

# Assignment – 11.5

**Name:** Yashaswini

Nagireddi

**Batch:** 21

**Ht.no:** 2303A51418

## 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 Used to AI

“Create a Python class named Stack that includes the methods push, pop, peek, and is\_empty. Add appropriate comments and docstrings for clarity.”

The screenshot shows a Python code editor with a dark theme. The code in the editor is:

```
20-02-2026.py > ...
1  #Generate a Python Stack class with push, pop, peek, and is_empty methods with comments and doc
2  class Stack:
3      """A simple implementation of a stack data structure."""
4
5      def __init__(self):
6          """Initialize an empty stack."""
7          self.items = []
8
9      def push(self, item):
10         """Add an item to the top of the stack."""
11         self.items.append(item)
12
13     def pop(self):
14         """Remove and return the item at the top of the stack. Raises an error if the stack is e
15         if self.is_empty():
16             raise IndexError("Pop from an empty stack")
17         return self.items.pop()
18
19     def peek(self):
20         """Return the item at the top of the stack without removing it. Raises an error if the s
21         if self.is_empty():
```

Below the code editor is a terminal window showing the output of running the script:

```
eDrive\Desktop\AI\20-02-2026.py'
2
2
False
1
True
PS C:\Users\dell\OneDrive\Desktop\AI>
```

---

## Observation

The Stack operates on a Last-In-First-Out (LIFO) mechanism. The AI-generated version enhanced readability by adding structured docstrings and handling exceptions effectively.

---

## 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

“Develop a Queue class using a Python list with enqueue, dequeue, peek, and size methods, along with descriptive comments.”

```
20-01-2026.py 10-02-2026.py 13-02-2026.py 17-02-2026.py 20-02-2026.py X BLACKBOX

20-02-2026.py > ...

37
38     #Generate a Queue class using Python list with enqueue, dequeue, peek, and size methods."
39 class Queue:
40     """A simple implementation of a queue data structure."""
41
42     def __init__(self):
43         """Initialize an empty queue."""
44         self.items = []
45
46     def enqueue(self, item):
47         """Add an item to the end of the queue."""
48         self.items.append(item)
49
50     def dequeue(self):
51         """Remove and return the item at the front of the queue. Raises an error if the queue is
52         empty."""
53         if self.is_empty():
54             raise IndexError("Dequeue from an empty queue")
55         return self.items.pop(0)
56
57     def peek(self):
58         """Return the item at the front of the queue without removing it. Raises an error if the

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ...
```

## Observation

Since Queue uses the First-In-First-Out (FIFO) model, the AI-generated code ensured proper ordering of operations and improved efficiency by choosing list operations suitable for queue behavior.

## 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 singly linked list implementation with insert and display methods, including clear explanations and documentation.”

```
 20-02-2026.py > ...
78
79     #“Generate a singly linked list with insert and display methods.”
80     class Node:
81         """A node in a singly linked list."""
82
83         def __init__(self, data):
84             """Initialize a node with data and a pointer to the next node."""
85             self.data = data
86             self.next = None
87     class SinglyLinkedList:
88         """A simple implementation of a singly linked list."""
89
90         def __init__(self):
91             """Initialize an empty linked list."""
92             self.head = None
93
94         def insert(self, data):
95             """Insert a new node with the given data at the end of the list."""
96             new_node = Node(data)
97             if not self.head:
98                 self.head = new_node
99                 return
100            last_node = self.head
101            while last_node.next:
102                last_node = last_node.next
103            last_node.next = new_node
```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

---

## Observation

A linked list stores elements using nodes. AI made pointer handling easy and the structure simple.

