AI Assisted Coding Lab Assignement-11.1

Name: M. Sai Teja

Batch:14

2303A510A3

Task Description #1 – Stack Implementation

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

Sample Input Code:

class Stack:

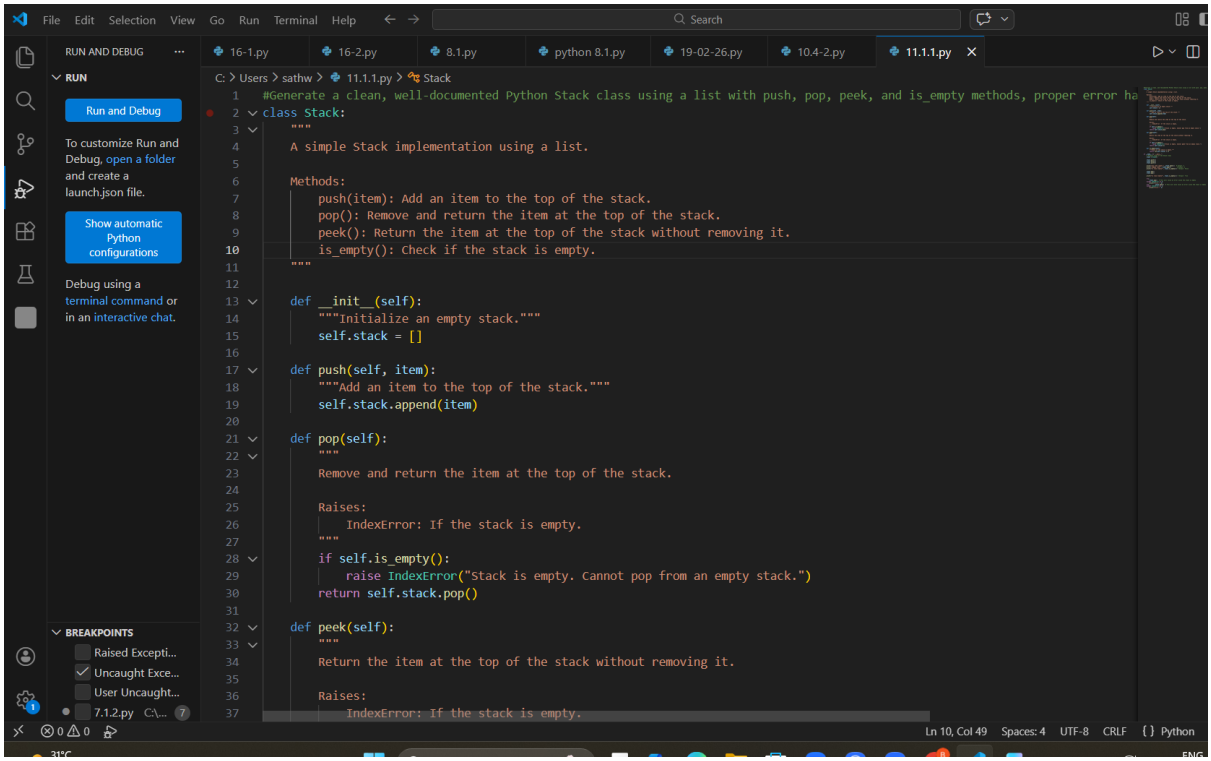
pass

Expected Output:

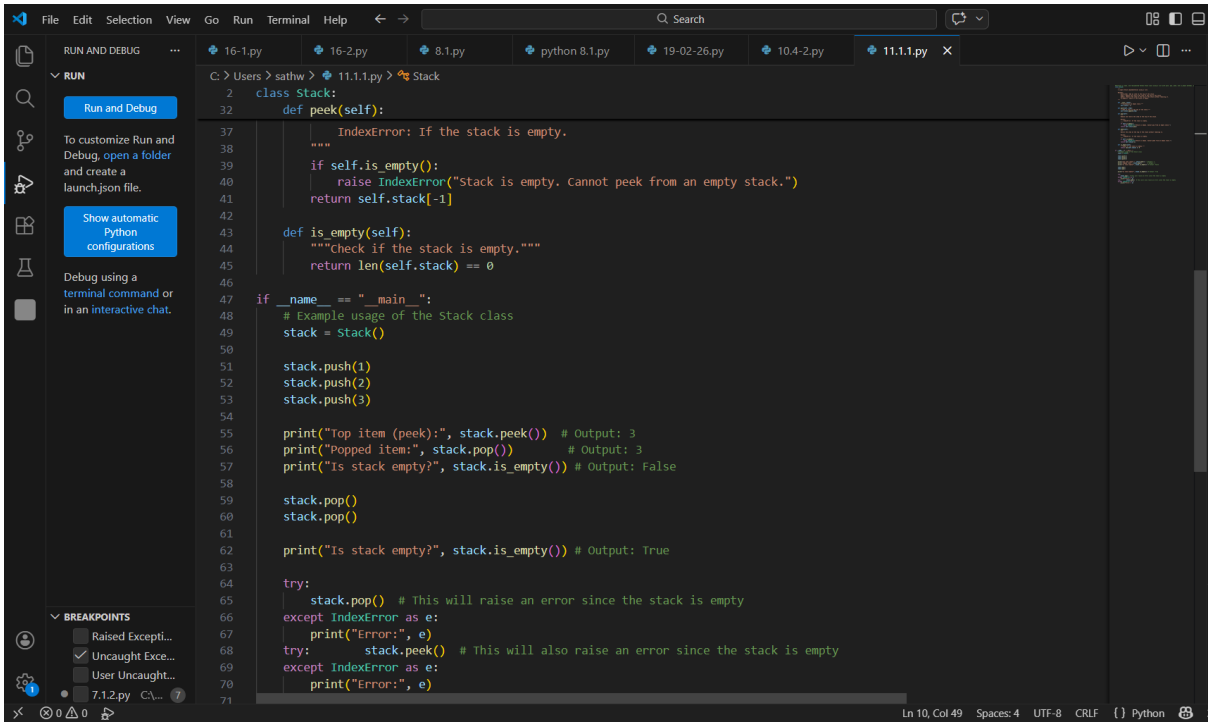
• A functional stack implementation with all required methods and docstrings.

PROMPT:

#Generate a clean, well-documented Python Stack class using a list with push, pop, peek, and is_empty methods, proper error handling, comments, and example usage.

