

AI ASSISTED CODING

LAB ASSIGNMENT: 11.2

NAME:G.Jeevan

HTNO:2403A52055

Batch No:-03

TASK: Stack Implementation

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Sample Input Code: class Stack: pass.

PROMPT:

Generate python code and stack
Implementation

Task: Use AI to generate a Stack class with push, pop, peek, and is_empty
methods. Sample

Input Code: class
Stack: pass.

CODE & OUTPUT:

The image shows two side-by-side terminal windows in Microsoft Visual Studio Code, both running Python 3.11.0. The left terminal displays the implementation of a Stack class, while the right terminal shows the execution of this code.

Left Terminal (Python 3.11.0):

```
PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windowsapps/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/iinternetcache/ie/jbooxdla/ai/lab 11.1.py"
PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windowsapps/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/iinternetcache/ie/jbooxdla/ai/lab 11.1.py"
20
20
False
10
True
PS C:\Users\kadal>
```

Right Terminal (Python 3.11.0):

```
PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windowsapps/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/iinternetcache/ie/jbooxdla/ai/lab 11.1.py"
PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windowsapps/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/iinternetcache/ie/jbooxdla/ai/lab 11.1.py"
20
20
False
10
True
PS C:\Users\kadal>
```

EXPLAINATION:

A **stack** is a linear data structure that follows the **LIFO** principle — **Last In, First Out**. Think of it like a stack of plates:

- You add (push) a plate to the top.
- You remove (pop) the top plate first.
- You can peek at the top plate without removing it.
- You can check if the stack is empty.

TASK 2:

Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

```
class Queue: pass.
```

PROMPT:

Generate python code and queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

```
class Queue: pass.
```

CODE & OUTPUT:

The screenshot shows the Microsoft Visual Studio Code interface. The left sidebar indicates "NO FOLDER OPENED". The main editor window displays a Python script named `AI LAB 11.1 T2.py` containing the following code:

```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def enqueue(self, item):
6          """Add an item to the end of the queue."""
7          self.items.append(item)
8
9      def dequeue(self):
10         """Remove and return the front item of the queue. Raises IndexError if empty."""
11         if self.is_empty():
12             raise IndexError("Dequeue from empty queue")
13         return self.items.pop(0)
14
15     def peek(self):
16         """Return the front item without removing it. Raises IndexError if empty."""
17         if self.is_empty():
18             raise IndexError("Peek from empty queue")
19         return self.items[0]
20
21     def is_empty(self):
22         """Check if the queue is empty."""
23         return len(self.items) == 0
```

The terminal tab shows the execution of the script:

```
PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windows/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/iinetcache/ie/30000000/ai_lab_11.1_t2.py"
1
1
False
2
3
True
PS C:\Users\kadal>
```

The status bar at the bottom right shows "Spaces: 4" and "Python 3.11.9 (Microsoft Store)".

This screenshot shows the same Visual Studio Code interface as the first one, but the code in the editor has been modified to include a `__str__` method:

```
1  class Queue:
2      def __init__(self):
3          self.items = []
4
5      def enqueue(self, item):
6          """Add an item to the end of the queue."""
7          self.items.append(item)
8
9      def dequeue(self):
10         """Remove and return the front item of the queue. Raises IndexError if empty."""
11         if self.is_empty():
12             raise IndexError("Dequeue from empty queue")
13         return self.items.pop(0)
14
15     def peek(self):
16         """Return the front item without removing it. Raises IndexError if empty."""
17         if self.is_empty():
18             raise IndexError("Peek from empty queue")
19         return self.items[0]
20
21     def is_empty(self):
22         """Check if the queue is empty."""
23         return len(self.items) == 0
24
25     def __str__(self):
26         """Return a string representation of the queue."""
27         return f"Queue({self.items})"
28
29 q = Queue()
30 q.enqueue(1)
31 q.enqueue(2)
32 q.enqueue(3)
33 print(q.peek())    # Output: 1
34 print(q.dequeue()) # Output: 1
35 print(q.is_empty())# Output: False
36 print(q.dequeue()) # Output: 2
37 print(q.dequeue()) # Output: 3
38 print(q.is_empty())# Output: True
```

The terminal output remains the same as in the first screenshot.

