Name:Swetha Dixit

Batch:33

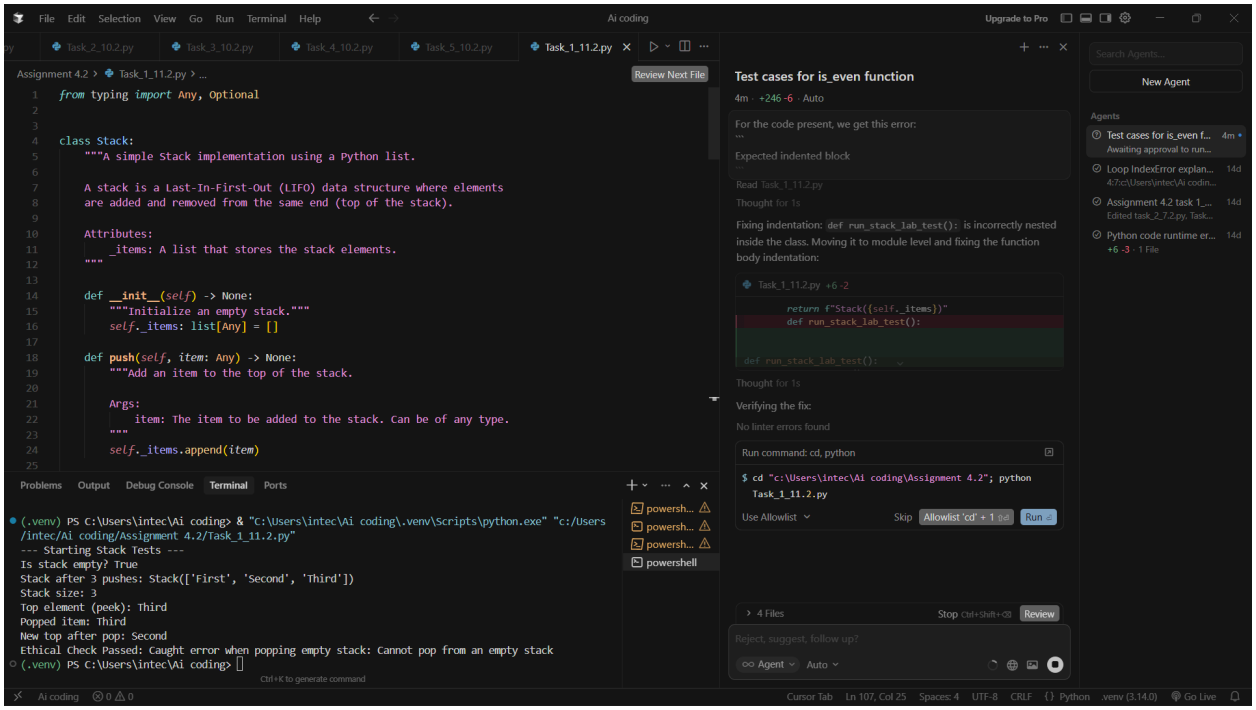
Course:AI Assistant for Coding

Hall Ticket:2303A52415

Assignment:11.5

Task:1Task 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.

