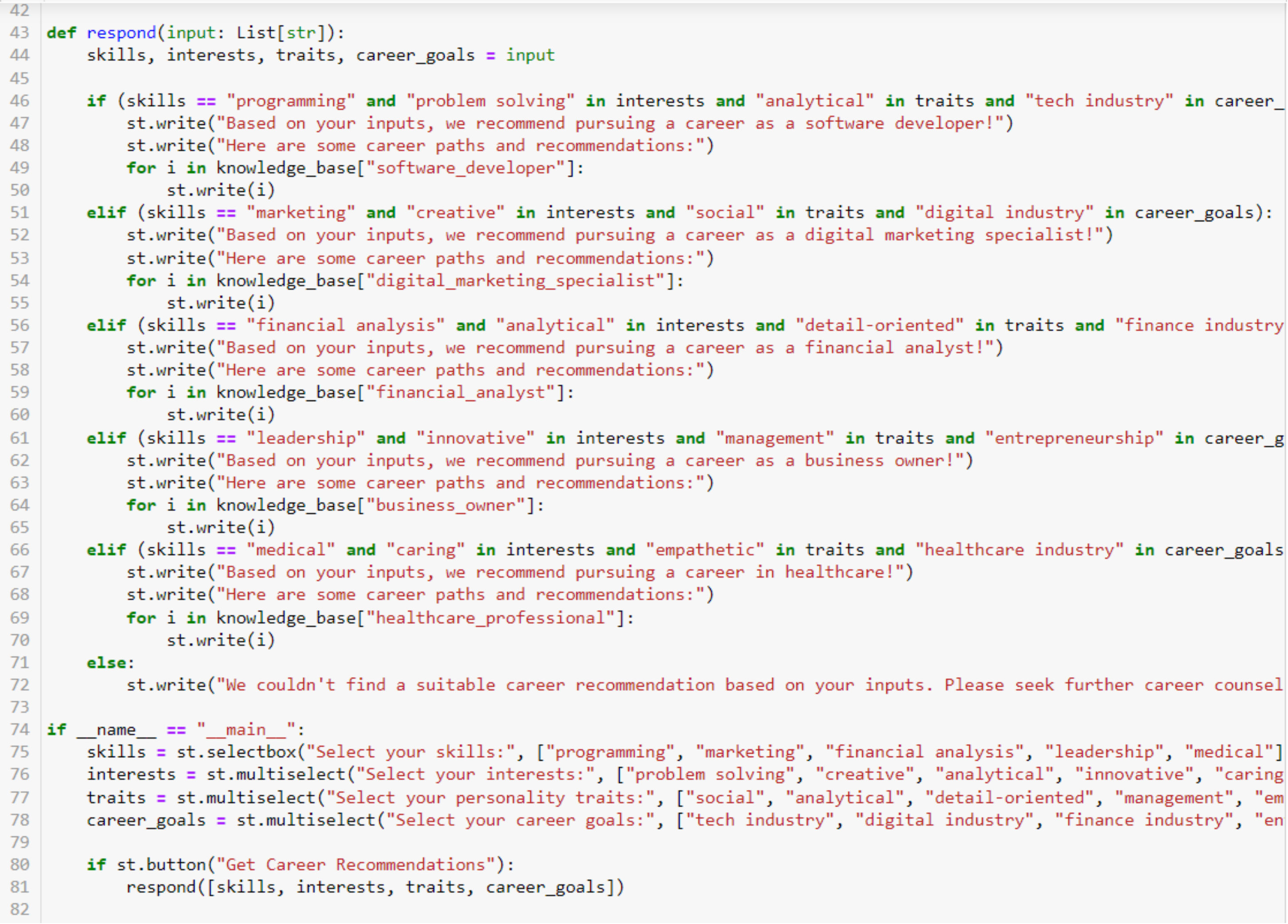
Does not have IT branch

ENTC Engineering: 99.2 percentile

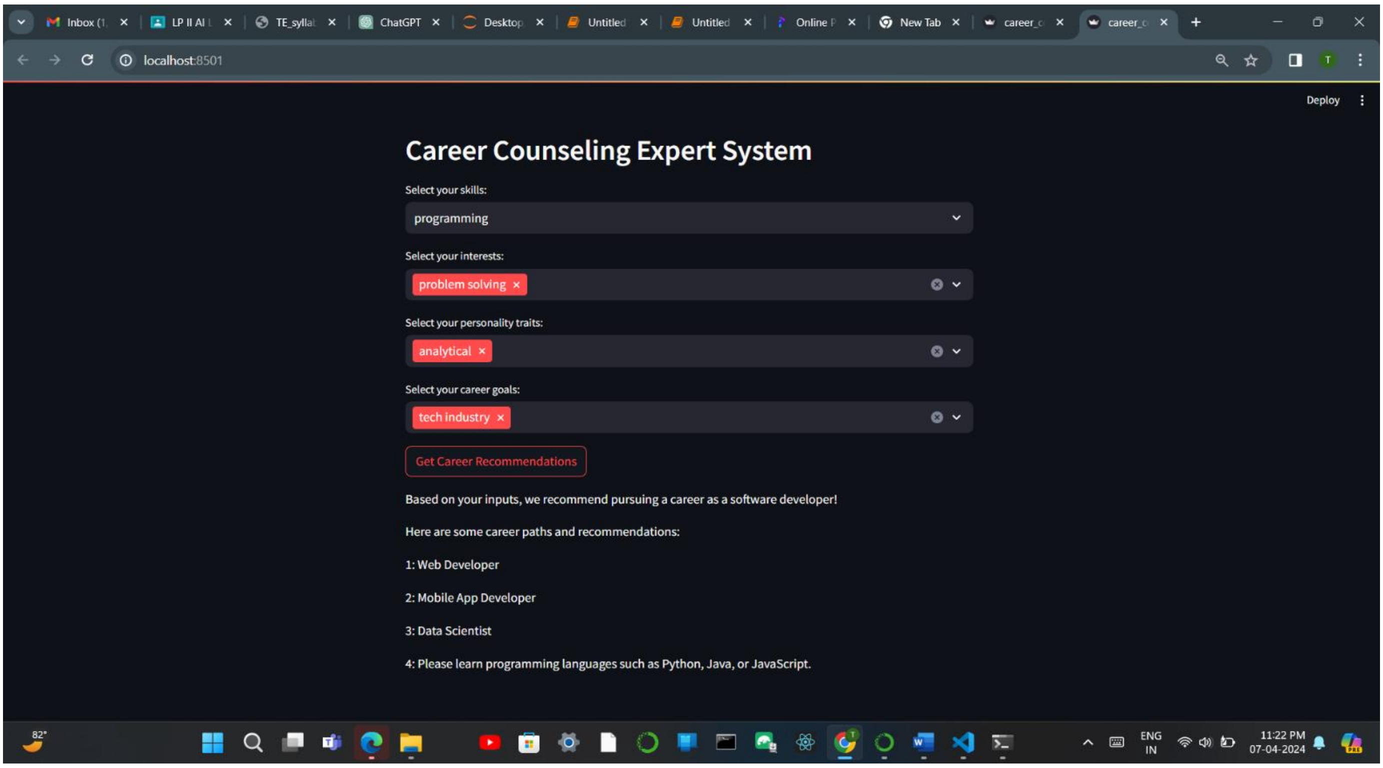
Expert System



```python
import streamlit as st
from typing import List

knowledge_base = {
    "software_developer": [
        "1: Web Developer",
        "2: Mobile App Developer",
        "3: Data Scientist",
        "4: Please learn programming languages such as Python, Java, or JavaScript."
    ],

    "digital_marketing_specialist": [
        "1: Social Media Manager",
        "2: SEO Specialist",
        "3: Content Marketer",
        "4: Please familiarize yourself with digital marketing tools and platforms like Google Analytics, Facebook Ads, etc.
    ],

    "financial_analyst": [
        "1: Investment Analyst",
        "2: Risk Manager",
        "3: Financial Planner",
        "4: Please develop strong analytical and quantitative skills, and consider pursuing certifications such as CFA or CP
    ],

    "business_owner": [
        "1: Entrepreneur",
        "2: Small Business Owner",
        "3: Startup Founder",
        "4: Please focus on building leadership, management, and business development skills."
    ],

    "healthcare_professional": [
        "1: Physician",
        "2: Nurse Practitioner",
        "3: Physical Therapist",
        "4: Please pursue relevant medical education and training, and gain clinical experience."
    ]
}
```



```python
def respond(input: List[str]):
    skills, interests, traits, career_goals = input

    if (skills == "programming" and "problem solving" in interests and "analytical" in traits and "tech industry" in career_
        st.write("Based on your inputs, we recommend pursuing a career as a software developer!")
        st.write("Here are some career paths and recommendations:")
        for i in knowledge_base["software_developer"]:
            st.write(i)
    elif (skills == "marketing" and "creative" in interests and "social" in traits and "digital industry" in career_goals):
        st.write("Based on your inputs, we recommend pursuing a career as a digital marketing specialist!")