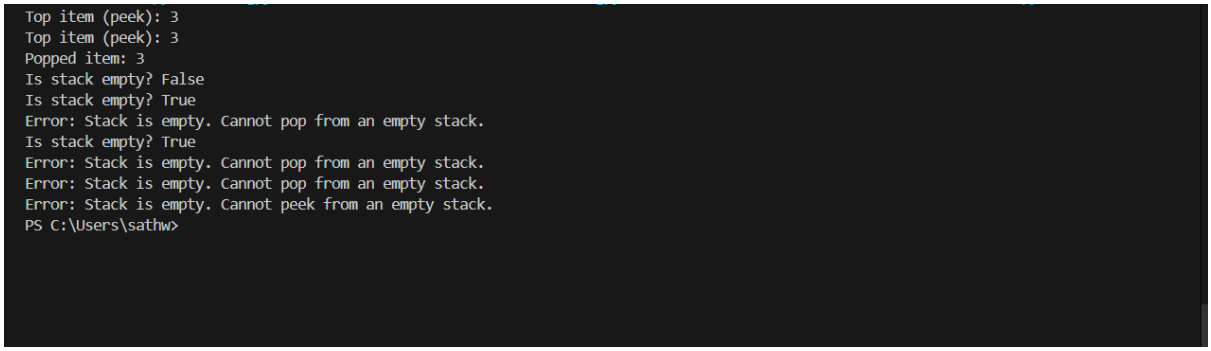


```python
#Generate a clean, well-documented Python Stack class using a list with push, pop, peek, and is_empty methods, proper error ha
class Stack:
    """
    A simple Stack implementation using a list.

    Methods:
        push(item): Add an item to the top of the stack.
        pop(): Remove and return the item at the top of the stack.
        peek(): Return the item at the top of the stack without removing it.
        is_empty(): Check if the stack is empty.
    """

    def __init__(self):
        """Initialize an empty stack."""
        self.stack = []

    def push(self, item):
        """Add an item to the top of the stack."""
        self.stack.append(item)

    def pop(self):
        """
        Remove and return the item at the top of the stack.

        Raises:
            IndexError: If the stack is empty.
        """
        if self.is_empty():
            raise IndexError("Stack is empty. Cannot pop from an empty stack.")
        return self.stack.pop()

    def peek(self):
        """
        Return the item at the top of the stack without removing it.

        Raises:
            IndexError: If the stack is empty.
```



```python
class Stack:
    def peek(self):
            IndexError: If the stack is empty.
        """
        if self.is_empty():
            raise IndexError("Stack is empty. Cannot peek from an empty stack.")
        return self.stack[-1]

    def is_empty(self):
        """Check if the stack is empty."""
        return len(self.stack) == 0

if __name__ == "__main__":
    # Example usage of the Stack class
    stack = Stack()

    stack.push(1)
    stack.push(2)
    stack.push(3)

    print("Top item (peek):", stack.peek())  # Output: 3
    print("Popped item:", stack.pop())       # Output: 3
    print("Is stack empty?", stack.is_empty()) # Output: False

    stack.pop()
    stack.pop()

    print("Is stack empty?", stack.is_empty()) # Output: True

    try:
        stack.pop()  # This will raise an error since the stack is empty
    except IndexError as e:
        print("Error:", e)
    try:        stack.peek()  # This will also raise an error since the stack is empty
    except IndexError as e:
        print("Error:", e)
```

OUTPUT:



Task 1 – Stack

Justification:
Stack follows the LIFO principle, where the last inserted element is removed first. It is useful in undo operations, recursion, and expression evaluation.

Task Description #2 – Queue Implementation

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

Sample Input Code:

class Queue:

pass

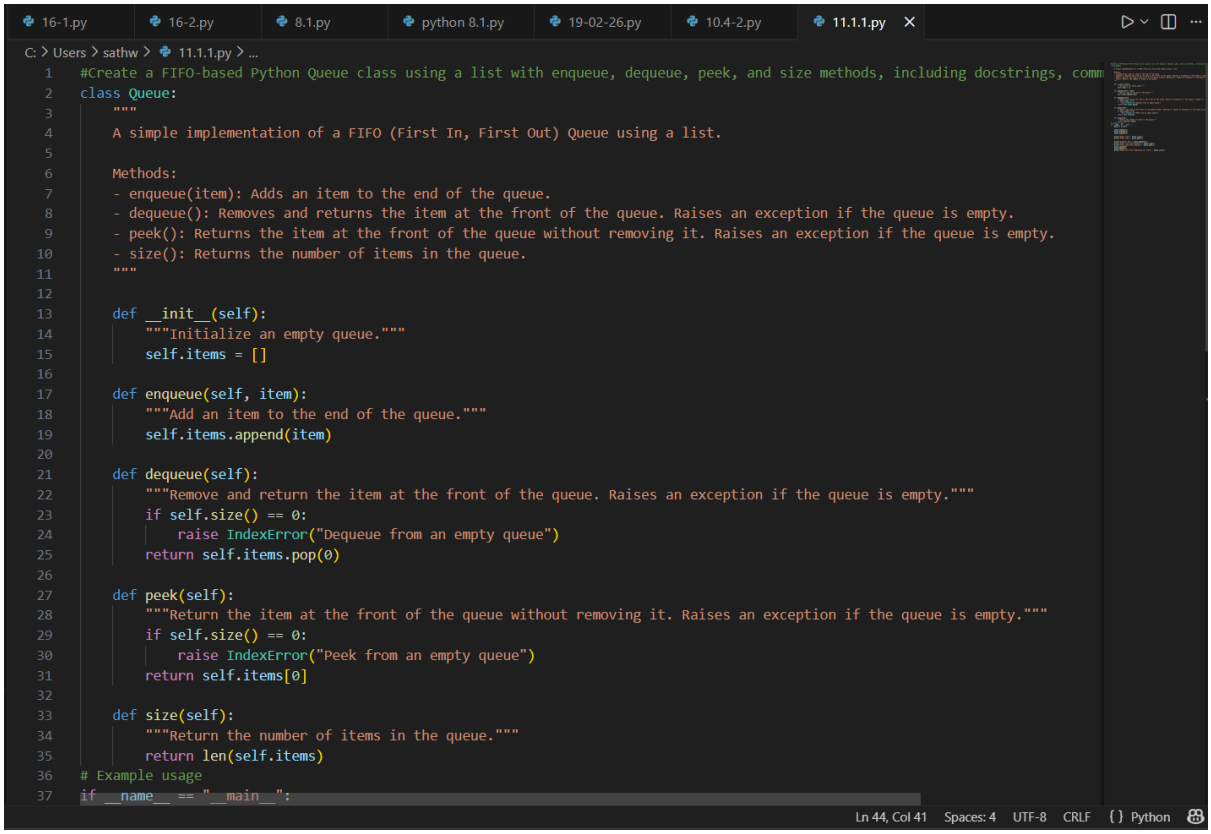
Expected Output:

• FIFO-based queue class with enqueue, dequeue, peek, and size

methods.

PROMPT:

Create a FIFO-based Python Queue class using a list with enqueue, dequeue, peek, and size methods, including docstrings, comments, and example usage.



```python
#Create a FIFO-based Python Queue class using a list with enqueue, dequeue, peek, and size methods, including docstrings, comm
class Queue:
    """
    A simple implementation of a FIFO (First In, First Out) Queue using a list.

    Methods:
    - enqueue(item): Adds an item to the end of the queue.
    - dequeue(): Removes and returns the item at the front of the queue. Raises an exception if the queue is empty.
    - peek(): Returns the item at the front of the queue without removing it. Raises an exception if the queue is empty.
    - size(): Returns the number of items in the queue.
    """

    def __init__(self):
        """Initialize an empty queue."""
        self.items = []

    def enqueue(self, item):
        """Add an item to the end of the queue."""
        self.items.append(item)

    def dequeue(self):
        """Remove and return the item at the front of the queue. Raises an exception if the queue is empty."""
        if self.size() == 0:
            raise IndexError("Dequeue from an empty queue")
        return self.items.pop(0)

    def peek(self):
        """Return the item at the front of the queue without removing it. Raises an exception if the queue is empty."""
        if self.size() == 0:
            raise IndexError("Peek from an empty queue")
        return self.items[0]

    def size(self):
        """Return the number of items in the queue."""
        return len(self.items)
# Example usage
if __name__ == "__main__":
```

Ln 44, Col 41 Spaces: 4 UTF-8 CRLF { } Python

```python
        return len(self.items)
# Example usage
if __name__ == "__main__":
    queue = Queue()

    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)

    print("Queue size:", queue.size())
    print("Front item:", queue.peek())

    print("Dequeue item:", queue.dequeue())
    print("Queue size after dequeue:", queue.size())
    print("Front item after dequeue:", queue.peek())
    queue.dequeue()
    queue.dequeue()
    print("Queue size after dequeuing all items:", queue.size())
```

OUTPUT:

```
Queue size: 3
Front item: 1
Dequeue item: 1
Queue size after dequeue: 2
Front item after dequeue: 2
Queue size after dequeuing all items: 0
PS C:\Users\sathw>
```

Task 2 – Queue

Justification:
Queue follows the FIFO principle, where the first inserted element is removed first. It is commonly used in scheduling and real-time processing systems.