EXPLAINATION:



Explanation

Method	Description	Time Complexity
<code>__init__</code>	Initializes an empty list to store queue elements	$O(1)$
<code>enqueue()</code>	Adds an item to the end of the list (rear of the queue)	$O(1)$
<code>dequeue()</code>	Removes and returns the first item (front of the queue)	$O(n)$
<code>peek()</code>	Returns the first item without removing it	$O(1)$
<code>is_empty()</code>	Checks if the queue is empty	$O(1)$

⚠ Note: `dequeue()` uses `pop(0)`, which is $O(n)$ because it shifts all remaining elements. For better performance, you can use `collections.deque`.

TASK 3:

Linked List

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

```
class Node: pass.
```

PROMPT:

Generate python code and linked List

Task: Use AI to generate a Singly Linked List with insert and display methods.

Sample Input Code:

class Node: pass.

CODE & OUTPUT:

The image shows two side-by-side screenshots of Microsoft Visual Studio Code (VS Code) running on a Windows operating system. Both screenshots display a terminal window with Python code and its execution output.

Top Screenshot (Left):

- Code:**

```
C:\> Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBOBXDLA > AI LAB 11.1 T3.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        """Insert a new node at the end of the list."""  
12        new_node = Node(data)  
13        if self.head is None:  
14            self.head = new_node  
15            return  
16        current = self.head  
17        while current.next:  
18            current = current.next  
19        current.next = new_node  
20  
21    def display(self):  
22        """Display the contents of the linked list."""  
23        current = self.head
```
- Terminal Output:**

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "C:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBOBXDLA/AI LAB 11.1 T3.py"  
10 -> 20 -> 30 -> None  
PS C:\Users\kadal>
```

Bottom Screenshot (Right):

- Code:**

```
C:\> Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBOBXDLA > AI LAB 11.1 T3.py > ...  
19 class LinkedList:  
20     current.next = new_node  
21  
22    def display(self):  
23        """Display the contents of the linked list."""  
24        current = self.head  
25        if current is None:  
26            print("Linked List is empty.")  
27            return  
28        while current:  
29            print(current.data, end=" -> ")  
30            current = current.next  
31        print("None")  
32 ll = LinkedList()  
33 ll.insert(10)  
34 ll.insert(20)  
35 ll.insert(30)  
36 ll.display() # Output: 10 -> 20 -> 30 -> None
```
- Terminal Output:**

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "C:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBOBXDLA/AI LAB 11.1 T3.py"  
10 -> 20 -> 30 -> None  
PS C:\Users\kadal>
```

EXPLANATION:

- Represents each element in the list.
- `data` : stores the value.
- `next` : points to the next node (or `None` if it's the last).



`LinkedList` Class

- Manages the chain of nodes.
- `head` : reference to the first node.



`insert(data)`

- Creates a new node.
- If the list is empty, sets it as the head.
- Otherwise, traverses to the end and links the new node.

TASK 4:

Binary Search Tree (BST)

Task: Use AI to create a BST with `insert` and `inorder traversal` methods.

Sample Input Code:

```
class BST: pass.
```

PROMPT:

Generate python code and binary Search Tree (BST)

Task: Use AI to create a BST with insert and inorder traversal methods.

Sample Input Code:

```
class BST: pass.
```

CODE & OUTPUT:

The screenshot shows the Microsoft Visual Studio Code interface. The Explorer sidebar is open, showing 'OPEN EDITORS' and 'NO FOLDER OPENED'. The terminal window at the bottom shows the command PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windows/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/InetCache/IE/JBOOKXDLA/AI LAB 11.1 T4.py". The code editor contains Python code for a Binary Search Tree (BST) with insert and in-order traversal methods.

```
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.left = None
5         self.right = None
6
7 class BST:
8     def __init__(self):
9         self.root = None
10
11     def insert(self, data):
12         """Insert a new node into the BST."""
13         if self.root is None:
14             self.root = Node(data)
15         else:
16             self._insert_recursive(self.root, data)
17
18     def _insert_recursive(self, current, data):
19         if data < current.data:
20             if current.left is None:
21                 current.left = Node(data)
22             else:
23                 self._insert_recursive(current.left, data)
24
25         else:
26             self._insert_recursive(current.right, data)
27
28     def in_order_traversal(self):
29         result = []
30         self._in_order_recursive(self.root, result)
31         return result
32
33     def _in_order_recursive(self, node, result):
34         if node:
35             self._in_order_recursive(node.left, result)
36             result.append(node.data)
37             self._in_order_recursive(node.right, result)
38
39     tree = BST()
40     tree.insert(50)
41     tree.insert(30)
42     tree.insert(70)
43     tree.insert(20)
44     tree.insert(40)
45     tree.insert(60)
46     tree.insert(80)
47
48     print("In-order Traversal:", tree.in_order_traversal())
49     # Output: In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
50
51
52
53
```