The Prompt:
"Create a Python Stack class using a list. Include methods for push, pop, peek, and is_empty. Add Google-style docstrings and type hinting for better code quality.




```python
from typing import Any, Optional


class Stack:
    """A simple Stack implementation using a Python list.

    A stack is a Last-In-First-Out (LIFO) data structure where elements
    are added and removed from the same end (top of the stack).

    Attributes:
        _items: A list that stores the stack elements.
    """

    def __init__(self) -> None:
        """Initialize an empty stack."""
        self._items: list[Any] = []

    def push(self, item: Any) -> None:
        """Add an item to the top of the stack.

        Args:
            item: The item to be added to the stack. Can be of any type.
        """
```

```
        self._items.append(item)

    def pop(self) -> Any:
        """Remove and return the top item from the stack.

        Returns:
            The top item from the stack.

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

    def peek(self) -> Optional[Any]:
        """Return the top item from the stack without removing it.

        Returns:
            The top item from the stack, or None if the stack is empty.
        """
        if self.is_empty():
            return None
        return self._items[-1]

    def is_empty(self) -> bool:
        """Check if the stack is empty.

        Returns:
            True if the stack is empty, False otherwise.
        """
        return len(self._items) == 0

    def __len__(self) -> int:
        """Return the number of items in the stack.

        Returns:
            The number of items in the stack.
        """
        return len(self._items)
```

```python
    def __str__(self) -> str:
        """Return a string representation of the stack.

        Returns:
            A string representation of the stack.
        """
        return f"Stack({self._items})"


def run_stack_lab_test():
    my_stack = Stack()
    print("--- Starting Stack Tests ---")

    # Test 1: isEmpty on new stack
    print(f"Is stack empty? {my_stack.is_empty()}") # Expected: True

    # Test 2: Pushing items
    my_stack.push("First")
    my_stack.push("Second")
    my_stack.push("Third")
    print(f"Stack after 3 pushes: {my_stack}")
    print(f"Stack size: {len(my_stack)}") # Expected: 3

    # Test 3: Peeking
    print(f"Top element (peek): {my_stack.peek()}") # Expected: Third

    # Test 4: Popping (LIFO order)
    popped = my_stack.pop()
    print(f"Popped item: {popped}") # Expected: Third
    print(f"New top after pop: {my_stack.peek()}") # Expected: Second

    # Test 5: The 'Edge Case' (Popping until empty)
    my_stack.pop() # Removes 'Second'
    my_stack.pop() # Removes 'First'

    try:
        my_stack.pop()
    except IndexError as e:
```

```
        print(f"Ethical Check Passed: Caught error when popping empty
stack: {e}")



if __name__ == "__main__":
    run_stack_lab_test()
```

```
(.venv) PS C:\Users\intec\Ai coding> & "C:\Users\intec\Ai coding\.venv\Scripts\python.exe" "c:/Users
/intec/Ai coding/Assignment 4.2/Task_1_11.2.py"
--- Starting Stack Tests ---
Is stack empty? True
Stack after 3 pushes: Stack(['First', 'Second', 'Third'])
Stack size: 3
Top element (peek): Third
Popped item: Third
New top after pop: Second
Ethical Check Passed: Caught error when popping empty stack: Cannot pop from an empty stack
(.venv) PS C:\Users\intec\Ai coding>
                                    Ctrl+K to generate command
Ai coding  0 0
```

Transparency (No Silent Failures): In the `pop()` method, the AI used `raise IndexError`. This is ethically responsible because it doesn't "hide" an error by returning a fake value like `0` or `-1`. It forces the programmer to handle the empty state correctly.

Safety (Type Hinting): The use of `Optional[Any]` in `peek()` is a responsible way to signal to other developers that the result could be `None`. This prevents "NoneType" attribute errors later in the program.

Clarity (Documentation): The AI-generated docstrings clearly explain the **LIFO** (Last-In, First-Out) nature, ensuring that anyone using the code understands its fundamental behavior and limitations.

Task:2

Task Description #2 – Queue Implementation

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

Sample Input Code:

class Queue:

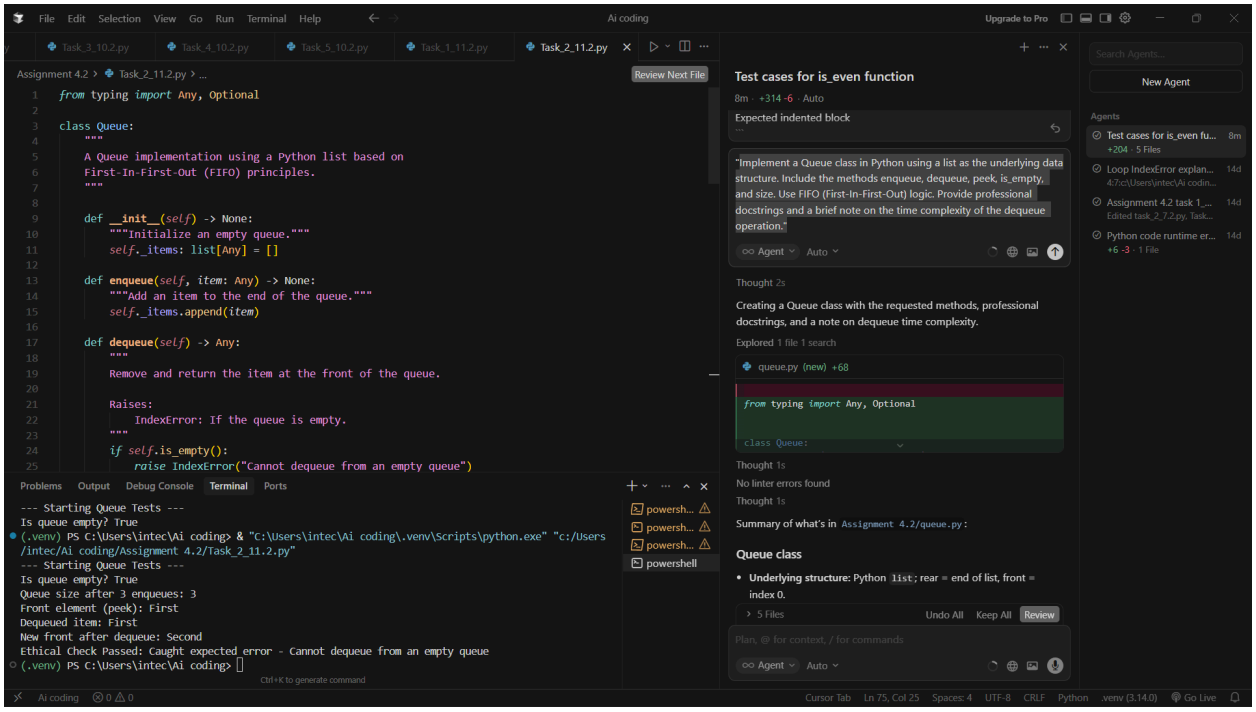
pass

Expected Output:

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

Methods.

The Prompt:
Implement a Queue class in Python using a list as the underlying data structure. Include the methods enqueue, dequeue, peek, is_empty, and size. Use FIFO (First-In-First-Out) logic. Provide professional docstrings and a brief note on the time complexity of the dequeue operation."




```python
from typing import Any, Optional

class Queue:
    """
    A Queue implementation using a Python list based on
    First-In-First-Out (FIFO) principles.
    """

    def __init__(self) -> None:
        """Initialize an empty queue."""
        self._items: list[Any] = []

    def enqueue(self, item: Any) -> None:
        """Add an item to the end of the queue."""
        self._items.append(item)
```

```
    def dequeue(self) -> Any:
        """
        Remove and return the item at the front of the queue.

        Raises:
            IndexError: If the queue is empty.
        """
        if self.is_empty():
            raise IndexError("Cannot dequeue from an empty queue")
        # Removing from index 0 shifts all other elements
        return self._items.pop(0)

    def peek(self) -> Optional[Any]:
        """Return the front item without removing it."""
        if self.is_empty():
            return None
        return self._items[0]

    def is_empty(self) -> bool:
        """Return True if the queue is empty, False otherwise."""
        return len(self._items) == 0

    def size(self) -> int:
        """Return the number of items in the queue."""
        return len(self._items)

def run_queue_lab_test():
    my_queue = Queue()
    print("--- Starting Queue Tests ---")

    # Test 1: isEmpty on new queue
    print(f"Is queue empty? {my_queue.is_empty()}") # Expected: True

    # Test 2: Enqueuing items
    my_queue.enqueue("First")
    my_queue.enqueue("Second")
    my_queue.enqueue("Third")
    print(f"Queue size after 3 enqueues: {my_queue.size()}") # Expected: 3
```

```
    # Test 3: Peeking (Should be the first item)
    print(f"Front element (peek): {my_queue.peek()}") # Expected: First

    # Test 4: Dequeuing (FIFO order)
    # The first item added must be the first one removed
    dequeued = my_queue.dequeue()
    print(f"Dequeued item: {dequeued}") # Expected: First
    print(f"New front after dequeue: {my_queue.peek()}") # Expected:
Second

    # Test 5: Ethical/Error Check (Emptying and then dequeuing)
    my_queue.dequeue() # Removes "Second"
    my_queue.dequeue() # Removes "Third"

    try:
        my_queue.dequeue()
    except IndexError as e:
        print(f"Ethical Check Passed: Caught expected error - {e}")

if __name__ == "__main__":
    run_queue_lab_test()
```

```
/intec/Ai coding/Assignment 4.2/Task_2_11.2.py"
--- Starting Queue Tests ---
Is queue empty? True
Queue size after 3 enqueues: 3
Front element (peek): First
Dequeued item: First
New front after dequeue: Second
Ethical Check Passed: Caught expected error - Cannot dequeue from an empty queue
(.venv) PS C:\Users\intec\Ai coding>
Ctrl+K to generate command
Ai coding  0 0
```