Task Description #3 – Linked List

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

methods.

Sample Input Code:

class Node:

pass

class LinkedList:

pass

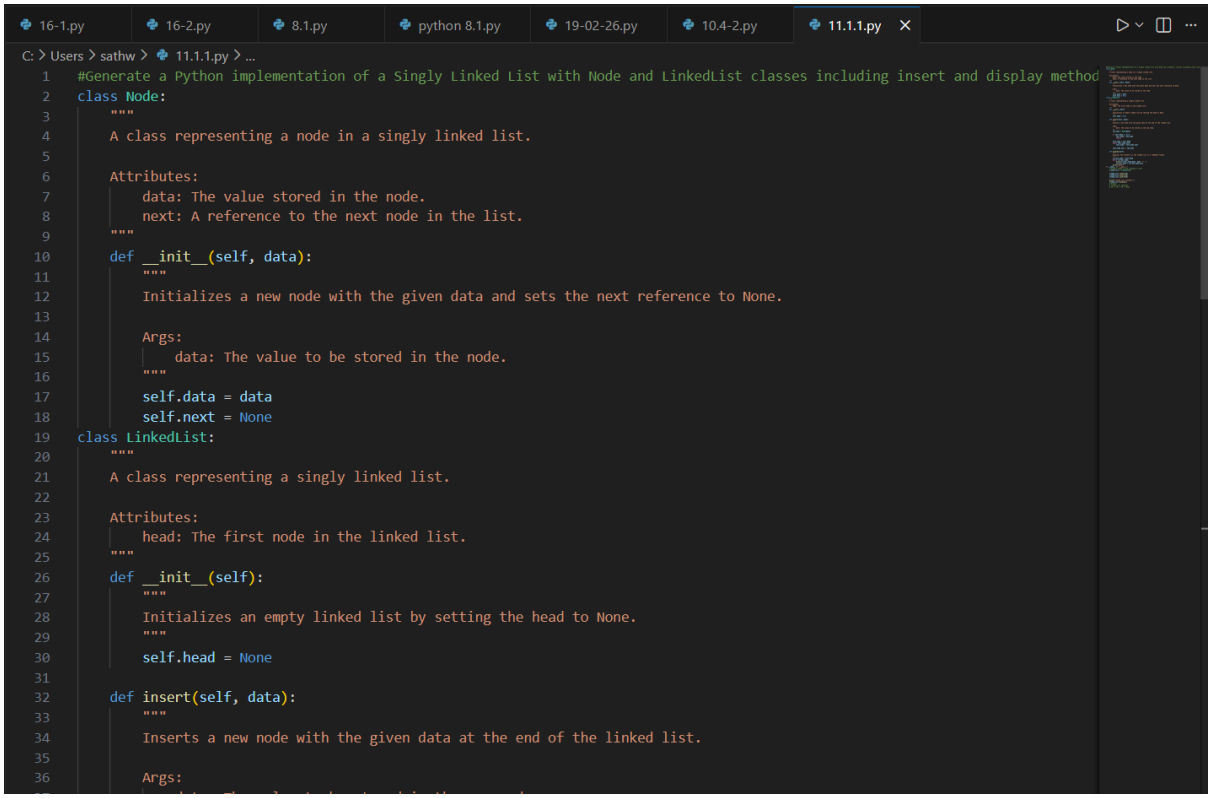
Expected Output:

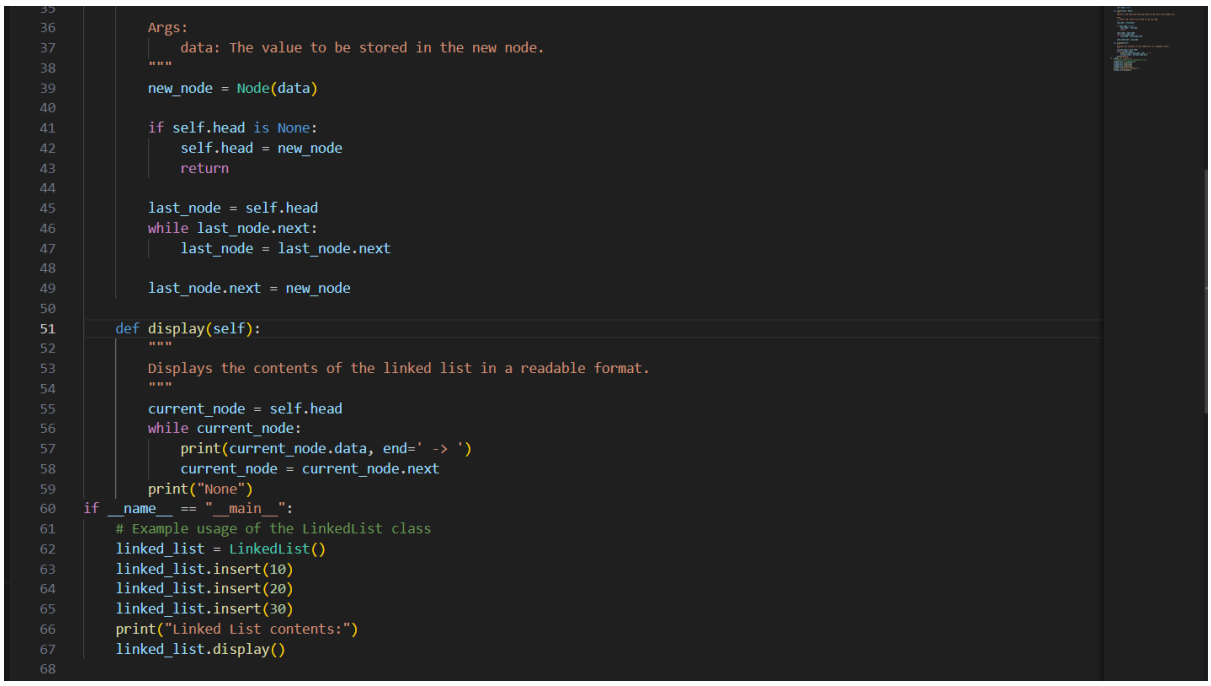
• A working linked list implementation with clear method

documentation.

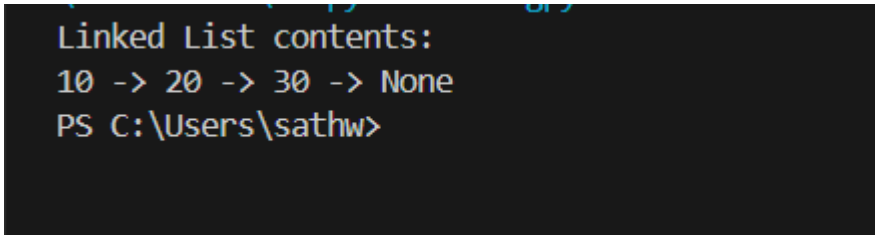
PROMPT:

#Generate a Python implementation of a Singly Linked List with Node and LinkedList classes including insert and display methods, proper documentation, comments, and example usage.