The screenshot shows the Microsoft Visual Studio Code interface. The Explorer sidebar is open, showing 'OPEN EDITORS' and 'NO FOLDER OPENED'. The terminal window at the bottom shows the command PS C:\Users\kadal> & c:/users/kadal/appdata/local/microsoft/windows/python3.11.exe "c:/users/kadal/appdata/local/microsoft/windows/InetCache/IE/JBOOKXDLA/AI LAB 11.1 T4.py". The code editor contains Python code for a Binary Search Tree (BST) with insert and in-order traversal methods.

```
1 class Node:
2     def __init__(self, data):
3         self.data = data
4         self.left = None
5         self.right = None
6
7 class BST:
8     def __init__(self):
9         self.root = None
10
11     def insert(self, data):
12         """Insert a new node into the BST."""
13         if self.root is None:
14             self.root = Node(data)
15         else:
16             self._insert_recursive(self.root, data)
17
18     def _insert_recursive(self, current, data):
19         if data < current.data:
20             if current.left is None:
21                 current.left = Node(data)
22             else:
23                 self._insert_recursive(current.left, data)
24
25         else:
26             self._insert_recursive(current.right, data)
27
28     def in_order_traversal(self):
29         result = []
30         self._in_order_recursive(self.root, result)
31         return result
32
33     def _in_order_recursive(self, node, result):
34         if node:
35             self._in_order_recursive(node.left, result)
36             result.append(node.data)
37             self._in_order_recursive(node.right, result)
38
39     tree = BST()
40     tree.insert(50)
41     tree.insert(30)
42     tree.insert(70)
43     tree.insert(20)
44     tree.insert(40)
45     tree.insert(60)
46     tree.insert(80)
47
48     print("In-order Traversal:", tree.in_order_traversal())
49     # Output: In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
50
51
52
53
```

EXPLANATION:

- Adds a new value to the tree.
- If the tree is empty, it becomes the root.
- Otherwise, it uses `_insert_recursive()` to find the correct position:
 - If `data < current.data` : go left.
 - If `data > current.data` : go right.
 - If equal: skip (no duplicates).

• `in_order_traversal()`

- Returns a sorted list of values.
- Uses `_in_order_recursive()` :
 - Traverse left subtree.
 - Visit current node.
 - Traverse right subtree.

TASK 5:

Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code: class HashTable: pass.

PROMPT:

Generate python code and hash Table Task:
Use AI to implement a hash table with

basic insert, search, and delete methods.

Sample Input Code: class HashTable: pass.

CODE & OUTPUT:

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash_function(self, key):
        """Simple hash function using modulo."""
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self._hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value) # Update existing key
                return
        self.table[index].append((key, value))

    def search(self, key):
        """Search for a value by key. Returns value or None."""
        index = self._hash_function(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None

def delete(self, key):
    """Delete a key-value pair from the hash table."""
    index = self._hash_function(key)
    for i, (k, _) in enumerate(self.table[index]):
        if k == key:
            del self.table[index][i]
            return True
    return False

def display(self):
    """Display the contents of the hash table."""
    for i, bucket in enumerate(self.table):
        print(f"Bucket {i}: {bucket}")

ht = HashTable()
ht.insert("apple", 100)
ht.insert("banana", 200)
ht.insert("orange", 300)

print(ht.search("banana")) # Output: 200
ht.delete("apple")
ht.display()
```

EXPLANATION:



`_hash_function(self, key)`

- Uses Python's built-in `hash()` function.
- Applies modulo to ensure the index fits within the table size.



`insert(self, key, value)`

- Computes the index using the hash function.
- Checks if the key already exists in the bucket:
 - If yes, updates the value.
 - If no, appends the new `(key, value)` pair.



`search(self, key)`

- Computes the index and scans the bucket.
- Returns the value if the key is found, otherwise returns `None`.

TASK 6:

Graph Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code: class

Graph:

pass.

PROMPT:

Generate python code and graph
Representation

Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

class Graph: pass.

CODE & OUTPUT:

The screenshot shows the Microsoft Visual Studio Code interface. In the center, there is a code editor with the following Python script:

```
1 class Graph:
2     def __init__(self):
3         self.adj_list = {}
4
5     def add_edge(self, u, v):
6         """Add an edge from vertex u to vertex v (undirected by default)."""
7         if u not in self.adj_list:
8             self.adj_list[u] = []
9         if v not in self.adj_list:
10            self.adj_list[v] = []
11        self.adj_list[u].append(v)
12        self.adj_list[v].append(u) # Remove this line for directed graph
13
14    def display(self):
15        """Display the adjacency list of the graph."""
16        for vertex in self.adj_list:
17            print(f'{vertex} -> {self.adj_list[vertex]}')
18
19 g = Graph()
20 g.add_edge("A", "B")
21 g.add_edge("A", "C")
22 g.add_edge("B", "D")
23 g.add_edge("C", "D")
24 g.display()
```

Below the code editor, the terminal window shows the output of running the script:

```
PS C:\Users\kadal> & c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T6.py"
A -> [B, C]
B -> [A, D]
C -> [A, D]
D -> [B, C]
PS C:\Users\kadal>
```

The status bar at the bottom indicates the following information: Space: 4, UTF-8, Python, 3.11.9 (Microsoft Store), ENG IN, 15:41:57, 16-09-2025.

This screenshot is identical to the one above, showing the same Python script and its execution output in the terminal. The code defines a `Graph` class with methods to add edges and display the adjacency list. The terminal shows the resulting graph structure with vertices A, B, C, and D and their connections.

EXPLANATION:

- Initializes an empty list called `heap`.
- This list will store tuples of `(priority, item)`.

`insert(priority, item)`

- Uses `heapq.heappush()` to add a tuple to the heap.
- The heap maintains order based on the **priority** (lowest number = highest priority).

`remove()`

- Uses `heapq.heappop()` to remove and return the item with the **lowest priority value**.
- Raises an error if the queue is empty.

`peek()`

- Returns the item with the highest priority without removing it.

TASK 7:

Priority Queue

Task: Use AI to implement a priority queue using Python's `heapq` module.

Sample Input Code: class

PriorityQueue: pass.

PROMPT:

Generate a python code and priority Queue

Task: Use AI to implement a priority queue using Python's heapq module.

Sample Input Code: class

PriorityQueue: pass.

CODE & OUTPUT:

The screenshot shows the Microsoft Visual Studio Code interface. The Explorer sidebar indicates 'NO FOLDER OPENED'. The terminal window displays the following Python code for a PriorityQueue class using the heapq module:

```
1 import heapq
2
3 class PriorityQueue:
4     def __init__(self):
5         self.heap = []
6
7     def insert(self, priority, item):
8         """Insert an item with a given priority."""
9         heapq.heappush(self.heap, (priority, item))
10
11    def remove(self):
12        """Remove and return the item with the highest priority (lowest number)."""
13        if self.is_empty():
14            raise IndexError("Remove from empty priority queue")
15        return heapq.heappop(self.heap)[1]
16
17    def peek(self):
18        """Return the item with the highest priority without removing it."""
19        if self.is_empty():
20            raise IndexError("Peek from empty priority queue")
21        return self.heap[0][1]
22
23    def is_empty(self):
24
```

The terminal also shows the execution of the code and its output:

```
wslApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/InetCache/IE/10000000/AI LAB 11.1 T6.py"
A -> [ 'A', 'C' ]
B -> [ 'A', 'D' ]
C -> [ 'A', 'D' ]
D -> [ 'B', 'C' ]
PS C:\Users\kadal> & c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/InetCache/IE/10000000/AI LAB 11.1 T7.py"
Task A
Task A
Priority Queue: [ 'Task B', 'Task C' ]
PS C:\Users\kadal>
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The left sidebar displays the Explorer, Open Editors, and Timeline sections. The main area shows a Python file named `AI LAB 11.1 T.py` with the following code:

```
class PriorityQueue:
    def __init__(self):
        self.heap = [None]
    def __len__(self):
        return len(self.heap) - 1
    def is_empty(self):
        """Check if the priority queue is empty."""
        return len(self.heap) == 1
    def display(self):
        """Display the contents of the priority queue."""
        print("Priority Queue:", [item for _, item in self.heap])
    def insert(self, priority, item):
        self.heap.append((priority, item))
        self._bubble_up(len(self.heap) - 1)
    def peek(self):
        if self.is_empty():
            raise IndexError("Priority Queue is empty")
        return self.heap[1][1]
    def remove(self):
        if self.is_empty():
            raise IndexError("Priority Queue is empty")
        self._swap(1, len(self.heap) - 1)
        item = self.heap.pop()
        self._bubble_down(1)
        return item
    def _bubble_up(self, index):
        parent_index = index // 2
        if parent_index >= 1 and self.heap[parent_index] < self.heap[index]:
            self._swap(index, parent_index)
            self._bubble_up(parent_index)
    def _swap(self, i, j):
        self.heap[i], self.heap[j] = self.heap[j], self.heap[i]
    def _bubble_down(self, index):
        left_index = index * 2
        right_index = index * 2 + 1
        if left_index < len(self.heap) and self.heap[left_index] > self.heap[index]:
            self._swap(index, left_index)
            self._bubble_down(left_index)
        if right_index < len(self.heap) and self.heap[right_index] > self.heap[index]:
            self._swap(index, right_index)
            self._bubble_down(right_index)
```

The terminal tab at the bottom shows the output of running the script:

```
PS C:\Users\kadal> c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/InetCache/IE/3000NDLA/AI LAB 11.1 T.py"
A -> ['B', 'C']
B -> ['A', 'D']
C -> ['A', 'B']
D -> ['B', 'C']
PS C:\Users\kadal>
```

The status bar at the bottom indicates "No connection".

EXPLANATION:

<https://copilot.microsoft.com/hubv/NQ7rc2XvMHCQnGKa51L>

Error Control – Correction Techniques

7. What is the difference between error detection and error correction?

Answer:

- **Detection** identifies the presence of errors.
- **Correction** identifies and fixes the errors without retransmission.

8. Explain Hamming Code with an example.

Answer:

Hamming Code adds redundant bits at positions that are powers of 2. These bits help detect and correct single-bit errors.

Example: For 4-bit data 1011, Hamming code adds 3 parity bits to make it (p1 p2 1 p4) 0 1 1. Parity bits are calculated to detect errors.

9. What is the minimum Hamming distance required to correct single-bit errors?

Message Copilot

Quick response

26°C Light rain

Search

ENG IN 13-09-2023

TASK 8:

Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

```
class DequeDS: pass.
```

PROMPT:

Generate python code and deque Task:

Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

```
class DequeDS: pass.
```

CODE &OUTPUT:

The image shows two side-by-side terminal windows in Microsoft Visual Studio Code, both running the same Python script to demonstrate a Deque class.

Terminal 1 (Left):

```

C:\> Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBOBXDLA > AI LAB 11.1 T8.py ...
1 from collections import deque
2
3 class DequeDS:
4     def __init__(self):
5         self.dequeue = deque()
6
7     def add_front(self, item):
8         """Add an item to the front of the deque."""
9         self.dequeue.appendleft(item)
10
11     def add_rear(self, item):
12         """Add an item to the rear of the deque."""
13         self.dequeue.append(item)
14
15     def remove_front(self):
16         """Remove and return the item from the front."""
17         if self.is_empty():
18             raise IndexError("Remove from empty deque")
19         return self.dequeue.popleft()
20
21     def remove_rear(self):
22         """Remove and return the item from the rear."""
23         if self.is_empty():
24
PS C:\> & c:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "<c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBOBXDLA/AI LAB 11.1 T8.py"
Deque: [20, 10, 30]
20
30
Deque: [10]
PS C:\>

