

AI Assisted Coding

Assignment 11.3

Name: ch. koushik

Hall ticket no: 2303A51938

Batch no: 19

Task 1: Smart Contact Manager (Arrays & Linked Lists)

Prompt:

Generate Python code to implement a Contact Manager system using:

- Array (Python list)
- Linked List

The system must support:

- Add contact
- Search contact
- Delete contact

Use meaningful class names, proper methods, and include comments.

Code & Output (Arrays):

```
Assignment_11.3.py
1 "Task 1: Smart Contact Manager (Arrays)"
2 # Generate Python code to implement a Contact Manager system using:
3 # * Array (Python List)
4 # * Linked List
5 # The system must support:
6 # * Add contact
7 # * Search contact
8 # * Delete contact
9 # Use meaningful class names, proper methods, and include comments.
10
11 class ContactManagerArray:
12     def __init__(self):
13         self.contacts = [] # Using a List to store contacts
14     def add_contact(self, name, phone):
15         """Add a new contact to the list."""
16         self.contacts.append({'name': name, 'phone': phone})
17     def search_contact(self, name):
18         """Search for a contact by name."""
19         for contact in self.contacts:
20             if contact['name'] == name:
21                 return contact
22         return None # Contact not found
23     def delete_contact(self, name):
24         """Delete a contact by name."""
25         for i, contact in enumerate(self.contacts):
26             if contact['name'] == name:
27                 del self.contacts[i]
28                 return True # Contact deleted
29         return False # Contact not found
30
31 manager = ContactManagerArray()
32 manager.add_contact("Alice", "123-456-7890")
33 manager.add_contact("Bob", "987-654-3210")
34 print(manager.search_contact("Alice")) # Output: {'name': 'Alice', 'phone': '123-456-7890'}
35 print(manager.delete_contact("Bob")) # Output: True
36 print(manager.search_contact("Bob")) # Output: None
```

PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
{'name': 'Alice', 'phone': '123-456-7890'}
True
None

Explanation (Arrays):

This implementation uses a Python list to store contact dictionaries. Adding contacts is efficient ($O(1)$ average). Searching and deletion require linear traversal ($O(n)$). The array

approach is simple and easy to implement but less efficient for frequent deletions in large datasets.

Code & Output (Linked-Lists):

The screenshot shows a code editor with two panes. The left pane displays the Python code for a linked list implementation named `ContactManagerLinkedList`. The right pane shows the command prompt and the output of running the code. The code includes functions for adding, searching, and deleting contacts from the linked list.

```
Assignment_11.3.py
Assignment_11.3.py > delete_contact
File Edit Selection View Go Run Terminal Help ← →
Assignment_11.3
Code
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
{'name': 'Alice', 'phone': '123-456-7890'}
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
{'name': 'Alice', 'phone': '123-456-7890'}
True
None
{'name': 'Charlie', 'phone': '555-555-5555'}
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
Assignment_11.3.py
Assignment_11.3.py > delete_contact
File Edit Selection View Go Run Terminal Help ← →
Assignment_11.3
Code
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
{'name': 'Alice', 'phone': '123-456-7890'}
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
{'name': 'Alice', 'phone': '123-456-7890'}
True
None
{'name': 'Charlie', 'phone': '555-555-5555'}
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
```

Explanation (Linked-Lists):

The linked list implementation allows dynamic memory allocation. Insertion at the beginning is O(1). Searching and deletion are O(n). Unlike arrays, linked lists avoid shifting elements during deletion. However, they require extra memory for pointers and are slightly more complex to implement.

Comparision (Arrays VS Linked-Lists):

- Insertion Efficiency: Linked List (O(1) at head) is better than array when frequent insertions occur.
- Deletion Efficiency: Linked List avoids shifting elements.
- Search Efficiency: Both require O(n).
- Memory Usage: Array is more memory-efficient.

Task 2: Library Book Search System (Queue & Priority Queue)

Prompt:

Generate Python code to implement:

- A Queue (FIFO)
- A Priority Queue prioritizing faculty requests over student requests

Include enqueue and dequeue methods.

