ASSIGNMENT - 11.3

Lab 11: Data Structures with AI Implementing Fundamental Data

Structures using AI Assistance

NAME: M.NISHITHA

HTNO: 2303A51540

BTNO:19

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

Scenario

SR University's student club requires a simple Contact Manager Application to store members' names and phone numbers. The system should support efficient addition, searching, and deletion of contacts.

Tasks

1. Implement the contact manager using arrays (lists).

2. Implement the same functionality using a linked list for dynamic memory allocation.

3. Implement the following operations in both approaches:

o Add a contact

o Search for a contact

o Delete a contact

4. Use GitHub Copilot to assist in generating search and delete methods.

5. Compare array vs. linked list approaches with respect to:

o Insertion efficiency

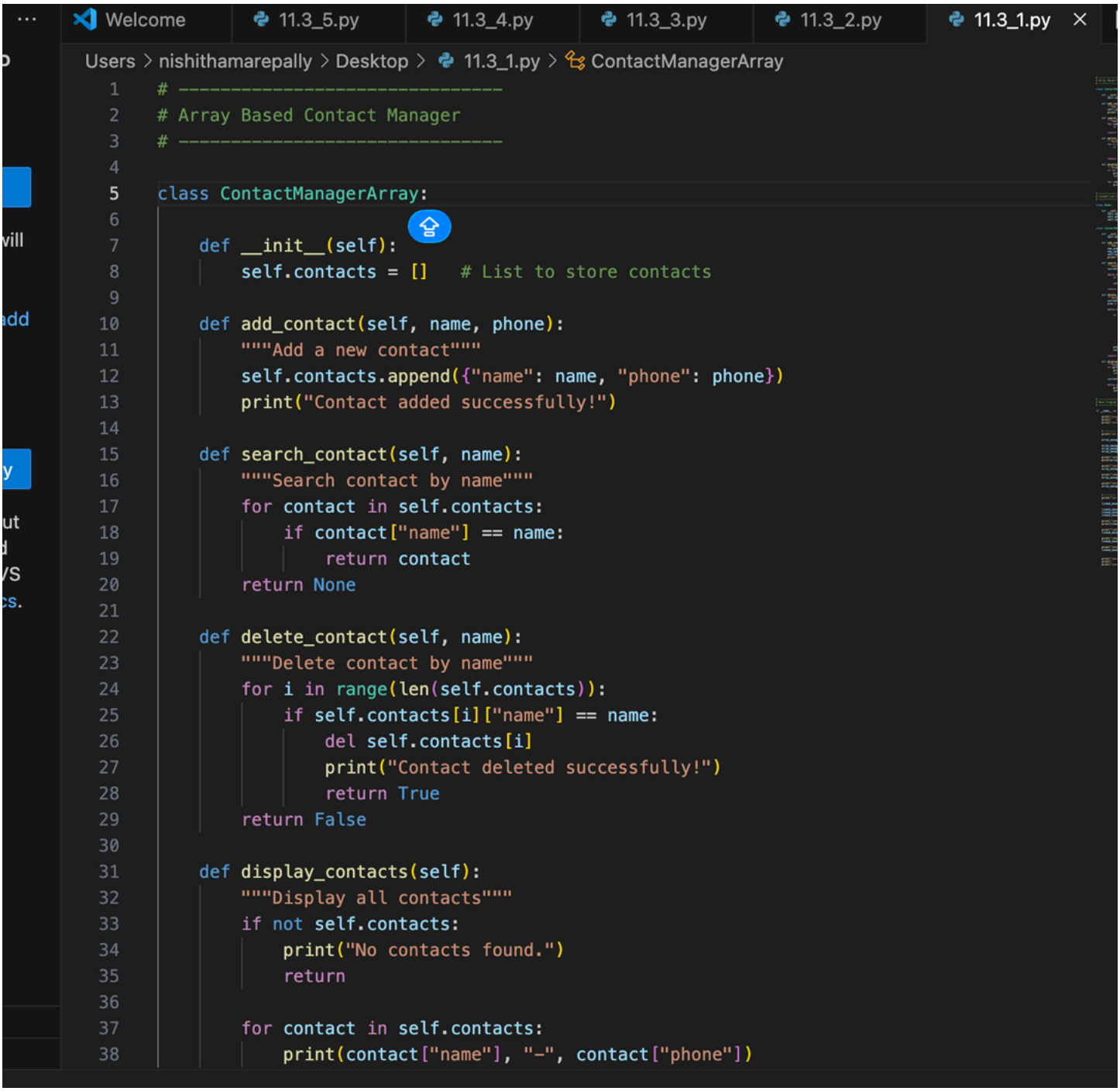
o Deletion efficiency

Expected Outcome

• Two working implementations (array-based and linked-list-based).

• A brief comparison explaining performance differences.

CODE:



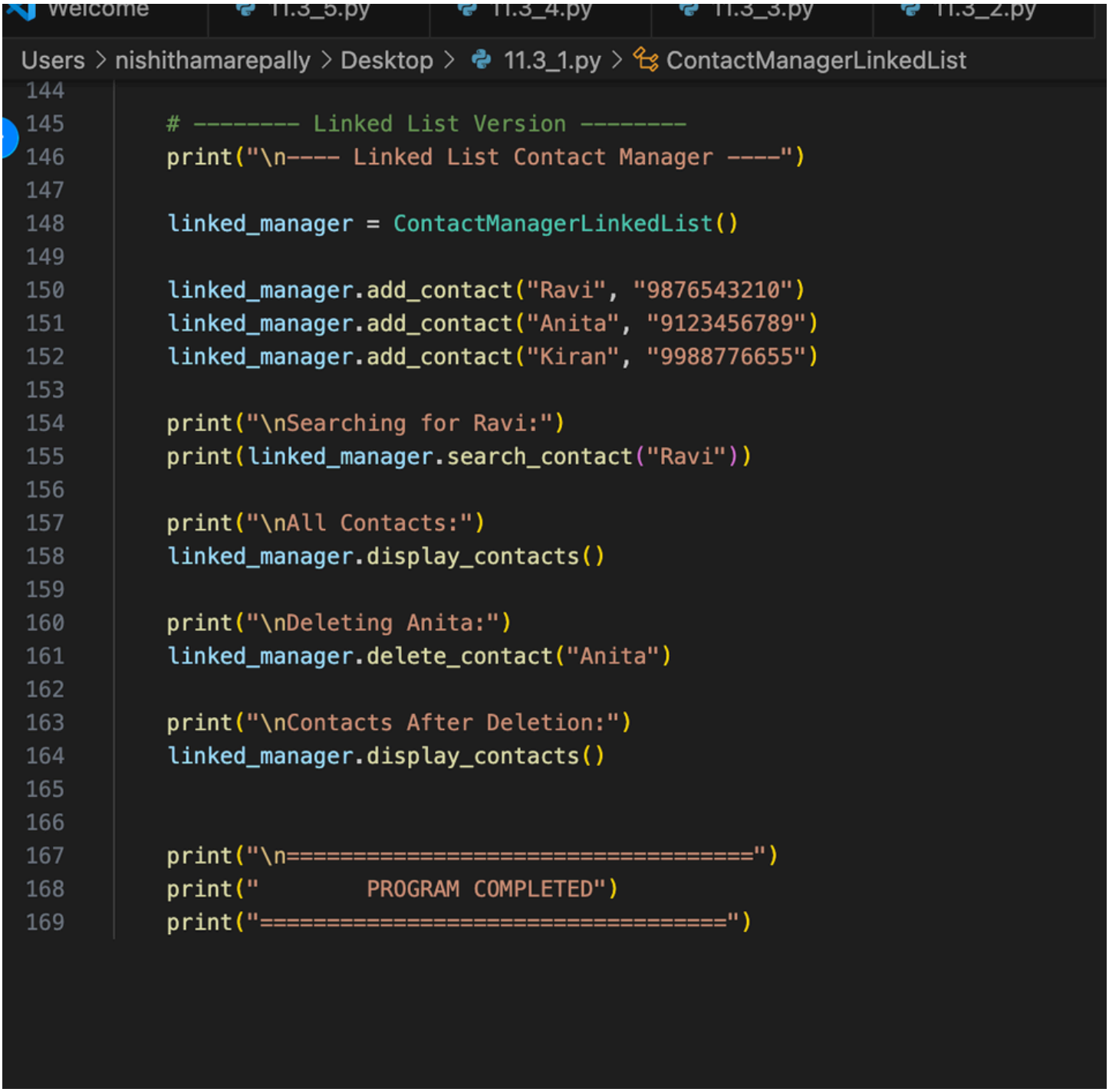
Welcome | 11.3_5.py | 11.3_4.py | 11.3_3.py | 11.3_2.py | 11.3_1.py

