

## AI Assisted Coding

### Lab 11.2 – Data Structures with AI

2303A51209

Batch 04

#### Objective:

To use AI-assisted coding to design and implement fundamental data structures in Python including Stack, Queue, Linked List, Binary Search Tree, and Hash Table.

#### Task 1: Stack Implementation

##### Prompt:

"Using AI-assisted coding, design and implement a Stack data structure in Python with push, pop, peek, and isEmpty operations."

##### Code:

```
class Stack:  
    def __init__(self):  
        self.stack = []  
  
    def push(self, item):  
        self.stack.append(item)  
  
    def pop(self):  
        if self.isEmpty():  
            return "Stack is empty"  
        else:  
            return self.stack.pop()
```

```
return self.stack.pop()
```

```
def peek(self):  
    if self.isEmpty():  
        return "Stack is empty"  
    return self.stack[-1]
```

```
def isEmpty(self):  
    return len(self.stack) == 0
```

### Explanation:

This program implements Stack using list with LIFO principle.

### Output:

Top element: 20

Popped element: 20

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows a file named "11.2.py".
- Code Editor:** Displays Python code for a stack class. The code includes methods for pushing items onto the stack, popping items off, peeking at the top item, and checking if the stack is empty. It also contains a section for testing the stack.
- Terminal:** Shows the output of running the script "11.2.py". The output is:

```
PS F:\AI-A-coding> & C:/Python314/python.exe F:/AI-A-coding/11.2.py
10 pushed to stack
20 pushed to stack
Top element: 20
Popped element: 20
Is stack empty? False
> PS F:\AI-A-coding>
```
- Bottom Bar:** Shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is currently selected.

## Task 2: Queue Implementation

### Prompt:

"Design a Queue using FIFO principle with enqueue, dequeue, front and size operations."

### Code:

```
class Queue:

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

    def enqueue(self, item):
        self.queue.append(item)

    def dequeue(self):
        if len(self.queue) <= 0:
            return None
        else:
            return self.queue.pop(0)

    def front(self):
        if len(self.queue) <= 0:
            return None
        else:
            return self.queue[0]

    def size(self):
        return len(self.queue)
```

```
def dequeue(self):
    if self.isEmpty():
        return "Queue is empty"
    return self.queue.pop(0)
```

```
def front(self):
    return self.queue[0]
```

```
def size(self):
    return len(self.queue)
```

```
def isEmpty(self):
    return len(self.queue) == 0
```

### Explanation:

Queue follows FIFO  
principle.

### Output:

Front element: 5  
Dequeued element: 5

```
VS Code - A Coding > C:/pythons/pypython/11.2-A-Coding/11.2.py
• 5 added to queue
• 15 added to queue
Front element: 5
Dequeued element: 5
Queue size: 1
PS F:\A-Coding>
```

The screenshot shows a code editor with a file named '11.2.py'. The code defines a Queue class with methods for front, size, and isEmpty. It then creates a Queue object, adds two elements (5 and 15), prints the front element, dequeues it, and prints the queue size. The terminal below shows the execution and output of the script.

## Task 3: Singly Linked List

**Prompt:**

"Create singly linked list with insertion and traversal."

**Code:**

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None
```

```
class LinkedList:  
    def __init__(self):  
        self.head=None  
  
    def insert(self,data):  
        new_node=Node(data)  
        if self.head is None:  
            self.head=new_node  
        else:  
            temp=self.head  
            while temp.next:  
                temp=temp.next  
            temp.next=new_node
```

```
    def display(self):  
        temp=self.head  
        while temp:  
            print(temp.data,end="->")  
            temp=temp.next
```

### Explanation:

Linked list connects nodes using  
pointers.

### Output:

10->20->30->None

The screenshot shows a terminal window with two tabs: "Welcome" and "11.2.py". The "11.2.py" tab contains the following Python code:

```
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.next = None
5
6 class LinkedList:
7     def __init__(self):
8         self.head = None
9
10    def insert(self, data):
11        new_node = Node(data)
12        if self.head is None:
13            self.head = new_node
14        else:
15            temp = self.head
16            while temp.next:
17                temp = temp.next
18            temp.next = new_node
19
20    def display(self):
21        temp = self.head
22        while temp:
23            print(temp.data, end=" -> ")
24            temp = temp.next
25        print("None")
```

Below the code, the terminal shows the command and its output:

```
● PS F:\AI-A-coding> & C:/Python314/python.exe f:/AI-A-coding/11.2.py
10 -> 20 -> 30 -> None
○ PS F:\AI-A-coding>
```

## Task 4: Binary Search Tree

Prompt:

"Implement BST with insertion and inorder traversal."

Code:

```
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key
```

```
def insert(root, key):
```

```
    if root is None:
```

```
return Node(key)

if key<root.val:
    root.left=insert(root.left,key)

else:

    root.right=insert(root.right,key)

return root
```

```
def inorder(root):

    if root:

        inorder(root.left)

        print(root.val,end=" ")

        inorder(root.right)
```

### Explanation:

BST stores values in sorted order.

### Output:

20 30 40 50 60 70 80

```

11.2.py >_
1   class Node:
2       def __init__(self, key):
3           self.left = None
4           self.right = None
5           self.val = key
6
7       def insert(self, key):
8           if self is None:
9               return Node(key)
10          if key < self.val:
11              self.left = insert(self.left, key)
12          else:
13              self.right = insert(self.right, key)
14          return self
15
16      def inorder(self):
17          if self:
18              inorder(self.left)
19              print(self.val, end=" ")
20              inorder(self.right)
21
22
23  # Testing
24 root = None
25 elements = [50, 30, 20, 40, 70, 60, 80]
26 for e in elements:

```

● PS F:\AI-A-coding> & C:/Python314/python.exe f:/AI-A-coding/11.2.py  
 Inorder traversal:  
 20 30 40 50 60 70 80  
 ○ PS F:\AI-A-coding>

## Task 5: Hash Table

**Prompt:**

Create Hash Table with collision handling using chaining.

**Code:**

```

class HashTable:

def __init__(self,size):
    self.size=size
    self.table=[[] for _ in range(size)]

def hashFunction(self,key):
    return key%self.size
    def insert(self,key,value):
        index=self.hashFunction(key)

```

```
self.table[index].append((key,value))
```

```
def search(self,key):  
    index=self.hashFunction(key)  
    for pair in self.table[index]:  
        if pair[0]==key:  
            return pair[1]  
    return "Not Found"
```

```
def delete(self,key):  
    index=self.hashFunction(key)  
    for pair in self.table[index]:  
        if pair[0]==key:  
            self.table[index].remove(pair)  
    return "Deleted"  
    return "Not Found"
```

### Explanation:

Hash table stores key-value pairs using hashing with chaining.

### Output:

Apple

Deleted

Not Found

The screenshot shows a code editor with two tabs: 'Welcome' and '11.2.py'. The '11.2.py' tab contains the following Python code:

```
1  class HashTable:
2      def __init__(self, size):
3          self.size = size
4          self.table = [[] for _ in range(size)]
5
6      def hashFunction(self, key):
7          return key % self.size
8
9      def insert(self, key, value):
10         index = self.hashFunction(key)
11         self.table[index].append((key, value))
12
13     def search(self, key):
14         index = self.hashFunction(key)
15         for pair in self.table[index]:
16             if pair[0] == key:
17                 return pair[1]
18         return "Not Found"
19
20     def delete(self, key):
21         index = self.hashFunction(key)
22         for pair in self.table[index]:
23             if pair[0] == key:
24                 self.table[index].remove(pair)
25                 return "Deleted"
26         return "Not Found"
27
28
29 # Testing
30 ht = HashTable(5)
31 ht.insert(1, "Apple")
32 ht.insert(6, "Banana")
33 print(ht.search(1))
34 print(ht.delete(6))
35 print(ht.search(6))
```

Below the code editor, a terminal window shows the execution of the script:

```
PS F:\AI-A-coding> & C:/Python314/python.exe f:/AI-A-coding/11.2.py
Apple
Deleted
Not Found
PS F:\AI-A-coding>
```

## Conclusion:

AI-assisted coding simplifies implementation of data structures and improves understanding of algorithms and logic.