Job No: 01

Job Name: **Write a program to implement a simple chatbot.**

Theory:

A chatbot is a computer program designed to interact with humans in a conversational manner.It can be used to answer questions, provide information, or assist with tasks. A chatbot takes user input, typically in the form of text, and generates an appropriate response.

Code:

```
import time
now = time.ctime()
keyword_responses = {
    "hello": "Hello!",
    "hi": "Hello!",
    "how are you": "I'm doing well, thank you.",
    "good": "Great!",
    "bad": "I'm sorry to hear that.",
    "bye": "Goodbye!",
    "see you later": "Goodbye!",
    "have a nice day": "Thank you, you too!",
    "what's your name?":"piku",
    "what's the time now?": now,
    "who made you?": "Human",
}

def generate_response(user_input):
    for keyword in keyword_responses:
        if user_input.lower() in keyword or keyword in
user_input.lower():
            return keyword_responses[keyword]

    return "I'm sorry, I didn't understand that."

while True:
    user_input = input("You: ").strip()
```

```
    if user_input == "":
        continue

    response = generate_response(user_input)

    if response in keyword_responses.values():
        print(response)
    else:
        print("I'm sorry, I didn't understand that.")
```

Input/Output:

```
You: hi
Hello!
You: have a nice day
Thank you, you too!
You: bye
Goodbye!
You:
```

Job No: 02

Job Name: **Write a program for Breadth First Search algorithm.**

Theory: Breadth First Search (BFS) is a tree traversal algorithm used to search a tree or graph in a breadthward motion.The algorithm starts at a specific node (often referred to as the "root" node) and systematically visits all the neighboring nodes at the same level before moving to the next level.

Code:

```
class Node:
    def __init__(self,n_value):
        self.value = n_value
        self.children = []

def bfs(root):
    queue = [root]
    result = []

    while queue:
        node = queue.pop(0)
        result.append(node.value)
        for child in node.children:
            queue.append(child)
    return result

def build_tree():
    root_value = input("Enter the root node value: ")
    root = Node(root_value)
    queue = [root]

    while queue:
        node = queue.pop(0)

        child_Num = int(input(f"Enter the number of
children for node {node.value}(or 0 if it has no children):
"))
        for k in range(child_Num):
            child_value = input(f"Enter the value for
child {k+1} of node {node.value}: ")
            child_node = Node(child_value)
```

```
            node.children.append(child_node)
            queue.append(child_node)

    return root

root = build_tree()
result = bfs(root)
print(f"The result of BFS Search for given tree is :
{result}")
```

Input/Output:

```
Enter the root node value: A
Enter the number of children for node A(or 0 if it has no children): 2
Enter the value for child 1 of node A: B
Enter the value for child 2 of node A: C
Enter the number of children for node B(or 0 if it has no children): 2
Enter the value for child 1 of node B: D
Enter the value for child 2 of node B: E
Enter the number of children for node C(or 0 if it has no children): 2
Enter the value for child 1 of node C: F
Enter the value for child 2 of node C: G
Enter the number of children for node D(or 0 if it has no children): 0
Enter the number of children for node E(or 0 if it has no children): 0
Enter the number of children for node F(or 0 if it has no children): 0
Enter the number of children for node G(or 0 if it has no children): 0
The result of BFS Search for given tree is : ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```

Job No: 03

Job Name: **Write a program for Depth First Search algorithm.**

Theory: Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node and explores as far as possible along each branch before backtracking.

Code:

```
class Node:
    def __init__(self, value):
        self.value = value
        self.children = []

def depth_first_search(node):
    for child in node.children:
        depth_first_search(child)

    print(node.value, end=" ")

def build_tree():
    root_value = input("Enter the root node value: ")
    root = Node(root_value)
    queue = [root]

    while queue:
        node = queue.pop(0)

        num_children = int(input(f"Enter the number of
children for node {node.value} (or 0 if it has no children):
"))

        for i in range(num_children):
            child_value = input(f"Enter the value for
child {i+1} of node {node.value}: ")
            child_node = Node(child_value)
            node.children.append(child_node)
            queue.append(child_node)
```

```
    return root

root = build_tree()
print("The DFS is:")
depth_first_search(root)
```

Input/Output:

```
Enter the root node value: A
Enter the number of children for node A (or 0 if it has no children): 2
Enter the value for child 1 of node A: B
Enter the value for child 2 of node A: C
Enter the number of children for node B (or 0 if it has no children): 2
Enter the value for child 1 of node B: D
Enter the value for child 2 of node B: E
Enter the number of children for node C (or 0 if it has no children): 2
Enter the value for child 1 of node C: F
Enter the value for child 2 of node C: G
Enter the number of children for node D (or 0 if it has no children): 0
Enter the number of children for node E (or 0 if it has no children): 0
Enter the number of children for node F (or 0 if it has no children): 0
Enter the number of children for node G (or 0 if it has no children): 0
The DFS is:
D E B F G C A
```