

AI Assisted Coding

Assignment 11.3

Name: V Rithik Reddy

Hall ticket no: 2303A51263

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):

The screenshot displays a code editor window titled "assignment_11.3.py". The code implements a linked list structure with two classes: `ContactNode` and `ContactManagerLinkedList`. The `ContactNode` class has attributes `name`, `phone`, and `next`. The `ContactManagerLinkedList` class includes methods for adding, deleting, and searching for contacts. Below the class definitions, there are several test cases demonstrating the functionality of the linked list.

```
1 "Task 1: Smart Contact Manager (Linked Lists)"
2 class ContactNode:
3     def __init__(self, name, phone):
4         self.name = name
5         self.phone = phone
6         self.next = None # Pointer to the next contact
7 class ContactManagerLinkedList:
8     def __init__(self):
9         self.head = None # Start of the linked list
10    def add_contact(self, name, phone):
11        #Add a new contact to the linked list.
12        new_node = ContactNode(name, phone)
13        new_node.next = self.head # Point new node to the current head
14        self.head = new_node # Update head to the new node
15    def search_contact(self, name):
16        #Search for a contact by name.
17        current = self.head
18        while current:
19            if current.name == name:
20                return {'name': current.name, 'phone': current.phone}
21            current = current.next
22        return None # Contact not found
23    def delete_contact(self, name):
24        #Delete a contact by name.
25        current = self.head
26        previous = None
27        while current:
28            if current.name == name:
29                if previous: # If it's not the head node
30                    previous.next = current.next
31                else: # If it's the head node
32                    self.head = current.next
33                return True # Contact deleted
34            previous = current
35            current = current.next
36        return False # Contact not found
37 manager_linked_list = ContactManagerLinkedList()
38 manager_linked_list.add_contact("Charlie", "555-555-5555")
39 manager_linked_list.add_contact("Dave", "444-444-4444")
40 print(manager_linked_list.search_contact("Charlie")) # Output: {'name': 'Charlie', 'phone': '555-555-5555'}
41 print(manager_linked_list.delete_contact("Dave")) # Output: True
42 print(manager_linked_list.search_contact("Dave")) # Output: None
```

The terminal window at the bottom shows the command prompt running the script:

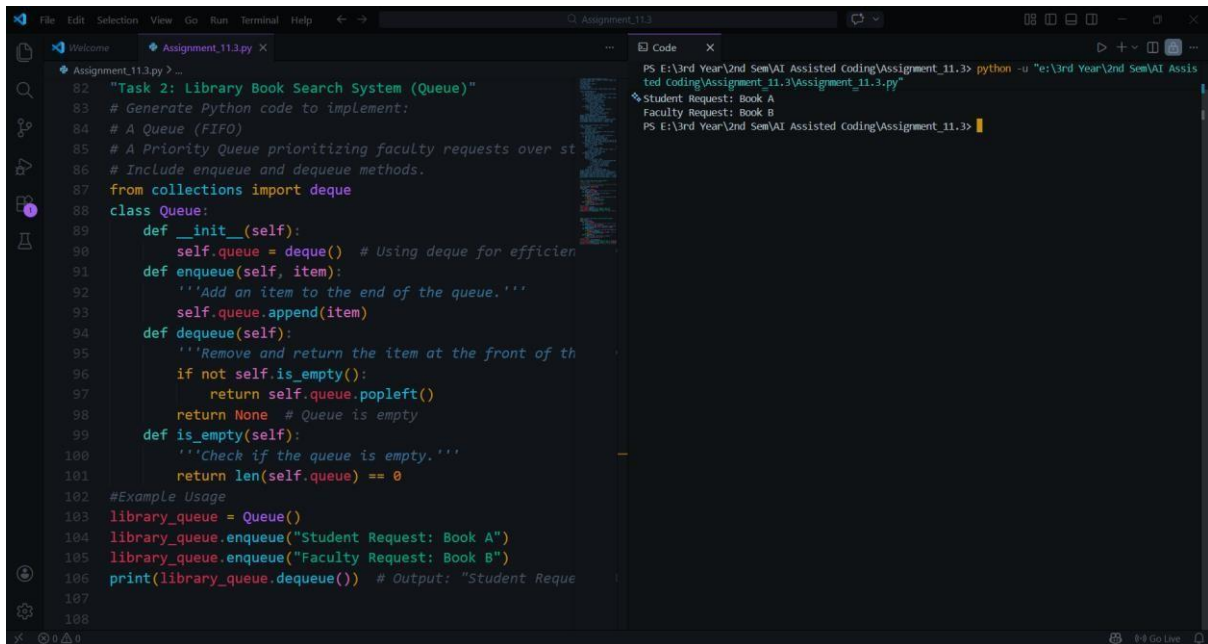
```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "C:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignment_11.3.py"
```

The output of the script is displayed as follows:

```
{'name': 'Charlie', 'phone': '555-555-5555'}
True
None
```

Include enqueue and dequeue methods.