        st.write("Here are some career paths and recommendations:")
        for i in knowledge_base["digital_marketing_specialist"]:
            st.write(i)
    elif (skills == "financial analysis" and "analytical" in interests and "detail-oriented" in traits and "finance industry
        st.write("Based on your inputs, we recommend pursuing a career as a financial analyst!")
        st.write("Here are some career paths and recommendations:")
        for i in knowledge_base["financial_analyst"]:
            st.write(i)
    elif (skills == "leadership" and "innovative" in interests and "management" in traits and "entrepreneurship" in career_g
        st.write("Based on your inputs, we recommend pursuing a career as a business owner!")
        st.write("Here are some career paths and recommendations:")
        for i in knowledge_base["business_owner"]:
            st.write(i)
    elif (skills == "medical" and "caring" in interests and "empathetic" in traits and "healthcare industry" in career_goals
        st.write("Based on your inputs, we recommend pursuing a career in healthcare!")
        st.write("Here are some career paths and recommendations:")
        for i in knowledge_base["healthcare_professional"]:
            st.write(i)
    else:
        st.write("We couldn't find a suitable career recommendation based on your inputs. Please seek further career counsel

if __name__ == "__main__":
    skills = st.selectbox("Select your skills:", ["programming", "marketing", "financial analysis", "leadership", "medical"]
    interests = st.multiselect("Select your interests:", ["problem solving", "creative", "analytical", "innovative", "caring
    traits = st.multiselect("Select your personality traits:", ["social", "analytical", "detail-oriented", "management", "em
    career_goals = st.multiselect("Select your career goals:", ["tech industry", "digital industry", "finance industry", "en

    if st.button("Get Career Recommendations"):
        respond([skills, interests, traits, career_goals])
```



