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
```

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:

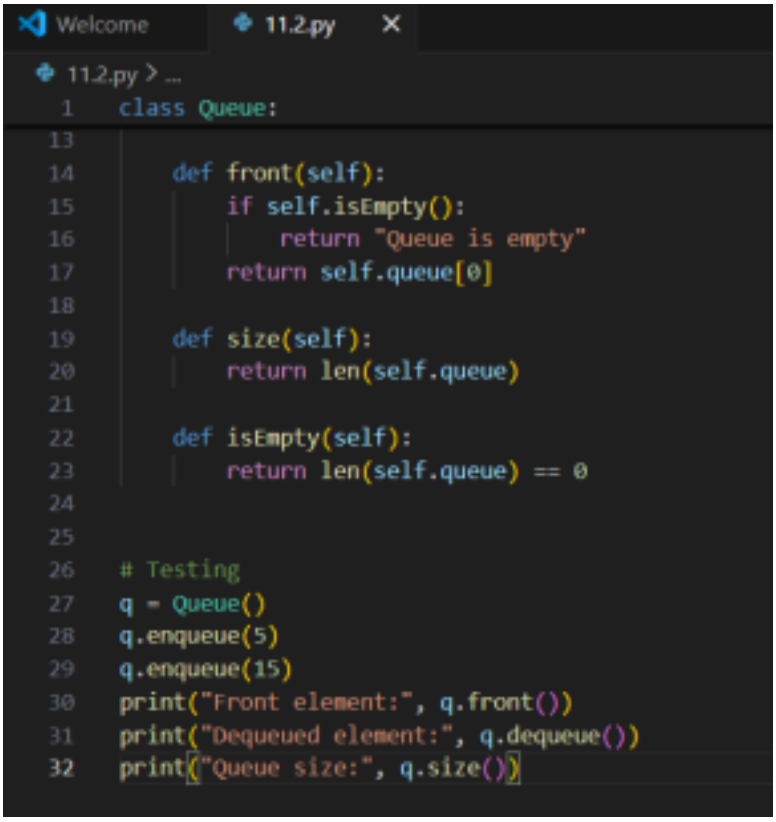
Queue follows FIFO

principle.

Output:

Front element: 5

Dequeued element: 5



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

```python
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
        print("None")
```

```
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
```

```python
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)


# Testing
root = None
elements = [50, 30, 20, 40, 70, 60, 80]
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

```python
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"


# Testing
ht = HashTable(5)
ht.insert(1, "Apple")
ht.insert(6, "Banana")
print(ht.search(1))
print(ht.delete(6))
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.