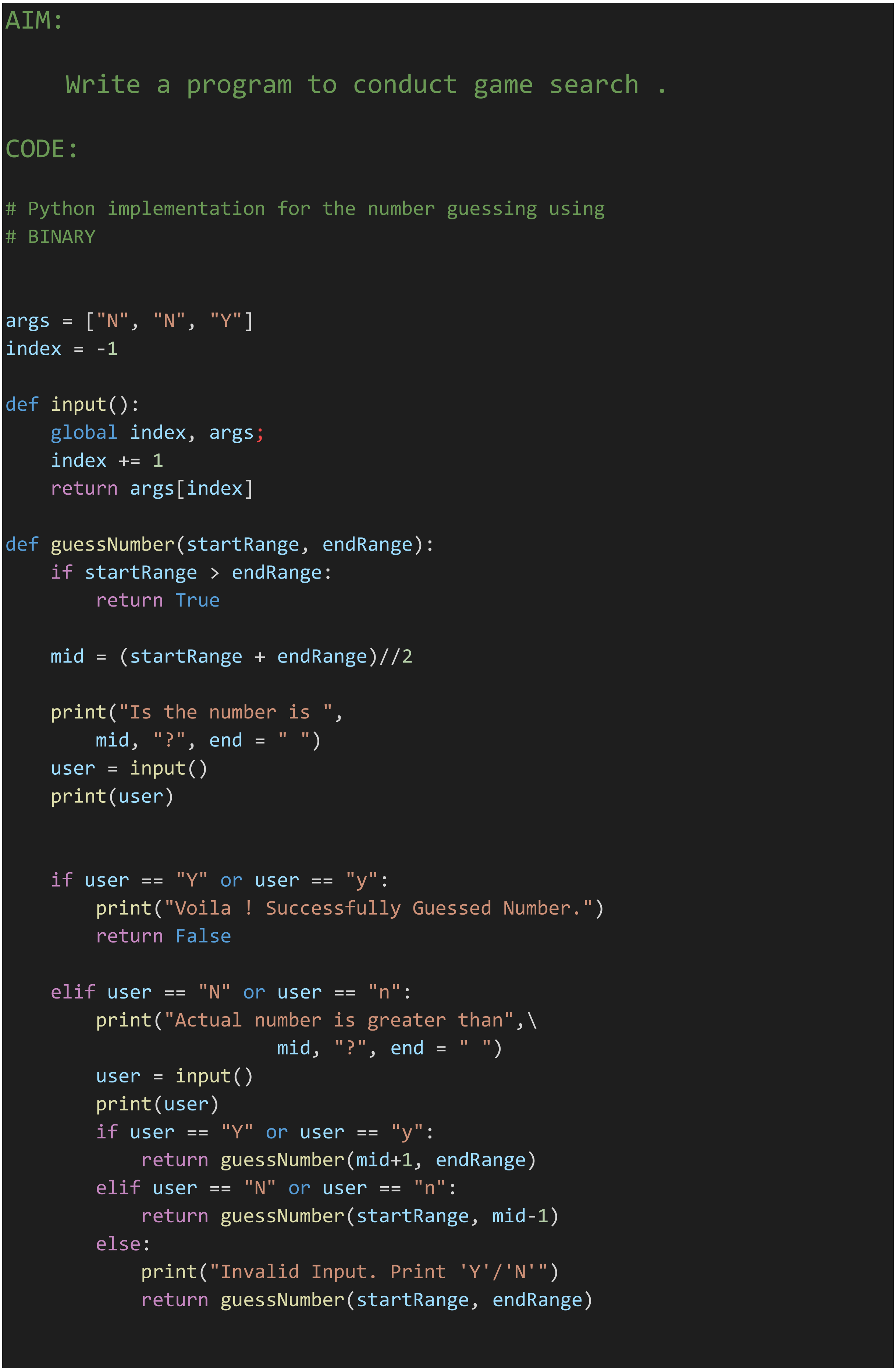
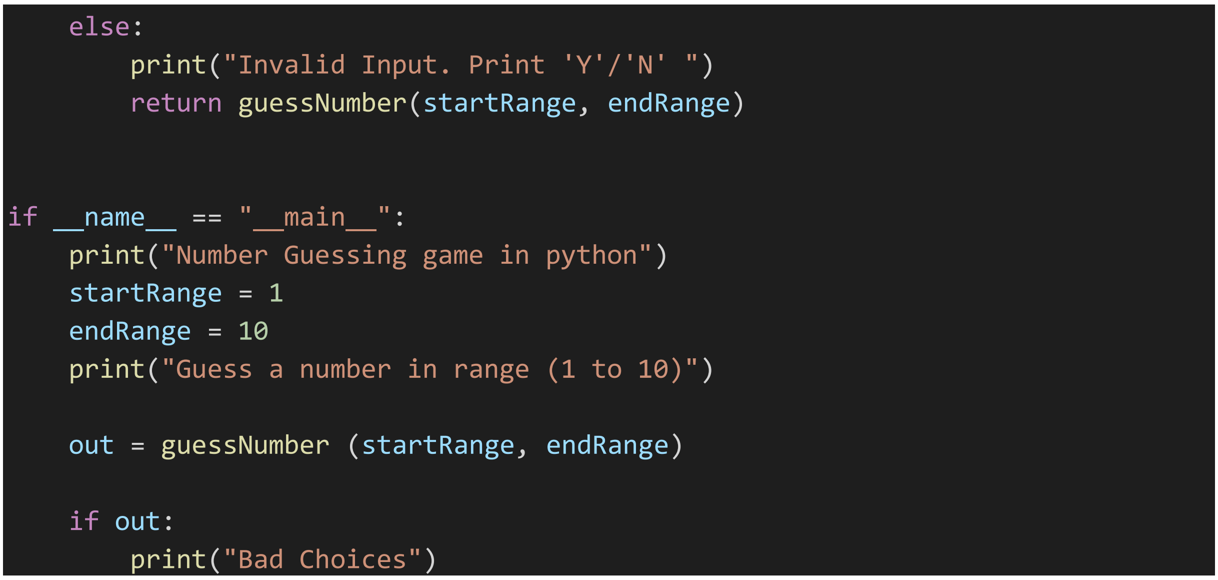