Code & Output (Queue):

The screenshot shows a code editor with two panes. The left pane displays the Python code for a Queue class:

```
82 "Task 2: Library Book Search System (Queue)"
83 # Generate Python code to implement:
84 # A Queue (FIFO)
85 # A Priority Queue prioritizing faculty requests over students
86 # Include enqueue and dequeue methods.
87 from collections import deque
88 class Queue:
89     def __init__(self):
90         self.queue = deque() # Using deque for efficiency
91     def enqueue(self, item):
92         '''Add an item to the end of the queue.'''
93         self.queue.append(item)
94     def dequeue(self):
95         '''Remove and return the item at the front of the queue.'''
96         if not self.is_empty():
97             return self.queue.popleft()
98         return None # Queue is empty
99     def is_empty(self):
100        '''Check if the queue is empty.'''
101        return len(self.queue) == 0
102 #Example Usage
103 library_queue = Queue()
104 library_queue.enqueue("Student Request: Book A")
105 library_queue.enqueue("Faculty Request: Book B")
106 print(library_queue.dequeue()) # Output: "Student Request: Book A"
107
108
```

The right pane shows the terminal output of running the script:

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
Student Request: Book A
Faculty Request: Book B
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
```

Explanation (Queue):

The queue follows FIFO (First In, First Out). Requests are processed in the order they arrive. This is suitable for standard book request management.

Code & Output (Priority Queue):

The screenshot shows a code editor with two panes. The left pane displays the Python code for a Priority Queue class using a heap:

```
112 "Task 2: Library Book Search System (Priority Queue)"
113 import heapq
114 class PriorityQueue:
115     def __init__(self):
116         self.queue = [] # Using a List to store the priority queue
117     def enqueue(self, item, priority):
118         '''Add an item with a given priority to the queue.'''
119         heapq.heappush(self.queue, (priority, item))
120     def dequeue(self):
121         '''Remove and return the item with the highest priority (lowest number).'''
122         if not self.is_empty():
123             return heapq.heappop(self.queue)[1] # Return the item, not the priority
124         return None # Queue is empty
125     def is_empty(self):
126         '''Check if the priority queue is empty.'''
127         return len(self.queue) == 0
128 # Example Usage
129 priority_queue = PriorityQueue()
130 priority_queue.enqueue("Student Request: Book A", priority=2) # Lower priority
131 priority_queue.enqueue("Faculty Request: Book B", priority=1) # Higher priority
132 print(priority_queue.dequeue()) # Output: "Faculty Request: Book B"
```

The right pane shows the terminal output of running the script:

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py
2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
Student Request: Book A
Faculty Request: Book B
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
```

Explanation (Priority Queue):

The priority queue uses a heap. Faculty requests are assigned higher priority (lower numeric value). This ensures faculty members are served before students.

Task 3: Emergency Help Desk (Stack)

Prompt:

Generate Python code to implement a Stack for managing support tickets with push, pop, peek, is_empty methods.

Code & Output:

The screenshot shows a code editor with two panes. The left pane displays the Python code for a Stack class, labeled "Assignment_11.3.py". The right pane shows the terminal output of running the script. The code defines a Stack class with methods for push, pop, peek, and is_empty. The terminal shows the stack operations being performed: pushing two support tickets, popping them, and checking if it's empty.

```
132 # Task 3: Emergency Help Desk (Stack)
133 # Generate Python code to implement a Stack for managing support
134 # tickets with push, pop, peek, is_empty methods.
135 class Stack:
136     def __init__(self):
137         self.stack = [] # Using a List to store stack items
138     def push(self, item):
139         '''Add an item to the top of the stack.'''
140         self.stack.append(item)
141     def pop(self):
142         '''Remove and return the item at the top of the stack.'''
143         if not self.is_empty():
144             return self.stack.pop()
145         return None # Stack is empty
146     def peek(self):
147         '''Return the item at the top of the stack without
148         removing it.'''
149         if not self.is_empty():
150             return self.stack[-1]
151         return None # Stack is empty
152     def is_empty(self):
153         '''Check if the stack is empty.'''
154         return len(self.stack) == 0
155 # Example Usage
156 help_desk_stack = Stack()
157 help_desk_stack.push("Support Ticket 1")
158 help_desk_stack.push("Support Ticket 2")
159 print(help_desk_stack.peek()) # Output: "Support Ticket 2"
160 print(help_desk_stack.pop()) # Output: "Support Ticket 2"
161 print(help_desk_stack.is_empty()) # Output: False
```

PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3.py"

Support Ticket 2

Support Ticket 2

False

Explanation:

The stack manages support tickets using LIFO order, where the most recent ticket is resolved first. Push, pop, and peek operations demonstrate escalation handling effectively. This structure is suitable for urgent issue resolution workflows. AI assistance helped design stack methods and improve operational clarity.

Task 4: Hash Table

Prompt:

Generate a Python HashTable class with insert, search, and delete methods using collision handling through chaining.