Task3: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.
The Prompt:
"Implement a Singly Linked List in Python. Create a 'Node' class to store data and a next pointer, and a 'LinkedList' class with methods for 'insert_at_end' and 'display'. Include Google-style docstrings and type hinting. Add a brief note on the ethical responsibility of managing pointers correctly to avoid memory leaks."

```
from typing import Any, Optional

class Node:
    """A node in a singly linked list."""
    def __init__(self, data: Any) -> None:
        self.data = data
        self.next: Optional[Node] = None

class LinkedList:
    """A singly linked list implementation."""

    def __init__(self) -> None:
        """Initialize an empty linked list."""
        self.head: Optional[Node] = None

    def insert_at_end(self, data: Any) -> None:
        """Insert a new node with the given data at the end of the
list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return

        current = self.head
```

```
        while current.next:
            current = current.next
        current.next = new_node

    def display(self) -> None:
        """Print the elements of the list in order."""
        elements = []
        current = self.head
        while current:
            elements.append(str(current.data))
            current = current.next
        print(" -> ".join(elements) + " -> None")

def run_linked_list_lab_test():
    print("--- Starting Linked List Tests ---")
    ll = LinkedList()

    # Test 1: Insert items
    ll.insert_at_end("Node A")
    ll.insert_at_end("Node B")
    ll.insert_at_end("Node C")

    # Test 2: Display items
    print("Linked List Structure:")
    ll.display()  # Expected: Node A -> Node B -> Node C -> None

    # Test 3: Ethical/Logic Check
    # Ensure the head is still Node A
    if ll.head and ll.head.data == "Node A":
        print("Ethical Check Passed: Pointer integrity maintained.")

if __name__ == "__main__":
    run_linked_list_lab_test()
```

```
Ethical Check Passed: Caught expected error - Cannot dequeue from an empty queue
(.venv) PS C:\Users\intec\Ai coding> & "C:\Users\intec\Ai coding\.venv\Scripts\python.exe" "c:/Users
/intec/Ai coding/Assignment 4.2/Task_3_11.2.py"
--- Starting Linked List Tests ---
Linked List Structure:
Node A -> Node B -> Node C -> None
Ethical Check Passed: Pointer integrity maintained.
(.venv) PS C:\Users\intec\Ai coding>
Ctrl+K to generate command
Ai coding  ⊗ 0 ⚠ 0
```