AIM:

Write a program to conduct game search .

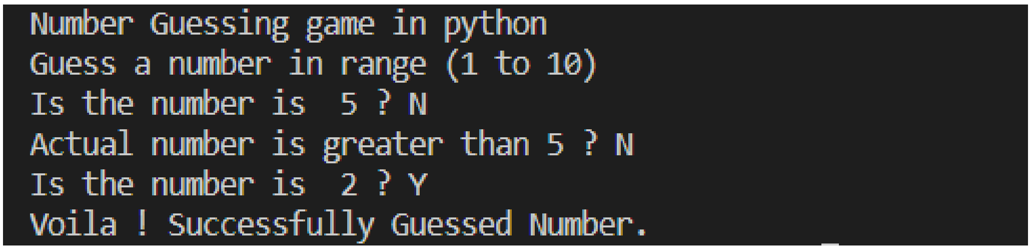
CODE:

```
# Python implementation for the number guessing using
# BINARY


args = ["N", "N", "Y"]
index = -1

def input():
    global index, args;
    index += 1
    return args[index]

def guessNumber(startRange, endRange):
    if startRange > endRange:
        return True

    mid = (startRange + endRange)//2

    print("Is the number is ",
        mid, "?", end = " ")
    user = input()
    print(user)


    if user == "Y" or user == "y":
        print("Voila ! Successfully Guessed Number.")
        return False

    elif user == "N" or user == "n":
        print("Actual number is greater than",\
                        mid, "?", end = " ")
        user = input()
        print(user)
        if user == "Y" or user == "y":
            return guessNumber(mid+1, endRange)
        elif user == "N" or user == "n":
            return guessNumber(startRange, mid-1)
        else:
            print("Invalid Input. Print 'Y'/'N'")
            return guessNumber(startRange, endRange)
```



```
    else:
        print("Invalid Input. Print 'Y'/'N' ")
        return guessNumber(startRange, endRange)


if __name__ == "__main__":
    print("Number Guessing game in python")
    startRange = 1
    endRange = 10
    print("Guess a number in range (1 to 10)")

    out = guessNumber (startRange, endRange)

    if out:
        print("Bad Choices")
```