Users > nishithamarepally > Desktop > 11.3_1.py > ContactManagerLinkedList

```python
# ------------------------------
# Linked List Contact Manager
# ------------------------------

class Node:

    def __init__(self, name, phone):
        self.name = name
        self.phone = phone
        self.next = None


class ContactManagerLinkedList:

    def __init__(self):
        self.head = None

    def add_contact(self, name, phone):
        """Add contact at beginning"""
        new_node = Node(name, phone)
        new_node.next = self.head
        self.head = new_node
        print("Contact added successfully!")

    def search_contact(self, name):
        """Search contact by name"""
        current = self.head

        while current:
            if current.name == name:
                return {"name": current.name, "phone": current.phone}
            current = current.next

        return None

    def delete_contact(self, name):
        """Delete contact by name"""
        current = self.head
```

```
53  class ContactManagerLinkedList:
74          return None
75
76      def delete_contact(self, name):
77          """Delete contact by name"""
78          current = self.head
79          prev = None
80
81          while current:
82
83              if current.name == name:
84
85                  # If first node
86                  if prev is None:
87                      self.head = current.next
88                  else:
89                      prev.next = current.next
90
91                  print("Contact deleted successfully!")
92                  return True
93
94              prev = current
95              current = current.next
96
97          return False
98
99      def display_contacts(self):
00          """Display all contacts"""
01          if self.head is None:
02              print("No contacts found.")
03              return
04
05          current = self.head
06
07          while current:
08              print(current.name, "-", current.phone)
09              current = current.next
10
```

```python

# ---------------------------------
# Main Program (Testing)
# ---------------------------------

if __name__ == "__main__":

    print("====================================")
    print("   SMART CONTACT MANAGER SYSTEM")
    print("====================================")


    # -------- Array Version --------
    print("\n---- Array Contact Manager ----")

    array_manager = ContactManagerArray()

    array_manager.add_contact("Ravi", "9876543210")
    array_manager.add_contact("Anita", "9123456789")
    array_manager.add_contact("Kiran", "9988776655")

    print("\nSearching for Ravi:")
    print(array_manager.search_contact("Ravi"))

    print("\nAll Contacts:")
    array_manager.display_contacts()

    print("\nDeleting Anita:")
    array_manager.delete_contact("Anita")

    print("\nContacts After Deletion:")
    array_manager.display_contacts()


    # -------- Linked List Version --------
    print("\n---- Linked List Contact Manager ----")

    linked_manager = ContactManagerLinkedList()
```


Welcome
11.3_5.py
11.3_4.py
11.3_3.py
11.3_2.py
Users > nishithamarepally > Desktop > 11.3_1.py > ContactManagerLinkedList
144
145 # -------- Linked List Version --------
146 print("\n---- Linked List Contact Manager ----")
147
148 linked_manager = ContactManagerLinkedList()
149
150 linked_manager.add_contact("Ravi", "9876543210")
151 linked_manager.add_contact("Anita", "9123456789")
152 linked_manager.add_contact("Kiran", "9988776655")
153
154 print("\nSearching for Ravi:")
155 print(linked_manager.search_contact("Ravi"))
156
157 print("\nAll Contacts:")
158 linked_manager.display_contacts()
159
160 print("\nDeleting Anita:")
161 linked_manager.delete_contact("Anita")
162
163 print("\nContacts After Deletion:")
164 linked_manager.display_contacts()
165
166
167 print("\n==================================")
168 print("   PROGRAM COMPLETED")
169 print("==================================")

OUTPUT:

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

Scenario

The SRU Library manages book borrow requests. Students and faculty submit requests, but faculty requests must be prioritized over student requests.