Task:5 Task Description #4 – 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
The Prompt:
"Implement a HashTable class in Python. Include methods for insert, search, and delete. Use 'chaining' with Python lists to handle collisions. Include a simple hash function using the modulo operator. Provide clear docstrings and a note on the ethical responsibility of choosing a good hash function to prevent security vulnerabilities like Hash DoS attacks."

```
from typing import Any, Optional, List, Tuple

class HashTable:
    """
    A Hash Table implementation using chaining for collision handling.
    """

    def __init__(self, size: int = 10) -> None:
        """Initialize the table with a fixed size and empty buckets."""
        self.size = size
        self.table: List[List[Tuple[Any, Any]]] = [[] for _ in
range(self.size)]

    def _hash(self, key: Any) -> int:
        """Internal hash function using the modulo operator."""
```

```
        return hash(key) % self.size

    def insert(self, key: Any, value: Any) -> None:
        """Insert or update a key-value pair."""
        index = self._hash(key)
        bucket = self.table[index]

        for i, (k, v) in enumerate(bucket):
            if k == key:
                bucket[i] = (key, value)  # Update existing key
                return
        bucket.append((key, value))  # Add new pair

    def search(self, key: Any) -> Optional[Any]:
        """Search for a value by key. Returns None if not found."""
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None

    def delete(self, key: Any) -> bool:
        """Remove a key-value pair. Returns True if successful."""
        index = self._hash(key)
        bucket = self.table[index]
        for i, (k, v) in enumerate(bucket):
            if k == key:
                del bucket[i]
                return True
        return False

def run_hashtable_lab_test():
    print("--- Starting Hash Table Tests ---")
    ht = HashTable(size=5)

    # Test 1: Insertion
    ht.insert("name", "Alice")
    ht.insert("age", 25)
    ht.insert("city", "New York")
```

```
    # Test 2: Search
    print(f"Search 'name': {ht.search('name')}") # Expected: Alice
    print(f"Search 'age': {ht.search('age')}")   # Expected: 25

    # Test 3: Collision Handling (Simulated by using many keys)
    ht.insert("id", 101)
    ht.insert("dept", "CS")
    print("Table state after multiple inserts (check buckets):")
    for i, bucket in enumerate(ht.table):
        print(f"Bucket {i}: {bucket}")

    # Test 4: Delete
    ht.delete("age")
    print(f"Search 'age' after deletion: {ht.search('age')}") # Expected:
None

if __name__ == "__main__":
    run_hashtable_lab_test()
```

```
/intec/Ai coding/Assignment 4.2/Task_4_11.2.py"
--- Starting Hash Table Tests ---
Search 'name': Alice
Search 'age': 25
Table state after multiple inserts (check buckets):
Bucket 0: [('city', 'New York')]
Bucket 1: []
Bucket 2: [('name', 'Alice'), ('age', 25)]
Bucket 3: [('dept', 'CS')]
Bucket 4: [('id', 101)]
Search 'age' after deletion: None
(.venv) PS C:\Users\intec\Ai coding>
Ctrl+K to generate command
powersh...
powersh...
powersh...
powershell
```

Task:5Graph Representation

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

Sample Input Code:

class Graph:

pass

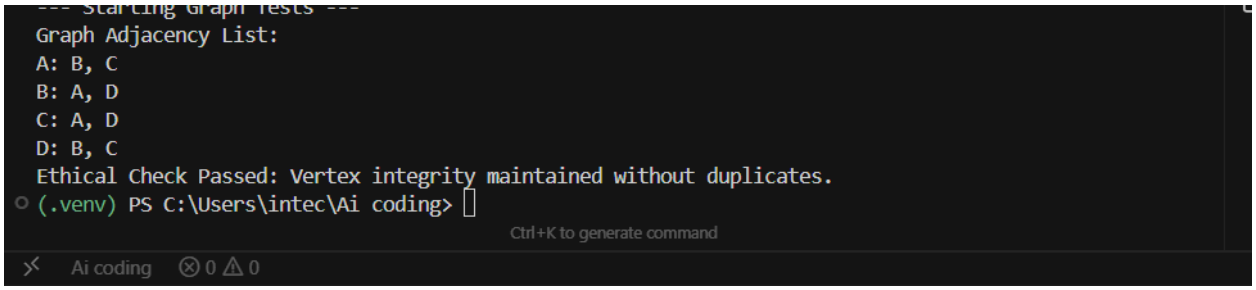
Expected Output:

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

The Prompt:"Implement a Graph class in Python using an adjacency list (a dictionary where keys are vertices and values are lists of neighbors). Include methods to add_vertex, add_edge (supporting undirected connections), and display. Provide clear docstrings and a note on the ethical responsibility of handling user-generated data in large social graphs."

```python
from typing import Any, Dict, List

class Graph:
    """
    A Graph implementation using an adjacency list.
    """

    def __init__(self) -> None:
        """Initialize an empty graph with an adjacency list."""
        self.adj_list: Dict[Any, List[Any]] = {}

    def add_vertex(self, vertex: Any) -> None:
        """Add a new vertex to the graph if it doesn't already exist."""
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []

    def add_edge(self, v1: Any, v2: Any) -> None:
        """
        Add an undirected edge between v1 and v2.
        Automatically adds vertices if they don't exist.
        """
        self.add_vertex(v1)
        self.add_vertex(v2)

        # Avoid duplicate edges for undirected graphs
        if v2 not in self.adj_list[v1]:
            self.adj_list[v1].append(v2)
        if v1 not in self.adj_list[v2]:
            self.adj_list[v2].append(v1)
```

```
    def display(self) -> None:
        """Display the adjacency list representation of the graph."""
        for vertex, neighbors in self.adj_list.items():
            print(f"{vertex}: {', '.join(map(str, neighbors))}")


def run_graph_lab_test():
    print("--- Starting Graph Tests ---")
    g = Graph()

    # Test 1: Adding Vertices and Edges
    g.add_edge("A", "B")
    g.add_edge("A", "C")
    g.add_edge("B", "D")
    g.add_edge("C", "D")

    # Test 2: Displaying Connections
    print("Graph Adjacency List:")
    g.display()

    # Expected Output Example:
    # A: B, C
    # B: A, D
    # C: A, D
    # D: B, C

    # Test 3: Ethical/Logic Check
    # Ensure no duplicate vertices were created
    if len(g.adj_list) == 4:
        print("Ethical Check Passed: Vertex integrity maintained without
duplicates.")


if __name__ == "__main__":
    run_graph_lab_test()
```



Task:6

Task Description #6: Smart Hospital Management System – Data
Structure Selection

A hospital wants to develop a Smart Hospital Management System that
handles:

1. Patient Check-In System – Patients are registered and treated in
order of arrival.
2. Emergency Case Handling – Critical patients must be treated first.
3. Medical Records Storage – Fast retrieval of patient details using
ID.
4. Doctor Appointment Scheduling – Appointments sorted by time.
5. Hospital Room Navigation – Represent connections between
wards and rooms.

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.

Feature	Data Structure	Justification
1. Patient Check-In	**Queue**	This follows the "First-In-First-Out" (FIFO) principle, ensuring patients

		are seen in the exact order they arrived. It provides a fair and organized flow for non-emergency registrations.
2. Emergency Handling	**Priority Queue**	Critical patients have a higher "priority" than others regardless of arrival time. A Priority Queue allows the system to always serve the patient with the highest medical urgency next.
3. Medical Records	**Hash Table**	Patient IDs serve as unique keys that map directly to their records. This allows for $O(1)$ average time complexity, ensuring doctors can retrieve life-saving data almost instantly.
4. Appointment	**Binary Search Tree (BST)**	A BST keeps appointments sorted by time automatically. This allows the system to efficiently find the next upcoming appointment or