16-1.py
16-2.py
8.1.py
python 8.1.py
19-02-26.py
10.4-2.py
11.1.1.py
C: > Users > sathw > 11.1.1.py > ...
#Generate a Python implementation of a Singly Linked List with Node and LinkedList classes including insert and display method
class Node:
"""
A class representing a node in a singly linked list.
Attributes:
data: The value stored in the node.
next: A reference to the next node in the list.
"""
def __init__(self, data):
"""
Initializes a new node with the given data and sets the next reference to None.
Args:
data: The value to be stored in the node.
"""
self.data = data
self.next = None
class LinkedList:
"""
A class representing a singly linked list.
Attributes:
head: The first node in the linked list.
"""
def __init__(self):
"""
Initializes an empty linked list by setting the head to None.
"""
self.head = None
def insert(self, data):
"""
Inserts a new node with the given data at the end of the linked list.
Args:


Args:
data: The value to be stored in the new node.
"""
new_node = Node(data)
if self.head is None:
self.head = new_node
return
last_node = self.head
while last_node.next:
last_node = last_node.next
last_node.next = new_node
def display(self):
"""
Displays the contents of the linked list in a readable format.
"""
current_node = self.head
while current_node:
print(current_node.data, end=' -> ')
current_node = current_node.next
print("None")
if __name__ == "__main__":
# Example usage of the LinkedList class
linked_list = LinkedList()
linked_list.insert(10)
linked_list.insert(20)
linked_list.insert(30)
print("Linked List contents:")
linked_list.display()

OUTPUT:



Task 3 – Linked List

Justification:
Linked List allows dynamic memory allocation and easy insertion or deletion. It is useful when the size of data changes frequently.

Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

Sample Input Code:

class BST:

pass

Expected Output:

• BST implementation with recursive insert and traversal methods.

PROMPT:

#Create a Python Binary Search Tree with recursive insert and in-order traversal methods, maintaining BST properties, with docstrings, comments, and example usage.

```python
class BST:
    """Binary Search Tree Implementation"""

    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    def insert(self, value):
        """Recursive insert."""
        if value < self.value:
            if self.left:
                self.left.insert(value)
            else:
                self.left = BST(value)
        else:
            if self.right:
                self.right.insert(value)
            else:
                self.right = BST(value)

    def inorder(self):
        """In-order traversal (sorted output)."""
        if self.left:
            self.left.inorder()
        print(self.value, end=" ")
        if self.right:
            self.right.inorder()
# ---------- MAIN PROGRAM ----------
# Take input from user
values = list(map(int, input("Enter numbers separated by space: ").split()))
# Create BST with first value
root = BST(values[0])
# Insert remaining values
for value in values[1:]:
    root.insert(value)
# Display sorted output
print("In-order Traversal (Sorted):")
root.inorder()
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60960' '--' 'C:\Users\sathw\11.1.1.py'
Enter numbers separated by space: 50 70 20 40 10 15 35
Enter numbers separated by space: 50 70 20 40 10 15 35
In-order Traversal (Sorted):
In-order Traversal (Sorted):
10 15 20 35 40 50 70
PS C:\Users\sathw>
```

Task 4 – Binary Search Tree (BST)

Justification:

BST maintains elements in sorted order for efficient searching. It provides faster insert and search operations compared to linear structures.

Task Description #5 – Hash Table

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

Sample Input Code:

class HashTable:

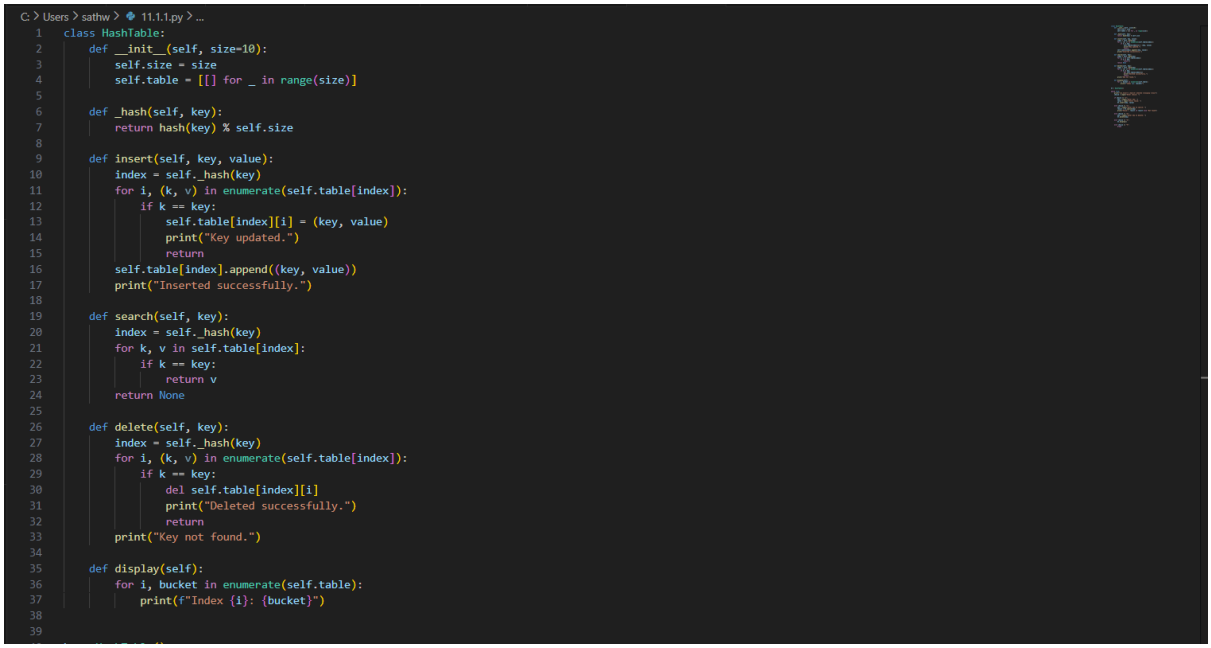
pass

Expected Output:

• Collision handling using chaining, with well-commented methods.

PROMPT:

#Implement a Python Hash Table using chaining for collision handling with insert, search, and delete methods, custom hash function, proper documentation, and example usage.



```
C: > Users > sathw > 11.1.1.py > ...
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        return hash(key) % self.size

    def insert(self, key, value):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                print("Key updated.")
                return
        self.table[index].append((key, value))
        print("Inserted successfully.")

    def search(self, key):
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None

    def delete(self, key):
        index = self._hash(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                print("Deleted successfully.")
                return
        print("Key not found.")

    def display(self):
        for i, bucket in enumerate(self.table):
            print(f"Index {i}: {bucket}")
```

```python

ht = HashTable()

while True:
    print("\n1.Insert 2.Search 3.Delete 4.Display 5.Exit")
    choice = input("Enter choice: ")

    if choice == "1":
        key = input("Enter key: ")
        value = input("Enter value: ")
        ht.insert(key, value)

    elif choice == "2":
        key = input("Enter key to search: ")
        result = ht.search(key)
        print("Value:", result if result else "Not found")

    elif choice == "3":
        key = input("Enter key to delete: ")
        ht.delete(key)

    elif choice == "4":
        ht.display()

    elif choice == "5":
        break


```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 1
Enter key: 101
Enter value: BALA
Inserted successfully.

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 2
Enter key to search: 1
Value: Not found

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 2
Enter key to search: 101
Value: BALA

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 3
Enter key to delete: 101
Deleted successfully.

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 4
Index 0: []
Index 1: []
Index 2: []
Index 3: []
Index 4: []
Index 5: []
Index 6: []
Index 7: []
Index 8: []
Index 9: []

1.Insert 2.Search 3.Delete 4.Display 5.Exit
Enter choice: 5
PS C:\Users\sathw>
```

Task 5 – Hash Table

Justification:
Hash Table provides very fast lookup, insertion, and deletion using keys. It is ideal for applications requiring quick data retrieval.

Task Description #6 – Graph Representation

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

Sample Input Code:

class Graph:

pass

Expected Output:

• Graph with methods to add vertices, add edges, and display

connections.

PROMPT:

#Generate a Python Graph implementation using an adjacency list with add_vertex, add_edge, and display methods, including documentation, comments, and example usage.

C: > Users > sathw > 11.1.6.py > ...