```

Terminal 2 (Right):

```

C:\> Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBOBXDLA > AI LAB 11.1 T8.py ...
3 class DequeDS:
4     def __init__(self):
5         raise IndexError("Peek from empty deque")
6         return self.dequeue[-1]
7
8     def is_empty(self):
9         """Check if the deque is empty."""
10        return len(self.dequeue) == 0
11
12     def display(self):
13         """Display the contents of the deque."""
14        print("Deque:", list(self.dequeue))
15
16    dq = DequeDS()
17    dq.add_rear(10)
18    dq.add_front(20)
19    dq.add_rear(30)
20
21    dq.display()      # Output: Deque: [20, 10, 30]
22    print(dq.peek_front()) # Output: 20
23    print(dq.peek_rear()) # Output: 30
24    dq.remove_front()
25    dq.remove_rear()
26
27    dq.display()      # Output: Deque: [10]
PS C:\>

```

EXPLANATION:

- Initializes an empty deque using `collections.deque`, which is optimized for fast appends and pops from both ends.
-  `add_front(item)`
- Adds an item to the **front** using `appendleft()`.
-  `add_rear(item)`
- Adds an item to the **rear** using `append()`.
-  `remove_front()`
- Removes and returns the item from the **front** using `popleft()`.
-  `remove_rear()`

TASK 9:

AI-Generated Data Structure Comparisons Task:
Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.

Sample Input Code:

```
# No code, prompt AI for a data structure comparison table.
```

PROMPT:

Generate python code and AI-Generated Data Structure Comparisons

Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.

Sample Input Code:

```
# No code, prompt AI for a data structure comparison table.
```

CODE & OUTPUT:

A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows multiple tabs: AI LAB 11.1 T10.py, AI LAB 11.1 T9.py, 9.00 Alipy, Untitled-4, 9 Alipy, Untitled-1, AI LAB 11.1 T4.py, AI LAB 11.1 T5.py, AI LAB 11.1 T6.py, and AI LAB 11.1 T7.py. The main editor area contains Python code for a 'Data Structure Comparison' table and a 'Stack Example'. The code includes imports, class definitions for Stack and Queue, and various methods like enqueue, dequeue, push, pop, peek, and is_empty. The terminal below shows the output of running the code, which includes examples for Queue and Stack operations. The status bar at the bottom right indicates the file is 16107 bytes long, has 4 spaces, and is in UTF-8 encoding.

```
C:\>Users>kadal>AppData>Local>Microsoft>Windows>INetCache>IE>JBO0XDLA>9.00 Alipy>...
1 # Data Structure comparison Table Example
2 def print_data_structure_comparison():
3     table = [
4         ["Data structure", "Insert/Push/Enqueue", "Delete/Pop/Dequeue", "Search/Access", "Peek/Front/End"],
5         ["Stack (List)", "O(1)", "O(1)", "O(n)", "O(1)"],
6         ["Queue (List)", "O(1)", "O(n) (enqueue)", "O(n) (dequeue)", "O(n)", "O(1)"],
7         ["Queue (Deque)", "O(1)", "O(1)", "O(n)", "O(1)"],
8         ["Singly Linked List", "O(1) (at head)", "O(1) (at head)", "O(n)", "O(1) (head)"],
9         ["Doubly Linked List", "O(1) (at ends)", "O(1) (at ends)", "O(n)", "O(1) (ends)"],
10    ]
11    for row in table:
12        print(" | ".join(row))
13
14 # Stack Example
15 class Stack:
16     def __init__(self):
17         self.items = []
18
19     def push(self, item):
20         self.items.append(item)
21
22     def pop(self):
23         return self.items.pop() if self.items else None
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS
Stack pop: 3
Stack peek: 2
Is stack empty? False
Queue Example:
Queue after enqueue: [1, 2, 3]
Queue dequeue: 1
Queue peek: 1
Is queue empty? False
Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
0 1 2 3 No connection
Python + ENG IN 16:00 16-09-2023
```