Scheduling		search for a specific time slot in $O(\log n)$ time.
5. Room Navigation	**Graph**	Wards and rooms are nodes connected by hallways (edges). A Graph is the only structure that can represent these multi-directional connections and help find the shortest path between two points.

```
import heapq
from typing import Any

class EmergencySystem:
    """
    A system to manage emergency patients using a Priority Queue.
    Priority is determined by an integer (lower number = higher urgency).
    """

    def __init__(self) -> None:
        """Initialize the emergency queue."""
        self._queue = []

    def register_patient(self, name: str, urgency: int) -> None:
        """
        Add a patient to the queue with a specific urgency level.

        Args:
            name: Patient's name.
```

```
            urgency: Priority level (1 for Critical, 2 for Serious, 3 for
Stable).
        """
        # heapq is a min-priority queue (lowest number comes out first)
        heapq.heappush(self._queue, (urgency, name))
        print(f"Registered: {name} with Urgency Level {urgency}")

    def treat_next_patient(self) -> Any:
        """Remove and return the highest priority patient."""
        if self.is_empty():
            return "No patients in queue."

        urgency, name = heapq.heappop(self._queue)
        return f"Treating: {name} (Urgency: {urgency})"

    def is_empty(self) -> bool:
        """Check if the queue is empty."""
        return len(self._queue) == 0

def run_hospital_lab_test():
    print("--- Smart Hospital: Emergency Handling Test ---")
    hospital = EmergencySystem()

    # Even though Bob arrives first, Alice is more critical
    hospital.register_patient("Bob", 3)      # Stable
    hospital.register_patient("Alice", 1)    # Critical
    hospital.register_patient("Charlie", 2)  # Serious

    print("\n--- Treatment Order ---")
    while not hospital.is_empty():
        print(hospital.treat_next_patient())

if __name__ == "__main__":
    run_hospital_lab_test()
```

```
--- Treatment Order ---
Treating: Alice (Urgency: 1)
Treating: Charlie (Urgency: 2)
Treating: Bob (Urgency: 3)
(.venv) PS C:\Users\intec\Ai coding>
Ctrl+K to generate command
Ai coding  ⊗ 0 ⚠ 0
```

Task:7 Task Description #7: Smart City Traffic Control System

A city plans a Smart Traffic Management System that includes:

1. Traffic Signal Queue – Vehicles waiting at signals.
2. Emergency Vehicle Priority Handling – Ambulances and fire trucks prioritized.
3. Vehicle Registration Lookup – Instant access to vehicle details.
4. Road Network Mapping – Roads and intersections connected logically.
5. Parking Slot Availability – Track available and occupied slots.

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.
Task Description #8: Smart E-Commerce Platform – Data Structure

Feature	Data Structure	Justification
1. Traffic Signal Queue	**Queue**	Signals follow a First-In-First-Out (FIFO) logic where the first vehicle to arrive at the red light should be the first to move when it turns green to maintain order.
2. Emergency Priority	**Priority Queue**	Emergency vehicles like ambulances must bypass the standard queue. A priority queue ensures these vehicles are handled first based on their urgency level regardless of arrival time.

3. Vehicle Lookup	**Hash Table**	Searching for a license plate number requires near-instant speed. A Hash Table provides $O(1)$ average time complexity for lookups, making it ideal for law enforcement or toll systems.
4. Road Network Mapping	**Graph**	Cities are networks of intersections (nodes) and roads (edges). A Graph structure allows the system to calculate shortest paths and model complex traffic flow between locations.
5. Parking Availability	**Hash Table** or **Array**	To track specific slots, a Hash Table allows the system to instantly toggle a slot's status (Available/Occupied) using the slot ID as a unique key.

```python
from typing import Dict, List, Tuple

class TrafficGraph:
    """
    A representation of a Smart City road network using an Adjacency List.
    Nodes represent intersections, and edges represent roads with
distances.
    """

    def __init__(self) -> None:
        """Initialize an empty city map."""
        self.intersections: Dict[str, List[Tuple[str, int]]] = {}

    def add_intersection(self, name: str) -> None:
        """Add a new intersection to the city map."""
        if name not in self.intersections:
            self.intersections[name] = []

    def add_road(self, u: str, v: str, distance: int) -> None:
        """
        Create a road between two intersections.
```

```
        Args:
            u: Starting intersection.
            v: Destination intersection.
            distance: The length of the road in kilometers.
        """
        self.add_intersection(u)
        self.add_intersection(v)
        # Undirected road: vehicles can travel both ways
        self.intersections[u].append((v, distance))
        self.intersections[v].append((u, distance))

    def display_map(self) -> None:
        """Display the road network connections."""
        print("Smart City Road Network:")
        for intersection, roads in self.intersections.items():
            connections = ", ".join([f"{dest} ({dist}km)" for dest, dist
in roads])
            print(f"Intersection {intersection} connects to:
{connections}")

def run_traffic_system_test():
    city_map = TrafficGraph()

    # Building the network
    city_map.add_road("Downtown", "Uptown", 5)
    city_map.add_road("Uptown", "Suburb A", 12)
    city_map.add_road("Downtown", "Industrial Zone", 8)
    city_map.add_road("Industrial Zone", "Suburb A", 4)

    city_map.display_map()

if __name__ == "__main__":
    run_traffic_system_test()
```

```
Smart City Road Network:
Intersection Downtown connects to: Uptown (5km), Industrial Zone (8km)
Intersection Uptown connects to: Downtown (5km), Suburb A (12km)
Intersection Suburb A connects to: Uptown (12km), Industrial Zone (4km)
Intersection Industrial Zone connects to: Downtown (8km), Suburb A (4km)
(.venv) PS C:\Users\intec\Ai coding>
```