```python
#Generate a Python Graph implementation using an adjacency list with add_vertex, add_edge, and display methods, including documentation, comments, and example usag
class Graph:
    """
    A class to represent a graph using an adjacency list.

    Attributes:
    graph (dict): A dictionary to store the graph where keys are vertices and values are lists of adjacent vertices.

    Methods:
    add_vertex(v): Adds a vertex to the graph.
    add_edge(v1, v2): Adds an edge between two vertices in the graph.
    display(): Displays the graph in an adjacency list format.
    """

    def __init__(self):
        """Initialize the graph as an empty dictionary."""
        self.graph = {}

    def add_vertex(self, v):
        """
        Add a vertex to the graph.

        Parameters:
        v (str): The vertex to be added.
        """
        if v not in self.graph:
            self.graph[v] = []

    def add_edge(self, v1, v2):
        """
        Add an edge between two vertices in the graph.

        Parameters:
        v1 (str): The first vertex.
        v2 (str): The second vertex.
        """
        self.graph.setdefault(v1, []).append(v2)
        self.graph.setdefault(v2, []).append(v1)

    def display(self):
        """Display the graph in an adjacency list format."""
        for vertex in self.graph:
            print(vertex, "->", self.graph[vertex])
# Example usage
if __name__ == "__main__":
    g = Graph()

    while True:
        print("\n1.Add Vertex 2.Add Edge 3.Display 4.Exit")
        choice = input("Enter choice: ")

        if choice == "1":
            v = input("Enter vertex: ")
            g.add_vertex(v)

        elif choice == "2":
            v1 = input("Enter first vertex: ")
            v2 = input("Enter second vertex: ")
            g.add_edge(v1, v2)

        elif choice == "3":
            g.display()

        elif choice == "4":
            break
```

OUTPUT:

Task 6 – Graph

Justification:
Graph represents relationships between nodes using edges. It is suitable for networks, routes, and connected systems.

Task Description #7 – Priority Queue

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

module.

Sample Input Code:

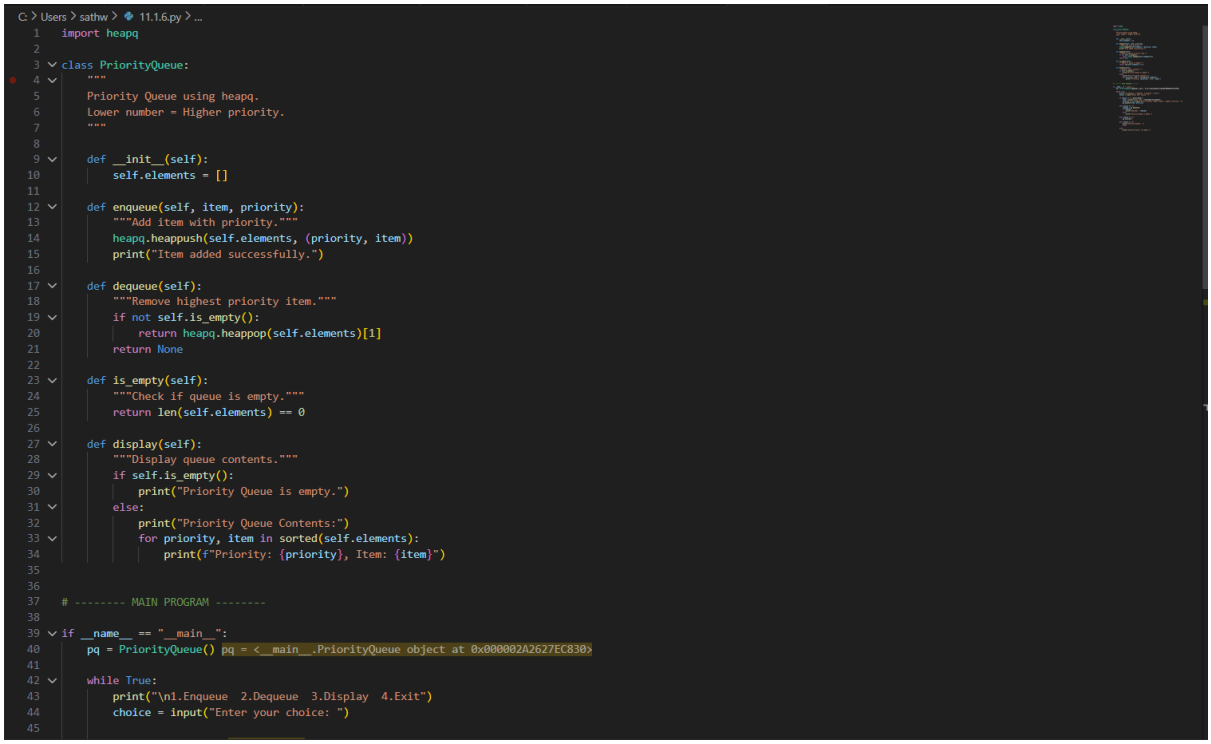
class PriorityQueue:

pass

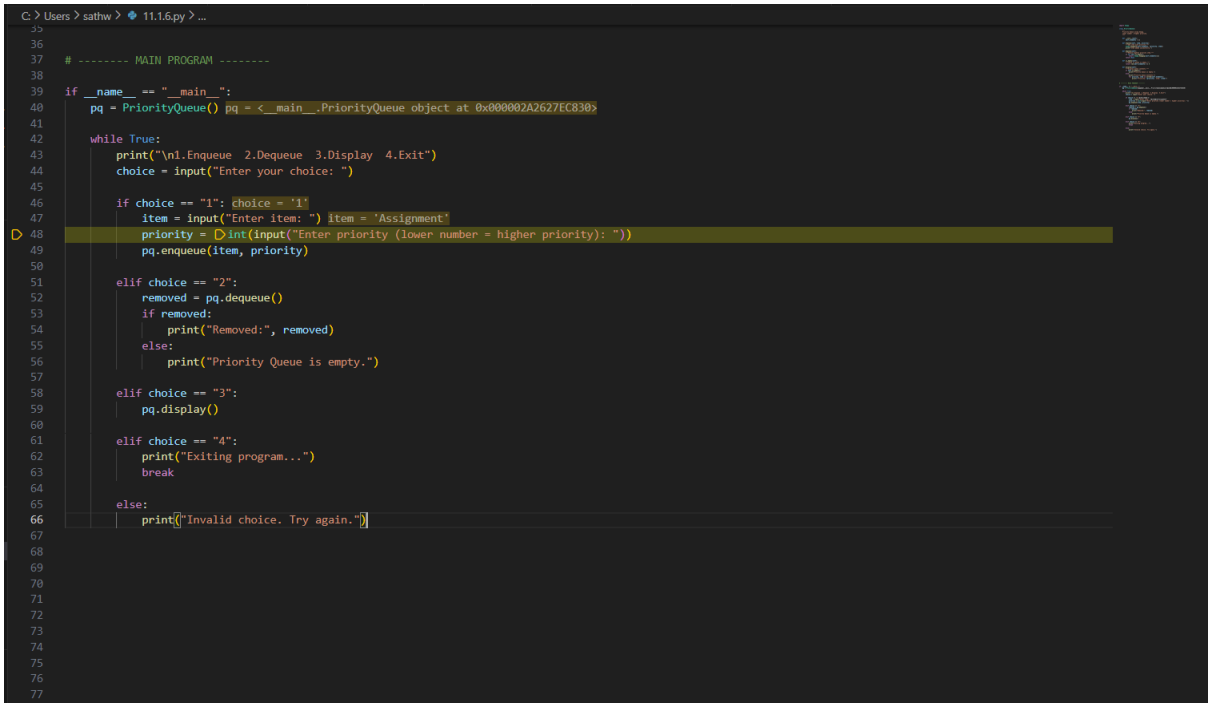
Expected Output:

• Implementation with enqueue (priority), dequeue (highest priority),

and display methods.



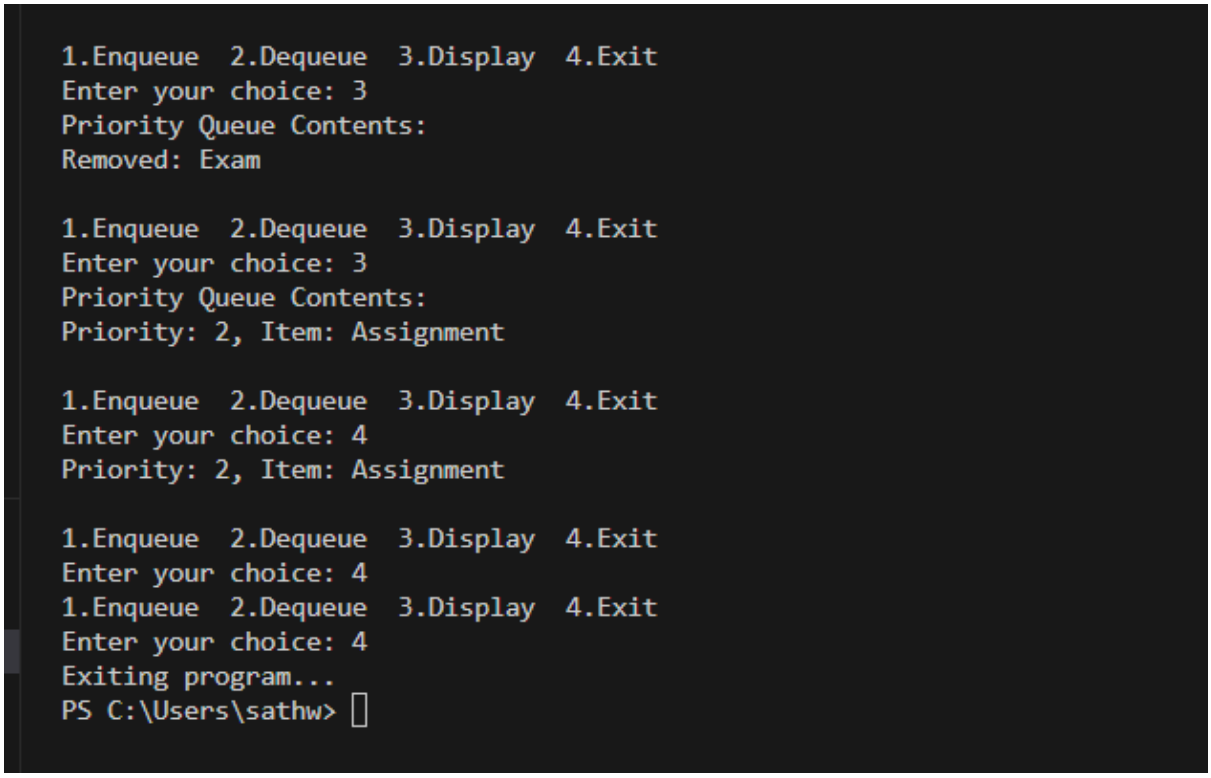
```
C: > Users > sathw > 11.1.6.py > ...
import heapq