Output:

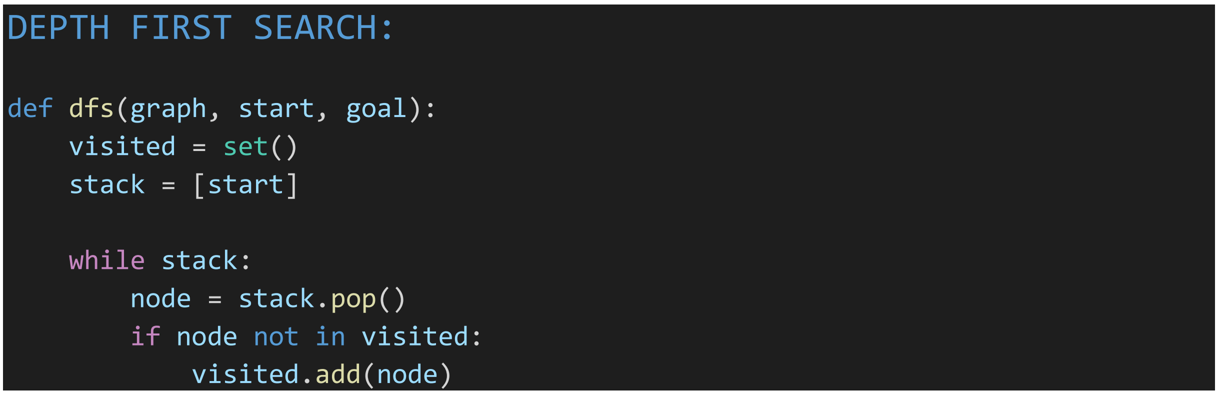


```
Number Guessing game in python
Guess a number in range (1 to 10)
Is the number is  5 ? N
Actual number is greater than 5 ? N
Is the number is  2 ? Y
Voila ! Successfully Guessed Number.
```

QUESTION: 14

AIM:

WAP a program to conduct uninformed search.

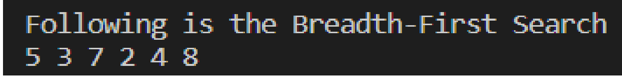


```
DEPTH FIRST SEARCH:

def dfs(graph, start, goal):
    visited = set()
    stack = [start]

    while stack:
        node = stack.pop()
        if node not in visited:
            visited.add(node)
            if node == goal:
                return
            for neighbor in graph[node]:
                if neighbor not in visited:
                    stack.append(neighbor)
```

BREADTH FIRST SEARCH:

```
graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}

visited = []
queue = []

def bfs(visited, graph, node):
  visited.append(node)
  queue.append(node)

  while queue:
    m = queue.pop(0)
    print (m, end = " ")

    for neighbour in graph[m]:
      if neighbour not in visited:
        visited.append(neighbour)
        queue.append(neighbour)


print("Following is the Breadth-First Search")
bfs(visited, graph, '5')
```