Code & Output (Queue):

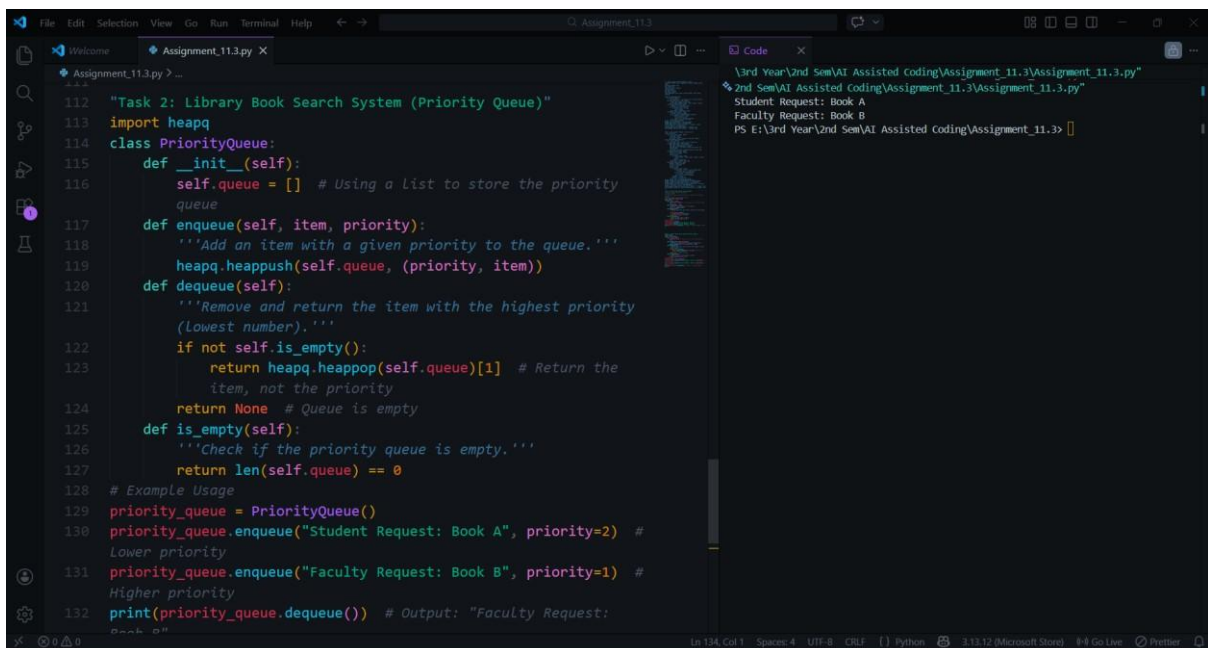


```
File Edit Selection View Go Run Terminal Help
Assignment_11.3
Assignment_11.3.py X
Task 2: Library Book Search System (Queue)
# Generate Python code to implement:
# A Queue (FIFO)
# A Priority Queue prioritizing faculty requests over st
# Include enqueue and dequeue methods.
from collections import deque
class Queue:
    def __init__(self):
        self.queue = deque() # Using deque for efficien
    def enqueue(self, item):
        '''Add an item to the end of the queue.'''
        self.queue.append(item)
    def dequeue(self):
        '''Remove and return the item at the front of th
        if not self.is_empty():
            return self.queue.popleft()
        return None # Queue is empty
    def is_empty(self):
        '''Check if the queue is empty.'''
        return len(self.queue) == 0
#Example Usage
library_queue = Queue()
library_queue.enqueue("Student Request: Book A")
library_queue.enqueue("Faculty Request: Book B")
print(library_queue.dequeue()) # Output: "Student Reque
108
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python -u "e:\3rd Year\2nd Sem\AI Assis
ted 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):