Code & Output:

The screenshot shows a code editor with two tabs: 'Assignment_11.3.py' and 'Assignment_11.3'. The code editor displays Python code for a HashTable class using chaining. The terminal window shows the execution of the script and its output.

```

161 "# Task 4: Hash Table"
162 # Generate a Python HashTable class with insert, search, and delete methods using collision handling through
163 # chaining.
164 class HashTable:
165     def __init__(self, size=10):
166         self.size = size
167         self.table = [None] * size # Create a list of empty lists for chaining
168     def _hash(self, key):
169         """Generate a hash for the given key."""
170         return hash(key) % self.size
171     def insert(self, key, value):
172         """Insert a key-value pair into the hash table."""
173         index = self._hash(key)
174         # Check if the key already exists and update it
175         for i, (k, v) in enumerate(self.table[index]):
176             if k == key:
177                 self.table[index][i] = (key, value) # Update existing key
178             return
179         # If key does not exist, add new key-value pair
180         self.table[index].append((key, value))
181     def search(self, key):
182         """Search for a value by its key."""
183         index = self._hash(key)
184         for k, v in self.table[index]:
185             if k == key:
186                 return v # Return the value associated with the key
187         return None # Key not found
188     def delete(self, key):
189         """Delete a key-value pair from the hash table."""
190         index = self._hash(key)
191         for i, (k, v) in enumerate(self.table[index]):
192             if k == key:
193                 del self.table[index][i] # Remove the key-value pair
194             return True # Deletion successful
195         return False # Key not found
196 # Example Usage
197 hash_table = HashTable()
198 hash_table.insert("name", "Alice")
199 hash_table.insert("age", 30)
200 print(hash_table.search("name")) # Output: "Alice"
201 print(hash_table.delete("age")) # Output: True
202 print(hash_table.search("age")) # Output: None

```

PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py"
Alice
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>

Explanation:

The hash table stores data using a hashing function to determine storage index. Collision handling is done using chaining, allowing multiple elements per bucket. This ensures efficient average-time operations. AI helped generate structured bucket management logic.

Task 5: Real-Time Application Challenge

Prompt:

Design a Campus Resource Management feature and implement one selected feature using an appropriate data structure.

Code & Output:

The screenshot shows a code editor with two tabs: 'Assignment_113.py' and 'Assignment_113.py'. The code editor displays Python code for a 'CampusResourceManager' class. The terminal window to the right shows the output of running the script, which includes importing the module, defining the class, and demonstrating its methods with example usage.

```
File Edit Selection View Go Run Terminal Help ← → Assignment_113.py ... Assignment_113.py ... Welcome Assignment_113.py ... 203 "Task 5: Real-Time Application Challenge" 204 # Design a Campus Resource Management feature and implement one selected feature 205 # using an appropriate data structure. 206 class CampusResourceManager: 207     def __init__(self): 208         self.resources = {} # Using a dictionary to manage resources 209     def add_resource(self, resource_name, quantity): 210         '''Add a resource with its quantity.''' 211         if resource_name in self.resources: 212             self.resources[resource_name] += quantity # Update existing resource 213         else: 214             self.resources[resource_name] = quantity # Add new resource 215     def search_resource(self, resource_name): 216         '''Search for a resource by name.''' 217         return self.resources.get(resource_name, None) # Return quantity or None if not found 218     def delete_resource(self, resource_name): 219         '''Delete a resource by name.''' 220         if resource_name in self.resources: 221             del self.resources[resource_name] # Remove the resource 222             return True # Deletion successful 223         return False # Resource not found 224 # Example Usage 225 campus_manager = CampusResourceManager() 226 campus_manager.add_resource("Projector", 5) 227 campus_manager.add_resource("Whiteboard", 10) 228 print(campus_manager.search_resource("Projector")) # Output: 5 229 print(campus_manager.delete_resource("Whiteboard")) # Output: True 230 print(campus_manager.search_resource("Whiteboard")) # Output: None 231  PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3\Assignment_11.3.py" 5 True None
```

Explanation:

The cafeteria system uses a queue to maintain FIFO order of service. Customers are served in the order they arrive, ensuring fairness. This data structure matches real-world queue behavior. AI assistance helped implement and structure the queue methods efficiently.

Final Conclusion:

This lab demonstrated implementation of fundamental data structures using AI assistance. Structures such as arrays, linked lists, stacks, queues, priority queues, and hash tables were explored in practical scenarios. AI tools improved code clarity and development speed. However, logical understanding and correct structure selection remain essential responsibilities of the developer.