Career Counseling Expert System

Select your skills: programming

Select your interests: problem solving

Select your personality traits: analytical

Select your career goals: tech industry

Based on your inputs, we recommend pursuing a career as a software developer!

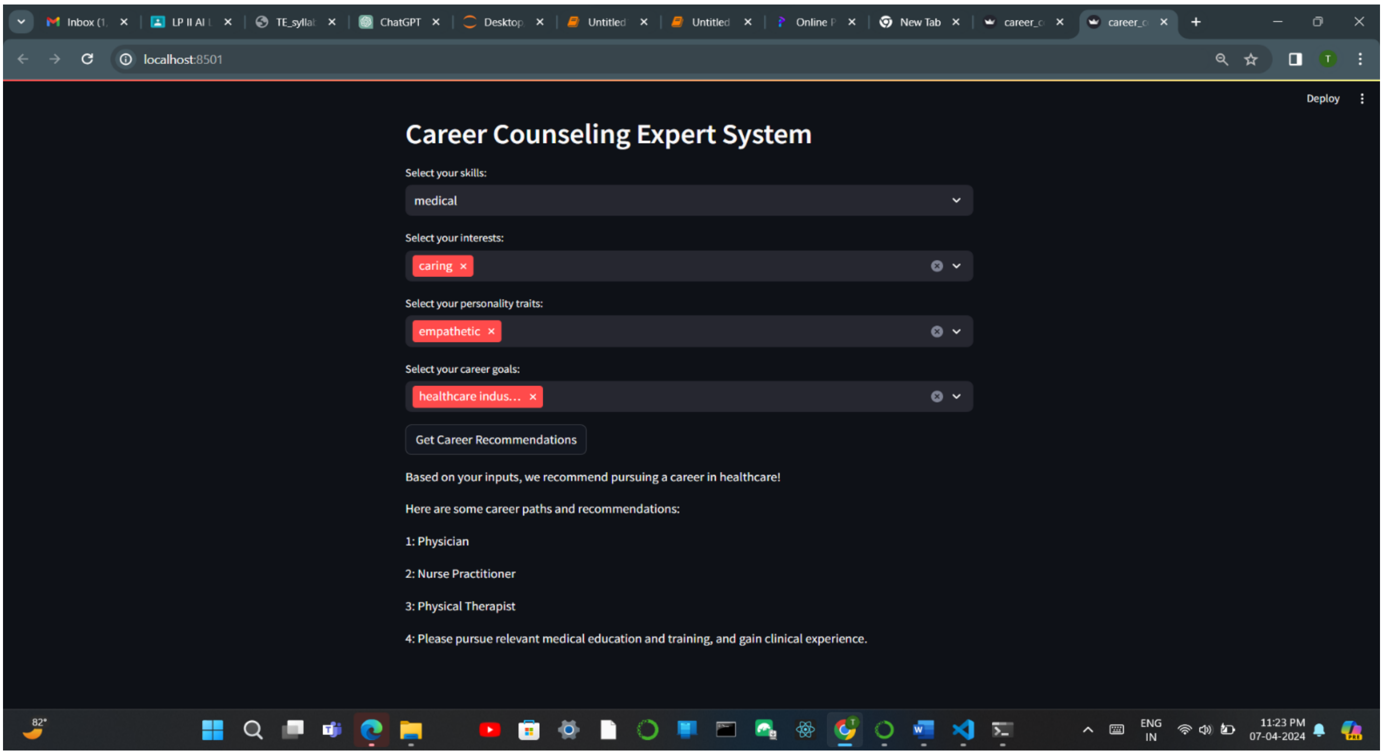
Here are some career paths and recommendations:

1: Web Developer

2: Mobile App Developer

3: Data Scientist

4: Please learn programming languages such as Python, Java, or JavaScript.



Career Counseling Expert System

Select your skills: medical

Select your interests: caring

Select your personality traits: empathetic

Select your career goals: healthcare indus…

Based on your inputs, we recommend pursuing a career in healthcare!

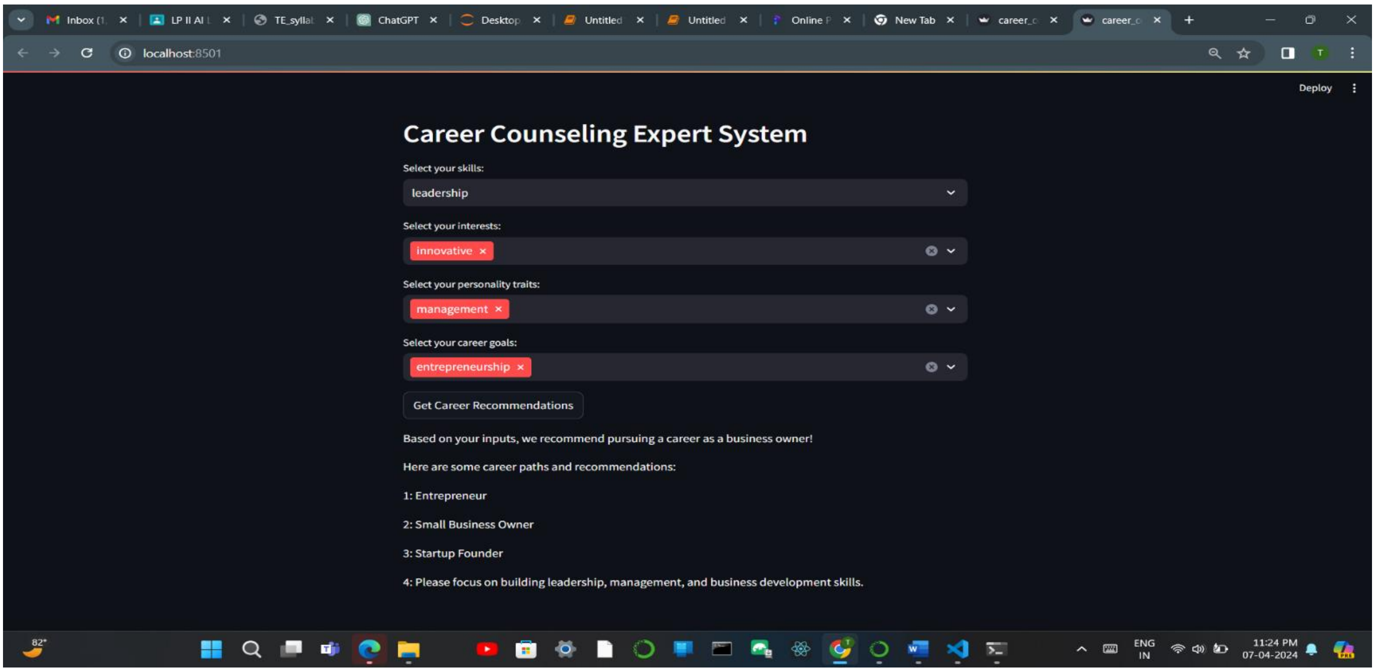
Here are some career paths and recommendations:

1: Physician

2: Nurse Practitioner

3: Physical Therapist

4: Please pursue relevant medical education and training, and gain clinical experience.



Career Counseling Expert System

Select your skills: leadership

Select your interests: innovative

Select your personality traits: management

Select your career goals: entrepreneurship

Based on your inputs, we recommend pursuing a career as a business owner!

Here are some career paths and recommendations:

1: Entrepreneur

2: Small Business Owner

3: Startup Founder

4: Please focus on building leadership, management, and business development skills.