A screenshot of the Visual Studio Code (VS Code) interface. The title bar shows multiple tabs: AI LAB 11.1 T10.py, AI LAB 11.1 T9.py, 9.00 Alipy, Untitled-4, 9 Alipy, Untitled-1, AI LAB 11.1 T4.py, AI LAB 11.1 T5.py, AI LAB 11.1 T6.py, and AI LAB 11.1 T7.py. The main editor area contains Python code for a 'Queue' class and a 'Linked List' example. The code includes class definitions and various methods like enqueue, dequeue, peek, and is_empty. The terminal below shows the output of running the code, which includes examples for Queue and Linked List operations. A WhatsApp window is open in the foreground, showing a message from 'Sheshi'. The status bar at the bottom right indicates the file is 16107 bytes long, has 4 spaces, and is in UTF-8 encoding.

```
C:\>Users>kadal>AppData>Local>Microsoft>Windows>INetCache>IE>JBO0XDLA>9.00 Alipy>...
32 class Queue:
33     def __init__(self):
34         self.items = []
35
36     def enqueue(self, item):
37         self.items.append(item)
38
39     def dequeue(self):
40         return self.items.pop(0) if self.items else None
41
42     def peek(self):
43         return self.items[0] if self.items else None
44
45     def is_empty(self):
46         return len(self.items) == 0
47
48 # linked List Example
49 class Node:
50     def __init__(self, data):
51         self.data = data
52         self.next = None
53
54 class linkedlist:
55     def __init__(self):
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS
Stack pop: 3
stack peek: 2
is stack empty? False
Queue Example:
Queue after enqueue: [1, 2, 3]
Queue dequeue: 1
Queue peek: 1
Is queue empty? False
Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
0 1 2 3 No connection
Python + ENG IN 16:00 16-09-2023
```

```

81     stack = Stack()
82     stack.push(1)
83     stack.push(2)
84     stack.push(3)
85     print("Stack after pushes:", stack.items)
86     print("Stack pop:", stack.pop())
87     print("Stack peek:", stack.peek())
88     print("Is stack empty?", stack.is_empty())
89
90     print("\nQueue Example:")
91     queue = Queue()
92     queue.enqueue(1)
93     queue.enqueue(2)
94     queue.enqueue(3)
95     print("Queue after enqueues:", queue.items)
96     print("Queue dequeue:", queue.dequeue())
97     print("Queue peek:", queue.peek())
98     print("Is queue empty?", queue.is_empty())
99
100    print("\nLinked List Example:")
101    ll = LinkedList()
102    ll.insert(1)
103    ll.insert(2)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Stack pop: 3
Stack peek: 2
Is stack empty? False

Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False

Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>

21°C Mostly cloudy

EXPLANATION:

`def print_data_structure_comparison():`

- Defines a function that prints a formatted comparison table.

`table = [...]`

- A list of lists, where each inner list represents a row in the table.
- The first row is the header: column titles like "Insert", "Delete", etc.
- Each subsequent row compares a specific data structure.

`" | ".join(row)`

- Joins each element in the row with `" | "` to mimic a table format.
- This makes the output readable and aligned like a markdown-style table.

TASK 10:

Task Description #10 Real-Time Application Challenge – Choose the Right Data Structure Scenario:

Your college wants to develop a Campus Resource Management System that handles:

1. StudentAttendanceTracking–Dailylog of students entering/exiting the campus.
2. EventRegistrationSystem–Manage participants in events with quick search and removal.
3. LibraryBookBorrowing–Keeptrackof available books and their due dates.
4. BusSchedulingSystem–Maintainbus routes and stop connections.
5. CafeteriaOrderQueue–Servestudents in the order they arrive.

Student Task:

- For each feature, select the most appropriate data structure from the list below:
 - o Stack
 - o Queue
 - o Priority Queue
 - o Linked List

- o Binary Search Tree (BST)
- o Graph o Hash Table o
Deque

- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

PROMPT:

Generate python code and task Description
#10 Real-Time Application Challenge – Choose
the

Right Data Structure Scenario:

Your college wants to develop a Campus
Resource Management System that handles:
1. Student Attendance Tracking – Daily log
of students entering/exiting the campus.

2. EventRegistrationSystem—Manage participants in events with quick search and removal.

3. LibraryBookBorrowing—Keep track of available books and their due dates.

4. BusSchedulingSystem—Maintain bus routes and stop connections.

5. CafeteriaOrderQueue—Serve students in the order they arrive.

Student Task:

- For each feature, select the most appropriate data structure from the list below:
 - o Stack
 - o Queue
 - o Priority Queue
 - o Linked List
 - o Binary Search Tree (BST)
 - o Graph
 - o Hash Table
 - o Deque
- Justify your choice in 2–3 sentences per feature.

- Implement one selected feature as a working Python program with AI-assisted code generation.

CODE & OUTPUT:

```

import collections

class CafeteriaQueue:
    """
    Implements a cafeteria order queue using a deque for efficient
    first-in-first-out (FIFO) operations.
    """

    def __init__(self):
        # Using a deque from the collections module for an efficient queue
        self.orders = collections.deque()
        print("Cafeteria Order Queue system initialized.")
        print("-" * 40)

    def add_order(self, student_id, order_details):
        """Adds a new order to the end of the queue."""
        self.orders.append((student_id, order_details))
        print(f"Order for student {student_id} has been placed.")
        print(f"Current queue size: {len(self.orders)}")

    def serve_next_order(self):
        """Serves the next order from the front of the queue."""
        if not self.orders:
            print("X The queue is empty. No orders to serve.")

    def display_queue(self):
        """Displays all orders currently in the queue."""
        if not self.orders:
            print("The queue is currently empty.")
        return

    # Main program to demonstrate the cafeteria queue
    if __name__ == "__main__":
        # Order for Student 101 has been placed.
        Current queue size: 2

        Serving remaining orders...
        ★ Serving order for Student 103: Salad with Grilled chicken
        Remaining orders in queue: 1
        ★ Serving order for Student 104: Veggie Wrap
        Remaining orders in queue: 0

        Attempting to serve from an empty queue...
        X The queue is empty. No orders to serve.
        PS C:\Users\kadal>
  
```

```

import collections

class CafeteriaQueue:
    """
    Implements a cafeteria order queue using a deque for efficient
    first-in-first-out (FIFO) operations.
    """

    def __init__(self):
        # Using a deque from the collections module for an efficient queue
        self.orders = collections.deque()
        print("Cafeteria Order Queue system initialized.")
        print("-" * 40)

    def add_order(self, student_id, order_details):
        """Adds a new order to the end of the queue."""
        self.orders.append((student_id, order_details))
        print(f"Order for student {student_id} has been placed.")
        print(f"Current queue size: {len(self.orders)}")

    def serve_next_order(self):
        """Serves the next order from the front of the queue."""
        if not self.orders:
            print("X The queue is empty. No orders to serve.")

    def display_queue(self):
        """Displays all orders currently in the queue."""
        if not self.orders:
            print("The queue is currently empty.")
        return

    # Main program to demonstrate the cafeteria queue
    if __name__ == "__main__":
        # Order for Student 101 has been placed.
        Current queue size: 2

        Serving remaining orders...
        ★ Serving order for Student 103: Salad with Grilled chicken
        Remaining orders in queue: 1
        ★ Serving order for Student 104: Veggie Wrap
        Remaining orders in queue: 0

        Attempting to serve from an empty queue...
        X The queue is empty. No orders to serve.
        PS C:\Users\kadal>
  
```

```

    # Serving the next student
    print("\nAttempting to serve next order...")
    cafeteria_queue.serve_next_order()

    # Display the updated queue
    cafeteria_queue.display_queue()

    # Another student places an order
    cafeteria_queue.add_order(student_id=104, order_details="Veggie Wrap")

    # Serve the remaining orders
    print("\nServing remaining orders...")
    while cafeteria_queue.orders:
        cafeteria_queue.serve_next_order()

    # Try to serve from an empty queue
    print("\nAttempting to serve from an empty queue...")
    cafeteria_queue.serve_next_order()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS

Order for Student 104 has been placed.
Current queue size: 2

Serving remaining orders...
★ Serving order for student 103: Salad with Grilled chicken
Remaining orders in queue: 1
★ Serving order for student 104: Veggie Wrap
Remaining orders in queue: 0

Attempting to serve from an empty queue...
✗ The queue is empty. No orders to serve.

OUTLINE TIMELINE No connection

31°C Mostly cloudy

EXPLANATION:

- Initializes an empty list `queue` to store student names.

`place_order(student_name)`

- Adds a student to the end of the queue using `append()` — O(1) time.

`serve_order()`

- Removes and returns the first student using `pop(0)` — O(n) time due to shifting.
- Raises an error if the queue is empty.

`peek_next()`

- Returns the first student without removing them — O(1) time.

`is_empty()`

- Checks if the queue is empty — O(1) time.