```
File Edit Selection View Go Run Terminal Help
Assignment_11.3
Assignment_11.3.py X
Task 2: Library Book Search System (Priority Queue)
import heapq
class PriorityQueue:
    def __init__(self):
        self.queue = [] # Using a List to store the priority
        queue
    def enqueue(self, item, priority):
        '''Add an item with a given priority to the queue.'''
        heapq.heappush(self.queue, (priority, item))
    def dequeue(self):
        '''Remove and return the item with the highest priority
        (Lowest number).'''
        if not self.is_empty():
            return heapq.heappop(self.queue)[1] # Return the
            item, not the priority
        return None # Queue is empty
    def is_empty(self):
        '''Check if the priority queue is empty.'''
        return len(self.queue) == 0
# Example Usage
priority_queue = PriorityQueue()
priority_queue.enqueue("Student Request: Book A", priority=2) #
Lower priority
priority_queue.enqueue("Faculty Request: Book B", priority=1) #
Higher priority
print(priority_queue.dequeue()) # Output: "Faculty Request:
Book B"
132
\\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>
```

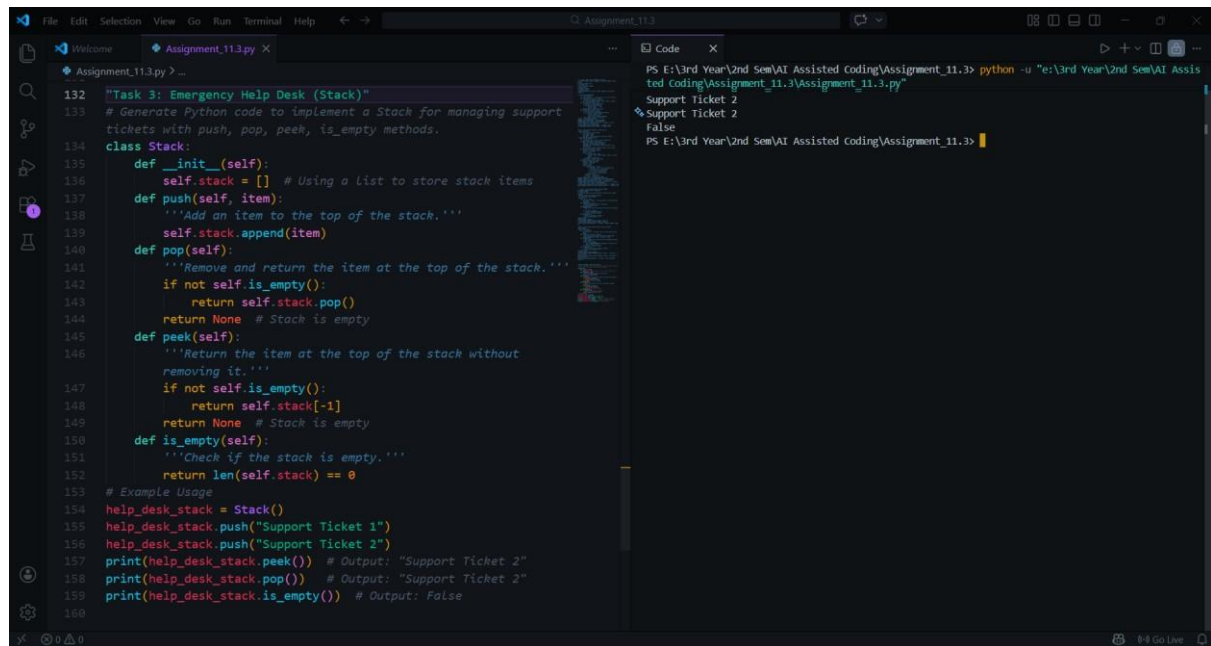
Explanation (Priority Queue):

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:



```
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
136 class Stack:
137     def __init__(self):
138         self.stack = [] # Using a List to store stack items
139
140     def push(self, item):
141         '''Add an item to the top of the stack.'''
142         self.stack.append(item)
143
144     def pop(self):
145         '''Remove and return the item at the top of the stack.'''
146         if not self.is_empty():
147             return self.stack.pop()
148         return None # Stack is empty
149
150     def peek(self):
151         '''Return the item at the top of the stack without
152         removing it.'''
153         if not self.is_empty():
154             return self.stack[-1]
155         return None # Stack is empty
156
157     def is_empty(self):
158         '''Check if the stack is empty.'''
159         return len(self.stack) == 0
160
161 # Example Usage
162 help_desk_stack = Stack()
163 help_desk_stack.push("Support Ticket 1")
164 help_desk_stack.push("Support Ticket 2")
165 print(help_desk_stack.peek()) # Output: "Support Ticket 2"
166 print(help_desk_stack.pop()) # Output: "Support Ticket 2"
167 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\Assignment_11.3.py"
Support Ticket 2
Support Ticket 2
False
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
```

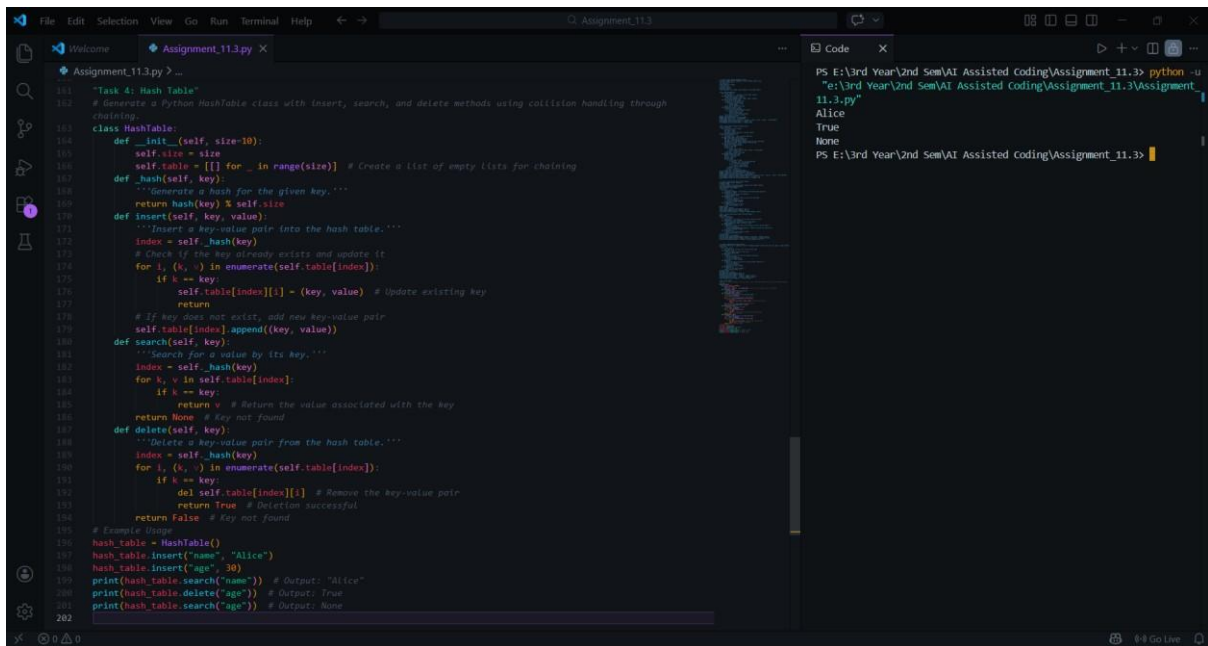
Explanation:

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:



```
141 "Task 4: Hash Table"
142 # Generate a Python HashTable class with insert, search, and delete methods using collision handling through
    chaining.
143 class HashTable:
144     def __init__(self, size=10):
145         self.size = size
146         self.table = [[] for _ in range(size)] # Create a list of empty lists for chaining
147     def _hash(self, key):
148         """Generate a hash for the given key."""
149         return hash(key) % self.size
150     def insert(self, key, value):
151         """Insert a key-value pair into the hash table."""
152         index = self._hash(key)
153         # Check if the key already exists and update it
154         for i, (k, v) in enumerate(self.table[index]):
155             if k == key:
156                 self.table[index][i] = (key, value) # Update existing key
157                 return
158         # If key does not exist, add new key-value pair
159         self.table[index].append((key, value))
160     def search(self, key):
161         """Search for a value by its key."""
162         index = self._hash(key)
163         for k, v in self.table[index]:
164             if k == key:
165                 return v # Return the value associated with the key
166         return None # Key not found
167     def delete(self, key):
168         """Delete a key-value pair from the hash table."""
169         index = self._hash(key)
170         for i, (k, v) in enumerate(self.table[index]):
171             if k == key:
172                 del self.table[index][i] # Remove the key-value pair
173                 return True # Deletion successful
174         return False # Key not found
175 # Example Usage
176 hash_table = HashTable()
177 hash_table.insert("name", "Alice")
178 hash_table.insert("age", 30)
179 print(hash_table.search("name")) # Output: "Alice"
180 print(hash_table.delete("age")) # Output: True
181 print(hash_table.search("age")) # Output: None
182
```