class PriorityQueue:
    """
    Priority Queue using heapq.
    Lower number = Higher priority.
    """

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

    def enqueue(self, item, priority):
        """Add item with priority."""
        heapq.heappush(self.elements, (priority, item))
        print("Item added successfully.")

    def dequeue(self):
        """Remove highest priority item."""
        if not self.is_empty():
            return heapq.heappop(self.elements)[1]
        return None

    def is_empty(self):
        """Check if queue is empty."""
        return len(self.elements) == 0

    def display(self):
        """Display queue contents."""
        if self.is_empty():
            print("Priority Queue is empty.")
        else:
            print("Priority Queue Contents:")
            for priority, item in sorted(self.elements):
                print(f"Priority: {priority}, Item: {item}")


# -------- MAIN PROGRAM --------

if __name__ == "__main__":
    pq = PriorityQueue() pq = <__main__.PriorityQueue object at 0x000002A2627EC830>

    while True:
        print("\n1.Enqueue  2.Dequeue  3.Display  4.Exit")
        choice = input("Enter your choice: ")
```



```
C: > Users > sathw > 11.1.6.py > ...


# -------- MAIN PROGRAM --------

if __name__ == "__main__":
    pq = PriorityQueue() pq = <__main__.PriorityQueue object at 0x000002A2627EC830>

    while True:
        print("\n1.Enqueue  2.Dequeue  3.Display  4.Exit")
        choice = input("Enter your choice: ")

        if choice == "1": choice = '1'
            item = input("Enter item: ") item = 'Assignment'
            priority = int(input("Enter priority (lower number = higher priority): "))
            pq.enqueue(item, priority)

        elif choice == "2":
            removed = pq.dequeue()
            if removed:
                print("Removed:", removed)
            else:
                print("Priority Queue is empty.")

        elif choice == "3":
            pq.display()

        elif choice == "4":
            print("Exiting program...")
            break

        else:
            print("Invalid choice. Try again.")
```

OUTPUT:

```
1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice: 1
Enter item: Assignment
Enter priority (lower number = higher priority): 2
Item added successfully.

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice: 1
Enter item: Exam
Enter priority (lower number = higher priority): 1
Item added successfully.

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice: 3
Priority Queue Contents:
Priority: 1, Item: Exam
Priority: 2, Item: Assignment

1.Enqueue  2.Dequeue  3.Display  4.Exit
1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice: 3
Priority Queue Contents:
Priority: 1, Item: Exam
Priority: 2, Item: Assignment

1.Enqueue  2.Dequeue  3.Display  4.Exit
Priority Queue Contents:
Priority: 1, Item: Exam
Priority: 2, Item: Assignment

1.Enqueue  2.Dequeue  3.Display  4.Exit
Priority: 1, Item: Exam
Priority: 2, Item: Assignment