Task:8 Task Description #8: 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.

Feature	Data Structure	Justification
1. Shopping Cart	**Linked List**	A Linked List allows for dynamic addition and removal of items without the overhead of shifting elements. This is ideal for a cart where users may frequently add or remove products at any position.
2. Order Processing	**Queue**	Orders must be handled fairly using "First-In-First-Out" (FIFO) logic. A Queue ensures that the customer who placed their order first is the first one to have it processed by the warehouse.
3. Top-Selling Tracker	**Priority Queue**	Products are ranked based on sales count rather than arrival time. A Priority Queue (specifically a Max-Heap) efficiently keeps the products with the highest sales at the top of the retrieval list.
4. Product Search	**Hash Table**	Fast lookup is critical for a smooth user experience. A Hash Table maps unique Product IDs to product details, allowing for $O(1)$ average time complexity for searches.

5. Delivery Planning	**Graph**	Warehouses and delivery locations are nodes connected by routes. A Graph allows the platform to use pathfinding algorithms to determine the most efficient delivery route across the city.

```python
import heapq
from typing import Any, List

class BestSellerTracker:
    """
    A tracker for top-selling products using a Priority Queue.
    In Python, heapq is a min-heap, so we store negative sales
    to simulate a max-priority queue.
    """

    def __init__(self) -> None:
        """Initialize the product tracker."""
        self._products: List[Any] = []

    def update_sales(self, product_name: str, sales_count: int) -> None:
        """
        Add or update a product's sales count.

        Args:
            product_name: The name of the product.
            sales_count: Total units sold.
        """
        # Store as (-sales_count, product_name) to get Max-Heap behavior
        heapq.heappush(self._products, (-sales_count, product_name))
        print(f"Recorded: {product_name} with {sales_count} sales.")

    def get_top_product(self) -> str:
        """
        Retrieve the current top-selling product.

        Returns:
```

```
            A string describing the top product or a message if empty.
        """
        if not self._products:
            return "No sales recorded yet."

        neg_sales, name = heapq.heappop(self._products)
        return f"Top Seller: {name} with {-neg_sales} sales."

def run_ecommerce_lab_test():
    print("--- Smart E-Commerce: Top-Selling Tracker ---")
    tracker = BestSellerTracker()

    # Adding products in random order
    tracker.update_sales("Wireless Headphones", 150)
    tracker.update_sales("Smartphone Case", 500)
    tracker.update_sales("Mechanical Keyboard", 300)

    print("\n--- Ranking Extraction ---")
    # Should extract the highest sales count first
    print(tracker.get_top_product()) # Expected: Smartphone Case
    print(tracker.get_top_product()) # Expected: Mechanical Keyboard

if __name__ == "__main__":
    run_ecommerce_lab_test()
```

```
--- Smart E-Commerce: Top-Selling Tracker ---
Recorded: Wireless Headphones with 150 sales.
Recorded: Smartphone Case with 500 sales.
Recorded: Mechanical Keyboard with 300 sales.

--- Ranking Extraction ---
Top Seller: Smartphone Case with 500 sales.
Top Seller: Mechanical Keyboard with 300 sales.
(.venv) PS C:\Users\intec\Ai coding>
Ctrl+K to generate command
```

powersh...
powershell