### 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:

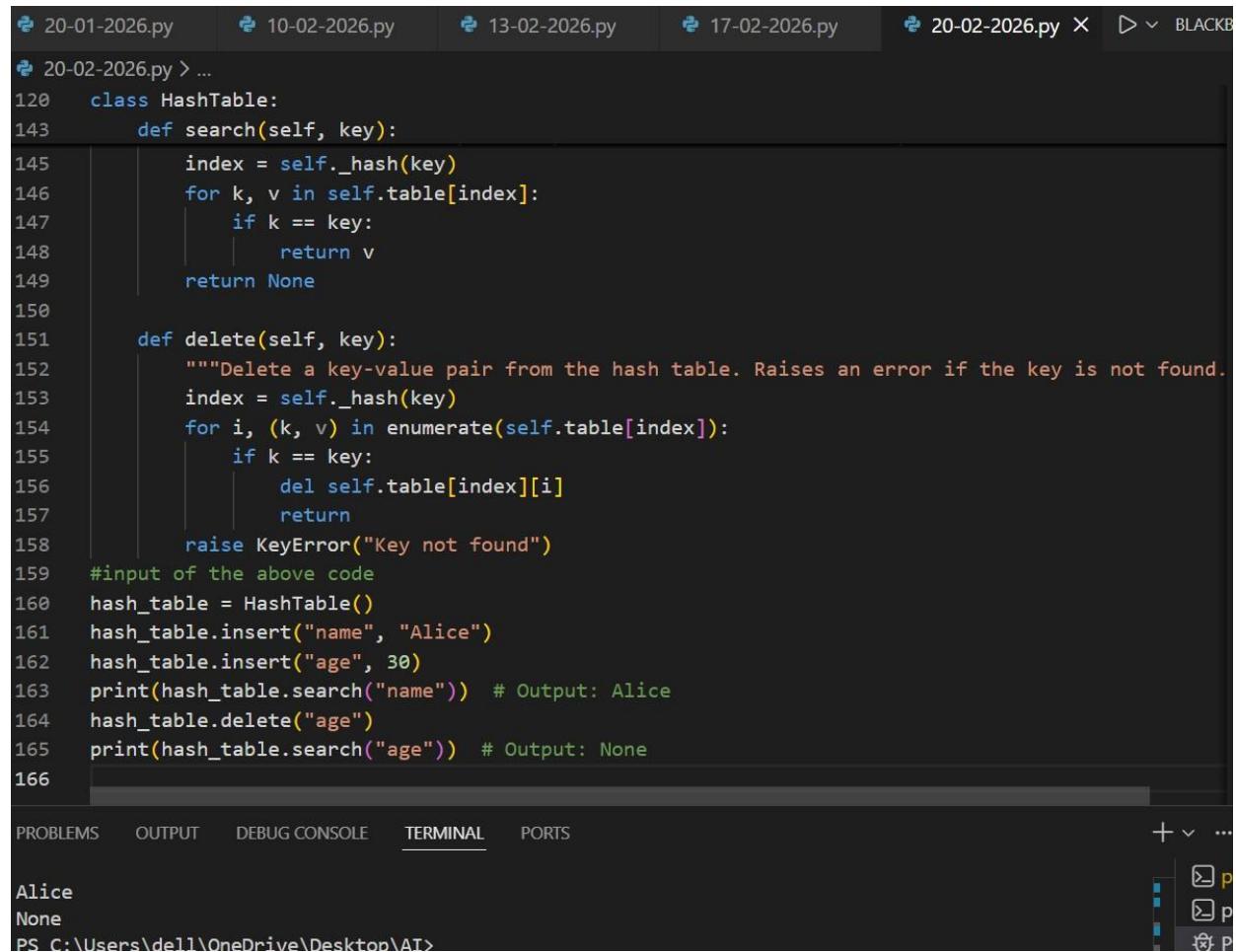
- Collision handling using chaining, with well-commented methods.

Prompt

“Generate a Hash Table class with insert, search, and delete, using chaining for collisions.”