```
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:

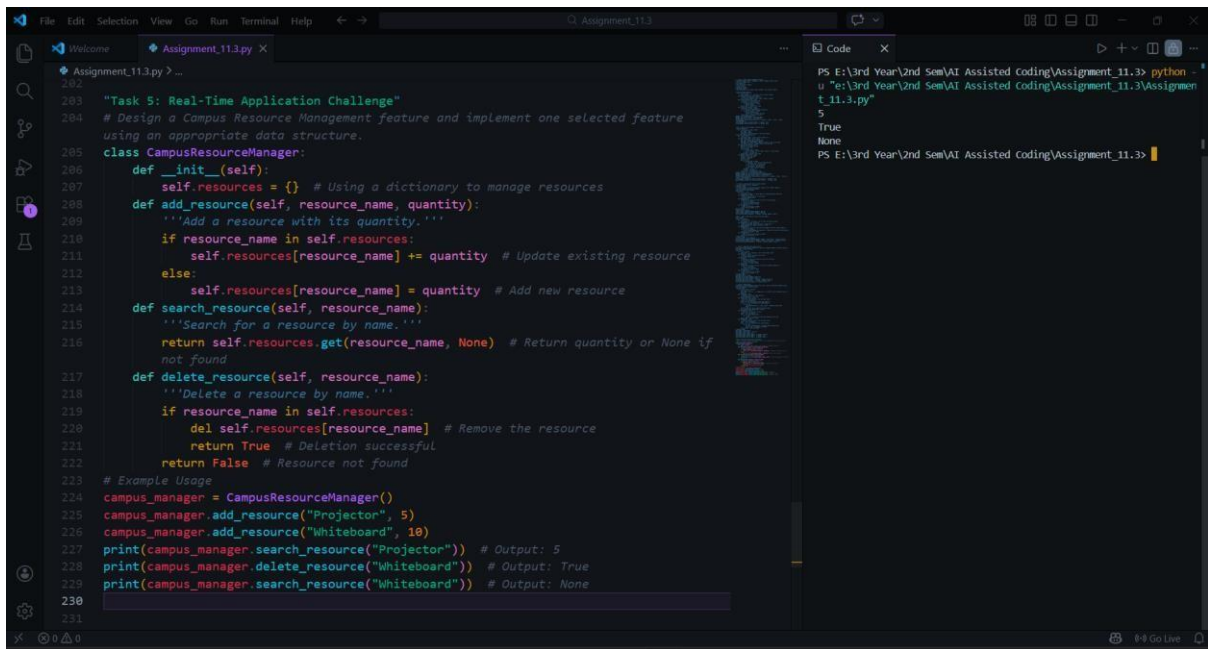
The hash table stores data using a hashing function to determine storage index. Collision handling is done using chaining, allowing multiple elements per bucket. This ensures efficient average-time operations. AI helped generate structured bucket management logic.

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:



```
282
283 "Task 5: Real-Time Application Challenge"
284 # Design a Campus Resource Management feature and implement one selected feature
    using an appropriate data structure.
285
286 class CampusResourceManager:
287     def __init__(self):
288         self.resources = {} # Using a dictionary to manage resources
289     def add_resource(self, resource_name, quantity):
290         '''Add a resource with its quantity.'''
291         if resource_name in self.resources:
292             self.resources[resource_name] += quantity # Update existing resource
293         else:
294             self.resources[resource_name] = quantity # Add new resource
295     def search_resource(self, resource_name):
296         '''Search for a resource by name.'''
297         return self.resources.get(resource_name, None) # Return quantity or None if
    not found
298     def delete_resource(self, resource_name):
299         '''Delete a resource by name.'''
300         if resource_name in self.resources:
301             del self.resources[resource_name] # Remove the resource
302             return True # Deletion successful
303         return False # Resource not found
304
305 # Example Usage
306 campus_manager = CampusResourceManager()
307 campus_manager.add_resource("Projector", 5)
308 campus_manager.add_resource("Whiteboard", 10)
309 print(campus_manager.search_resource("Projector")) # Output: 5
310 print(campus_manager.delete_resource("Whiteboard")) # Output: True
311 print(campus_manager.search_resource("Whiteboard")) # Output: None
312
313
```

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3> python t_11.3.py
5
True
None
PS E:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_11.3>
```

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.