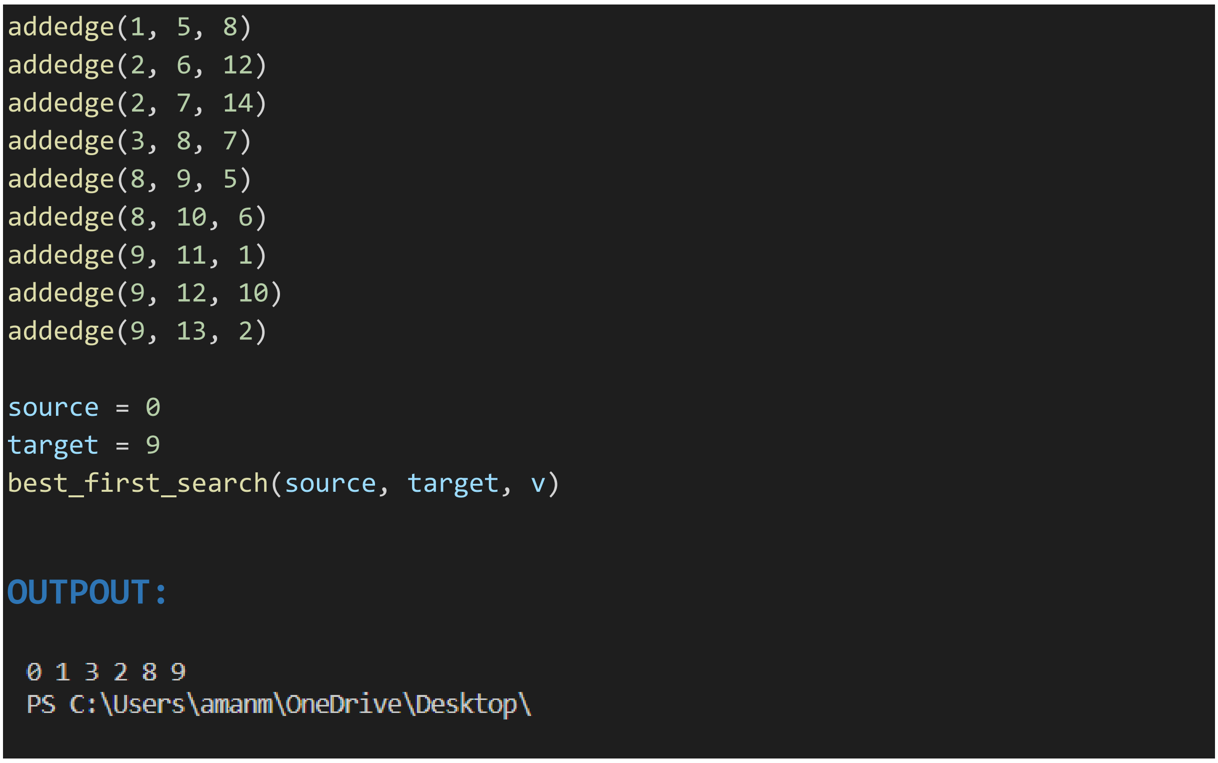
OUTPUT:



```
Following is the Breadth-First Search
5 3 7 2 4 8
```

BEST FIRST SEARCH:

```
from queue import PriorityQueue
v = 14
graph = [[] for i in range(v)]




def best_first_search(actual_Src, target, n):
    visited = [False] * n
    pq = PriorityQueue()
    pq.put((0, actual_Src))
    visited[actual_Src] = True

    while pq.empty() == False:
        u = pq.get()[1]

        print(u, end=" ")
        if u == target:
            break

        for v, c in graph[u]:
            if visited[v] == False:
                visited[v] = True
                pq.put((c, v))
    print()




def addedge(x, y, cost):
    graph[x].append((y, cost))
    graph[y].append((x, cost))



addedge(0, 1, 3)
addedge(0, 2, 6)
addedge(0, 3, 5)
addedge(1, 4, 9)
```



```
addedge(1, 5, 8)
addedge(2, 6, 12)
addedge(2, 7, 14)
addedge(3, 8, 7)
addedge(8, 9, 5)
addedge(8, 10, 6)
addedge(9, 11, 1)
addedge(9, 12, 10)
addedge(9, 13, 2)

source = 0
target = 9
best_first_search(source, target, v)
```

OUTPOUT:

```
 0 1 3 2 8 9
 PS C:\Users\amanm\OneDrive\Desktop\
```