Tasks

1. Implement a Queue (FIFO) to manage book requests.

2. Extend the system to a Priority Queue, prioritizing faculty requests.

3. Use GitHub Copilot to assist in generating:

o enqueue() method

o dequeue() method

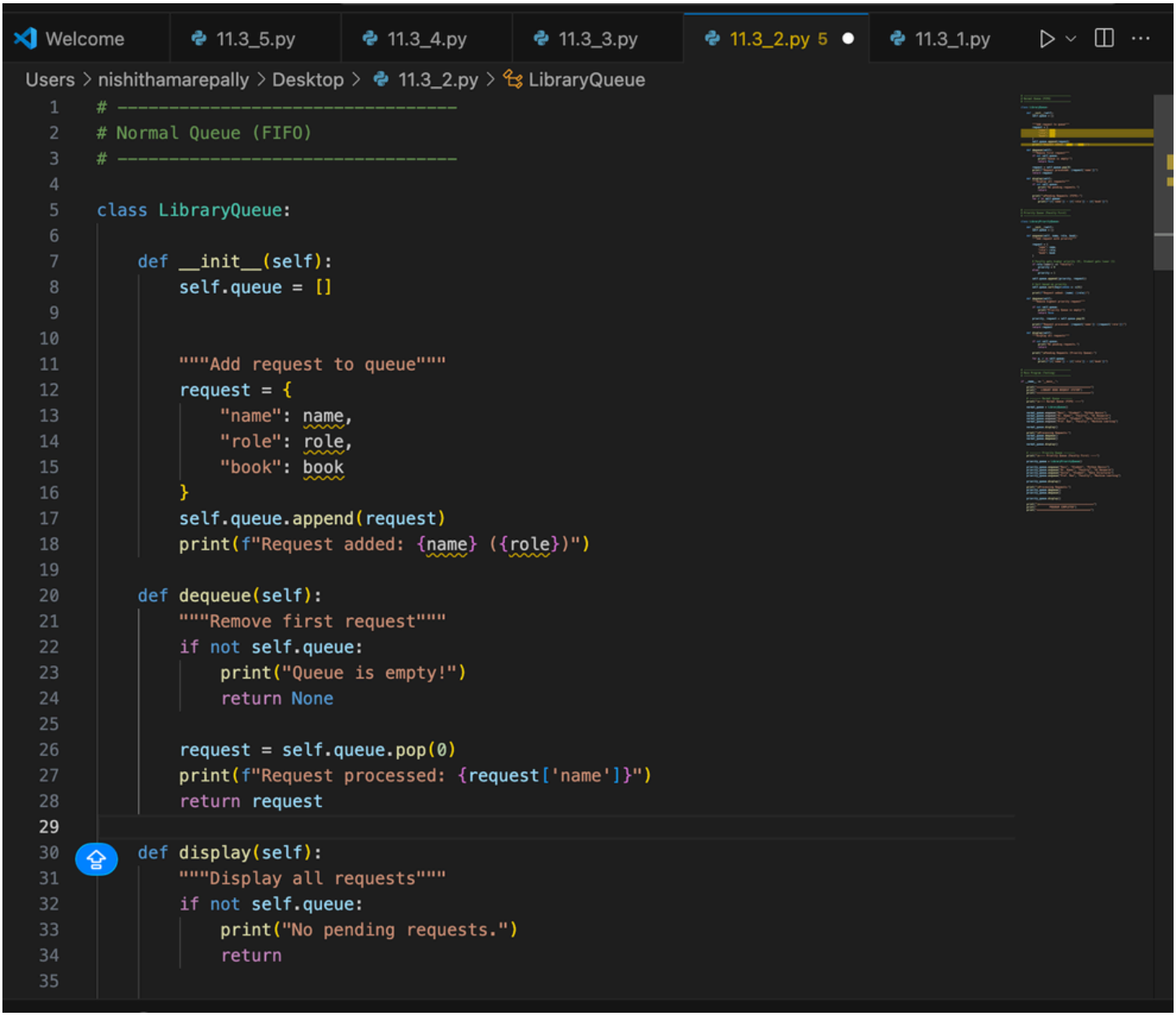
4. Test the system with a mix of student and faculty requests.

