

# 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"
```

```
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

```
11.2.py > ...
1 class Stack:
2     def __init__(self):
3         self.stack = []
4
5     def push(self, item):
6         self.stack.append(item)
7         print(item, "pushed to stack")
8
9     def pop(self):
10        if self.isEmpty():
11            return "stack is empty"
12        return self.stack.pop()
13
14    def peek(self):
15        if self.isEmpty():
16            return "stack is empty"
17        return self.stack[-1]
18
19    def isEmpty(self):
20        return len(self.stack) == 0
21
22
23 # Testing
24 s = Stack()
25 s.push(10)
26 s.push(20)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
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>
```

## 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 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

```
Welcome 11.2.py X
11.2.py > ...
1 class Queue:
13
14     def front(self):
15         if self.isEmpty():
16             return "Queue is empty"
17         return self.queue[0]
18
19     def size(self):
20         return len(self.queue)
21
22     def isEmpty(self):
23         return len(self.queue) == 0
24
25
26 # Testing
27 q = Queue()
28 q.enqueue(5)
29 q.enqueue(15)
30 print("Front element:", q.front())
31 print("Dequeued element:", q.dequeue())
32 print("Queue size:", q.size())

PS F:\AI-A-coding> C:\Python34\python.exe F:\AI-A-coding\11.2.py
5 added to queue
15 added to queue
Front element: 5
Dequeued element: 5
Queue size: 1
PS F:\AI-A-coding>
```

### 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
```

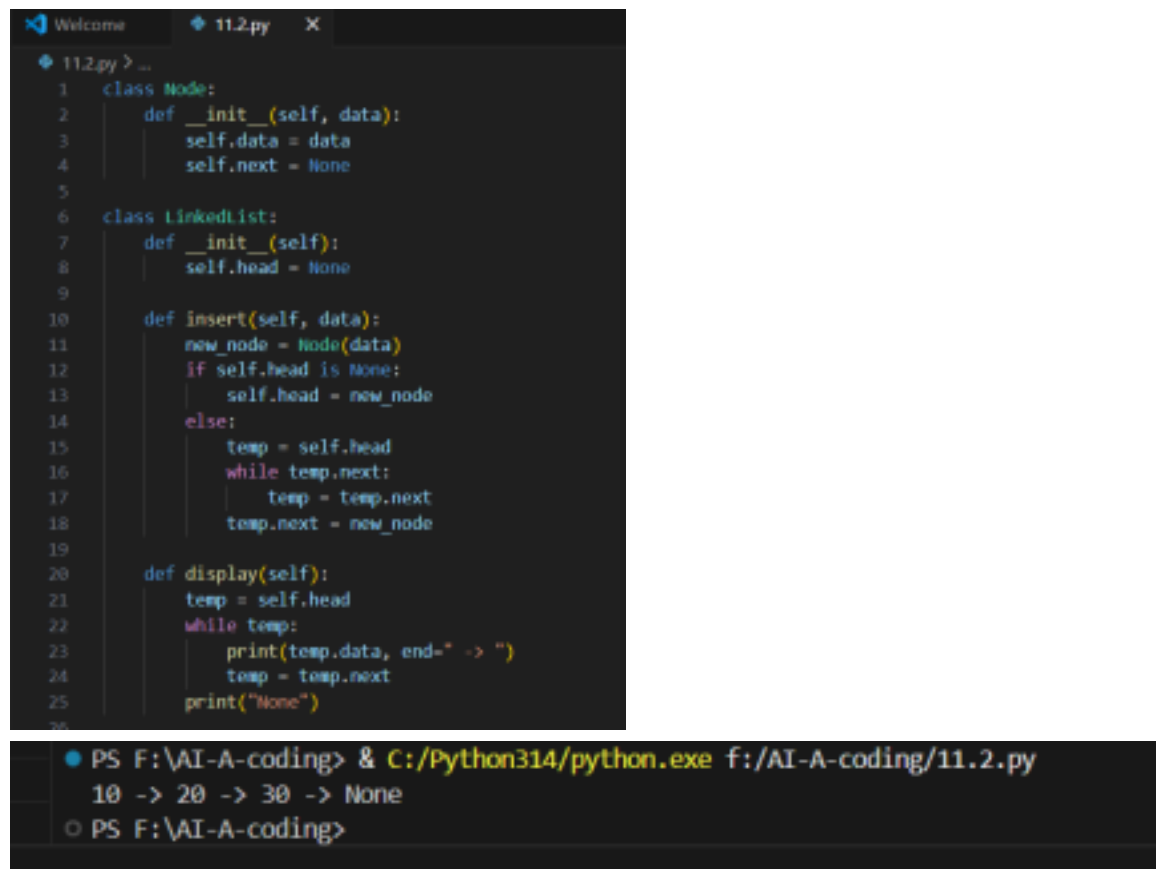
```
    def display(self):
        temp=self.head
        while temp:
            print(temp.data,end="->")
            temp=temp.next
```

### Explanation:

Linked list connects nodes using pointers.

### Output:

10->20->30->None



```
11.2.py > ...
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")
26
```

```
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(root, key):
8         if root is None:
9             return Node(key)
10        if key < root.val:
11            root.left = insert(root.left, key)
12        else:
13            root.right = insert(root.right, key)
14        return root
15
16    def inorder(root):
17        if root:
18            inorder(root.left)
19            print(root.val, end=" ")
20            inorder(root.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

```

Welcome 11.2.py
11.2.py > ...
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))

```

```

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.