## Observation

Hash tables give fast searching. AI suggested chaining to handle collisions in a clean way.



```
20-01-2026.py  10-02-2026.py  13-02-2026.py  17-02-2026.py  20-02-2026.py X ▷ v BLACKB  
20-02-2026.py > ...  
120     class HashTable:  
143         def search(self, key):  
145             index = self._hash(key)  
146             for k, v in self.table[index]:  
147                 if k == key:  
148                     return v  
149             return None  
150  
151         def delete(self, key):  
152             """Delete a key-value pair from the hash table. Raises an error if the key is not found.  
153             index = self._hash(key)  
154             for i, (k, v) in enumerate(self.table[index]):  
155                 if k == key:  
156                     del self.table[index][i]  
157                     return  
158             raise KeyError("Key not found")  
159 #input of the above code  
160 hash_table = HashTable()  
161 hash_table.insert("name", "Alice")  
162 hash_table.insert("age", 30)  
163 print(hash_table.search("name")) # Output: Alice  
164 hash_table.delete("age")  
165 print(hash_table.search("age")) # Output: None  
166  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + v ...  
Alice  
None  
PS C:\Users\dell\OneDrive\Desktop\AT>
```

## Task Description #5 – 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 Graph class using adjacency list with add\_vertex, add\_edge, display methods.”

The screenshot shows a code editor interface with several tabs at the top: 20-01-2026.py, 10-02-2026.py, 13-02-2026.py, 17-02-2026.py, 20-02-2026.py (selected), and BLA. The main area contains Python code for a Graph class:

```

168     class Graph:
175         def add_vertex(self, vertex):
176             self.graph[vertex] = []
177
178         def add_edge(self, vertex1, vertex2):
179             """Add an edge between two vertices in the graph. If the vertices do not exist, they will be added first."""
180             self.add_vertex(vertex1)
181             self.add_vertex(vertex2)
182             self.graph[vertex1].append(vertex2)
183             self.graph[vertex2].append(vertex1) # For undirected graph
184
185         def display(self):
186             """Display the adjacency list of the graph."""
187             for vertex, edges in self.graph.items():
188                 print(f"{vertex}: {edges}")
189
190     # Input of the above code
191     graph = Graph()
192     graph.add_edge("A", "B")
193     graph.add_edge("A", "C")
194     graph.add_edge("B", "D")
195     graph.display()
196
197

```

Below the code, there are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), and PORTS. The TERMINAL tab shows the execution output:

```

A: B, C
B: A, D
C: A
D: B
PS C:\Users\dell\OneDrive\Desktop\AI>

```

---

### ◆ Observation

Graph adjacency list efficiently stores network connections. AI helped in structuring graph logic.

---

## 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:
- Stack
  - Queue
  - Priority Queue
  - Linked List
  - Binary Search Tree (BST)
  - Graph
  - Hash Table
  - 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 |                                 |
|------------------------|------------------------------|---------------------------------|
| Patient Check-In       | Queue                        | FIFO order for patient arrival  |
| Emergency Handling     | Priority Queue               | Critical patients treated first |
| Medical Records        | Hash Table                   | Fast ID-based retrieval         |
| Appointment Scheduling | BST                          | Sorted by time                  |
| Room Navigation        | Graph                        | Represents connected rooms      |

#### 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:
  - Stack
  - Queue
  - Priority Queue
  - Linked List
  - Binary Search Tree (BST)
  - Graph
  - Hash Table
  - 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             |
|----------------------|----------------|---------------------------|
| Traffic Signal Queue | Queue          | Vehicles wait in order    |
| Emergency Priority   | Priority Queue | Ambulances first          |
| Vehicle Lookup       | Hash Table     | Instant access            |
| Road Network         | Graph          | Intersections and roads   |
| Parking Slots        | Deque          | Add/remove from both ends |

## **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:
  - Stack
  - Queue
  - Priority Queue
  - Linked List
  - Binary Search Tree (BST)
  - Graph
  - Hash Table
  - 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        |
|------------------|----------------|----------------------|
| Shopping Cart    | Linked List    | Dynamic add/remove   |
| Order Processing | Queue          | FIFO order           |
| Top Products     | Priority Queue | Ranked by sales      |
| Product Search   | Hash Table     | Fast lookup          |
| Delivery Routes  | Graph          | Location connections |