1.Enqueue  2.Dequeue  3.Display  4.Exit
Priority: 2, Item: Assignment

1.Enqueue  2.Dequeue  3.Display  4.Exit

1.Enqueue  2.Dequeue  3.Display  4.Exit
Enter your choice: 2
Enter your choice: 2
Removed: Exam
```



Task 7 – Priority Queue

Justification:
Priority Queue processes elements based on priority instead of order of arrival. It is used in scheduling and shortest path algorithms.

Task Description #8 – Deque

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

collections.deque.

Sample Input Code:

class DequeDS:

pass

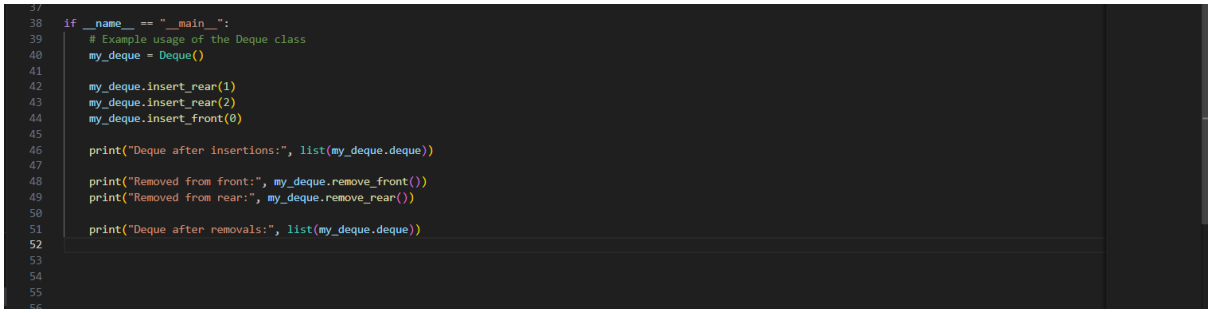
Expected Output:

• Insert and remove from both ends with docstrings.

PROMPT:

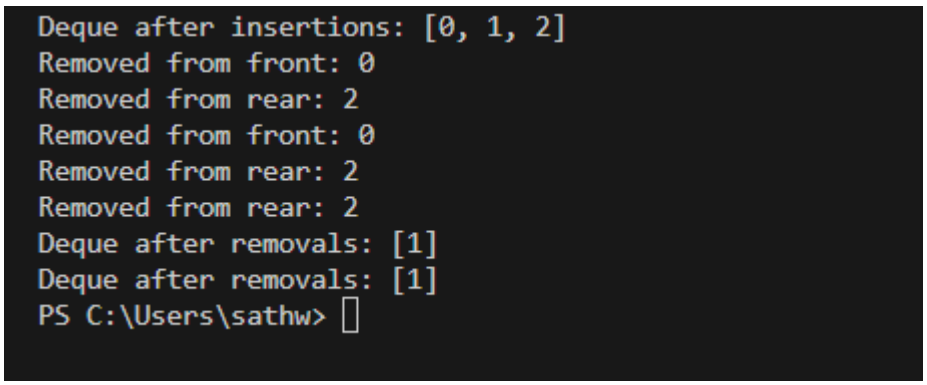
#Implement a Python Deque using collections.deque with insert_front, insert_rear, remove_front, and remove_rear methods, including proper documentation and example usage.

```python
#Implement a Python Deque using collections.deque with insert_front, insert_rear, remove_front, and remove_rear methods, including proper documentation and example
from collections import deque
class Deque:
    """
    A double-ended queue (deque) implementation using collections.deque.

    Methods:
    - insert_front(item): Inserts an item at the front of the deque.
    - insert_rear(item): Inserts an item at the rear of the deque.
    - remove_front(): Removes and returns the item at the front of the deque.
    - remove_rear(): Removes and returns the item at the rear of the deque.
    """

    def __init__(self):
        """Initialize an empty deque."""
        self.deque = deque()

    def insert_front(self, item):
        """Insert an item at the front of the deque."""
        self.deque.appendleft(item)

    def insert_rear(self, item):
        """Insert an item at the rear of the deque."""
        self.deque.append(item)

    def remove_front(self):
        """Remove and return the item at the front of the deque. Raises IndexError if empty."""
        if not self.deque:
            raise IndexError("Deque is empty")
        return self.deque.popleft()

    def remove_rear(self):
        """Remove and return the item at the rear of the deque. Raises IndexError if empty."""
        if not self.deque:
            raise IndexError("Deque is empty")
        return self.deque.pop()
```



```python
if __name__ == "__main__":
    # Example usage of the Deque class
    my_deque = Deque()

    my_deque.insert_rear(1)
    my_deque.insert_rear(2)
    my_deque.insert_front(0)

    print("Deque after insertions:", list(my_deque.deque))

    print("Removed from front:", my_deque.remove_front())
    print("Removed from rear:", my_deque.remove_rear())

    print("Deque after removals:", list(my_deque.deque))
```

OUTPUT:



```
Deque after insertions: [0, 1, 2]
Removed from front: 0
Removed from rear: 2
Removed from front: 0
Removed from rear: 2
Removed from rear: 2
Deque after removals: [1]
Deque after removals: [1]
PS C:\Users\sathw>
```

Task 8 – Deque

Justification:
Deque allows insertion and deletion from both ends. It is flexible and can work as both stack and queue.

Task Description #9 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. Event Registration System – Manage participants in events with

quick search and removal.

3. Library Book Borrowing – Keep track of available books and their

due dates.

4. Bus Scheduling System – Maintain bus routes and stop

connections.

5. Cafeteria Order Queue – 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.

Expected Output:

• A table mapping feature → chosen data structure → justification.

• A functional Python program implementing the chosen feature

with comments and docstrings.

PROMPT:

#Analyze the given campus system features, select the most appropriate data structure for each with justification, present the mapping in a table, and implement one feature as a documented Python program.

C: > Users > sathw > 11.1.1.py > ...

```python
#Analyze the given campus system features, select the most appropriate data structure for each with justification, present the mapping in a table, and implement on
class StudentRecords:
    """
    Student Records Management using Dictionary.
    Key = Student ID
    Value = Student details (name, age, major)
    """

    def __init__(self):
        self.records = {}

    def add_student(self, student_id, name, age, major):
        """Add a new student."""
        self.records[student_id] = {
            "name": name,
            "age": age,
            "major": major
        }
        print("Student added successfully.")

    def get_student(self, student_id):
        """Retrieve student details."""
        if student_id in self.records:
            info = self.records[student_id]
            print(f"ID: {student_id}")
            print(f"Name: {info['name']}")
            print(f"Age: {info['age']}")
            print(f"Major: {info['major']}")
        else:
            print("Student not found.")

    def remove_student(self, student_id):
        """Remove student from records."""
        if student_id in self.records:
            del self.records[student_id]
            print("Student removed successfully.")
        else:
            print("Student not found.")

    def display_all_students(self):
        """Display all student records."""
        if not self.records:
            print("No student records available.")
        else:
            for student_id, info in self.records.items():
                print(f"\nID: {student_id}")
                print(f"Name: {info['name']}")
                print(f"Age: {info['age']}")
                print(f"Major: {info['major']}")
# -------- MAIN PROGRAM --------
if __name__ == "__main__":
    student_records = StudentRecords()
    while True:
        print("\n1. Add Student")
        print("2. View Student")
        print("3. Remove Student")
        print("4. Display All Students")
        print("5. Exit")
        choice = input("Enter your choice: ")
        if choice == "1":
            student_id = input("Enter Student ID: ")
            name = input("Enter Name: ")
            age = input("Enter Age: ")
            major = input("Enter Major: ")
            student_records.add_student(student_id, name, age, major)

        elif choice == "2":
            student_id = input("Enter Student ID to search: ")
            student_records.get_student(student_id)

        elif choice == "3":
            student_id = input("Enter Student ID to remove: ")
            student_records.remove_student(student_id)

        elif choice == "4":
            student_records.display_all_students()

        elif choice == "5":
            print("Exiting program...")
            break
        else:
            print("Invalid choice. Try again.")
```

OUTPUT:

```
1. Add Student
2. View Student
3. Remove Student
4. Display All Students
5. Exit
Enter your choice: 1
Enter Student ID: 10B1
Enter Name: BALA
Enter Age: 21
Enter Major: CSE
Student added successfully.

1. Add Student
2. View Student
3. Remove Student
4. Display All Students
5. Exit
Enter your choice: 2
Enter Student ID to search: 10B1
ID: 10B1
Name: BALA
Age: 21
Major: CSE

1. Add Student
2. View Student
3. Remove Student
4. Display All Students
5. Exit
Enter your choice: 3
Enter Student ID to remove: 10B1
Student removed successfully.
```

```
1. Add Student
2. View Student
3. Remove Student
4. Display All Students
5. Exit
Enter your choice: 4
No student records available.

1. Add Student
2. View Student
3. Remove Student
4. Display All Students
5. Exit
Enter your choice: 5
Exiting program...
PS C:\Users\sathw>
```

JUSTIFICATION TASK -9

1️⃣ Student Records – Dictionary

Stores data using unique ID for fast access.

2️⃣ Course Enrollment – List

Allows dynamic addition and removal of courses.

3️⃣ Faculty Information – Dictionary

Provides quick lookup using Faculty ID.

4️⃣ Campus Events – List

Stores multiple events and supports easy updates.

Task Description #10: Smart E-Commerce Platform – Data Structure Challenge

An e-commerce company wants to build a Smart Online Shopping System with:

1. Shopping Cart Management – Add and remove products dynamically.
2. Order Processing System – Orders processed in the order they are placed.
3. Top-Selling Products Tracker – Products ranked by sales count.
4. Product Search Engine – Fast lookup of products using product ID.
5. Delivery Route Planning – Connect warehouses and delivery locations.

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.

Expected Output:

• A table mapping feature → chosen data structure → justification.

• A functional Python program implementing the chosen feature

with comments and docstrings.

PROMPT:

#Generate a menu-driven Python program for an Order Processing System using a Queue (FIFO) with user input support in VS Code, including place_order, process_order, and display_orders methods, proper documentation, and continuous loop until exit.

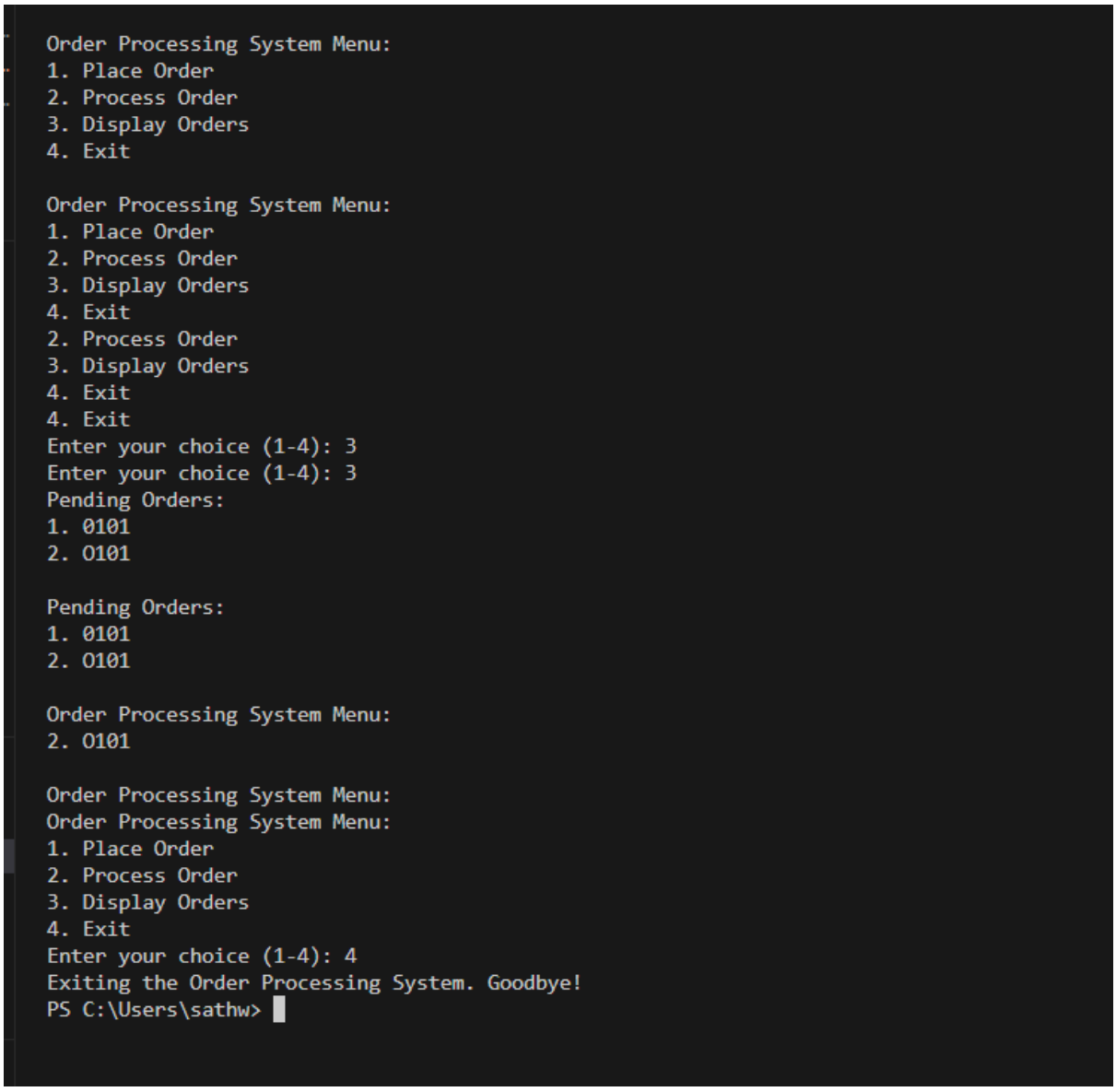
C: > Users > sathw > 11.1.1.py > ...

```python
#Generate a menu-driven Python program for an Order Processing System using a Queue (FIFO) with user input support in VS Code, including place_order, process_order
from collections import deque
class OrderProcessingSystem:
    def __init__(self):
        """Initialize the order processing system with an empty queue."""
        self.orders = deque()

    def place_order(self, order: str) -> None:
        """
        Place a new order in the queue.

        Args:
            order (str): The order to be placed.
        """
        self.orders.append(order)
        print(f"Order '{order}' has been placed.")

    def process_order(self) -> None:
        """
        Process the next order in the queue.

        Raises:
            IndexError: If there are no orders to process.
        """
        if not self.orders:
            print("No orders to process.")
            return
        order = self.orders.popleft()
        print(f"Order '{order}' has been processed.")

    def display_orders(self) -> None:
        """Display all pending orders in the queue."""
        if not self.orders:
            print("No pending orders.")
            return
        print("Pending Orders:")
```

```python
            return
        print("Pending Orders:")
        for idx, order in enumerate(self.orders, start=1):
            print(f"{idx}. {order}")
def main():
    system = OrderProcessingSystem()

    while True:
        print("\nOrder Processing System Menu:")
        print("1. Place Order")
        print("2. Process Order")
        print("3. Display Orders")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            order = input("Enter the order details: ")
            system.place_order(order)
        elif choice == '2':
            system.process_order()
        elif choice == '3':
            system.display_orders()
        elif choice == '4':
            print("Exiting the Order Processing System. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
if __name__ == "__main__":
    main()
```

OUTPUT:


Python Deb...
Python Deb...
Python Deb...
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): 1
Enter the order details: 0101
Order '0101' has been placed.
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): 1
Enter the order details: O101
Order 'O101' has been placed.
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): O102
Invalid choice. Please try again.
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): O102
Invalid choice. Please try again.
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): O102
Invalid choice. Please try again.


Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
2. Process Order
3. Display Orders
4. Exit
4. Exit
Enter your choice (1-4): 3
Enter your choice (1-4): 3
Pending Orders:
1. 0101
2. O101
Pending Orders:
1. 0101
2. O101
Order Processing System Menu:
2. O101
Order Processing System Menu:
Order Processing System Menu:
1. Place Order
2. Process Order
3. Display Orders
4. Exit
Enter your choice (1-4): 4
Exiting the Order Processing System. Goodbye!
PS C:\Users\sathw>

JUSTIFICATION:

Task 10 – Data Structure Mapping

Shopping Cart – Linked List

Allows dynamic add/remove of products easily.

Order Processing – Queue

Processes orders in FIFO order.

Top-Selling Products – Priority Queue

Retrieves highest selling products first.

Product Search Engine – Hash Table

Provides fast lookup using product ID.

Delivery Route Planning – Graph

Represents warehouses and locations as connected nodes.