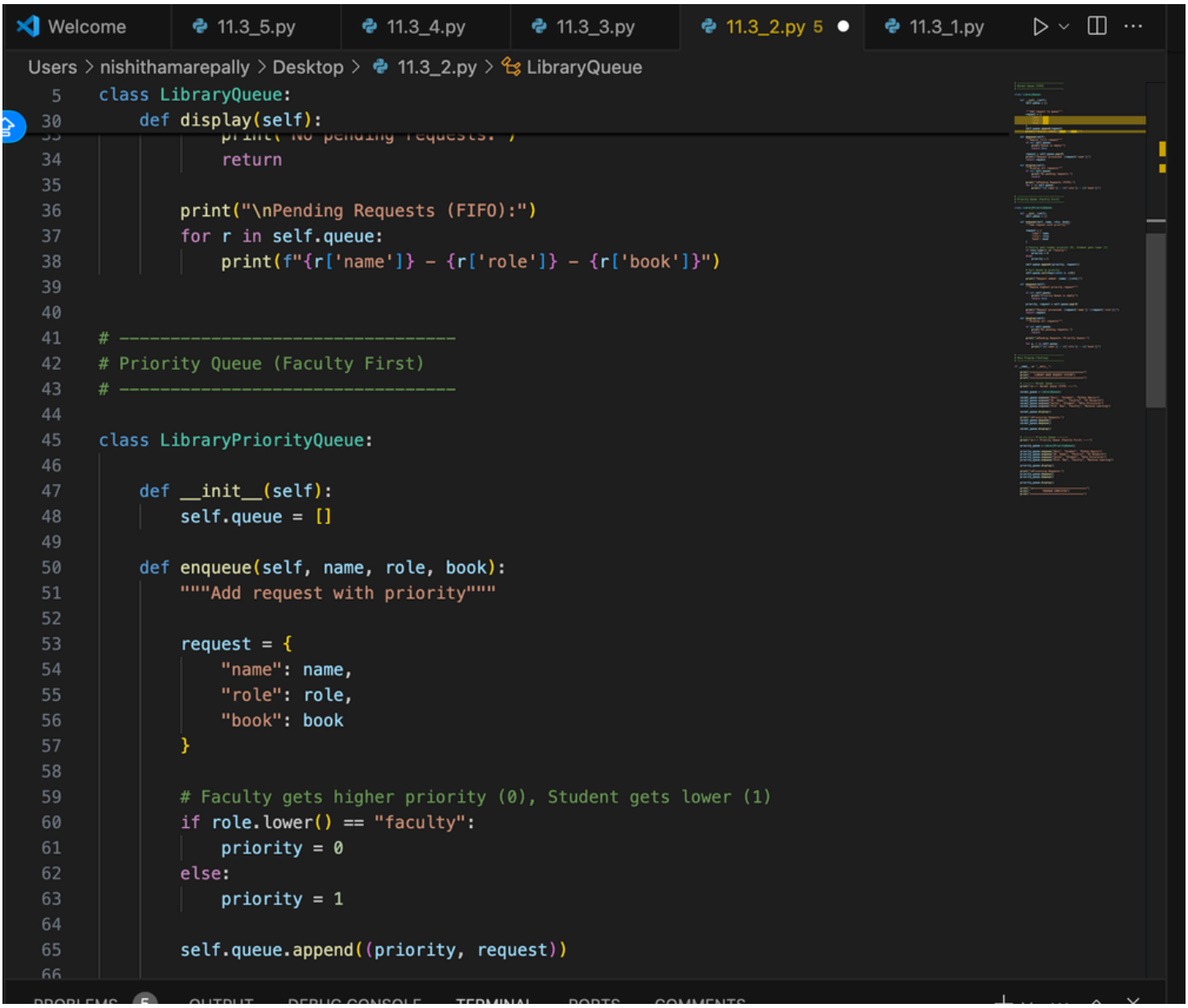
Expected Outcome

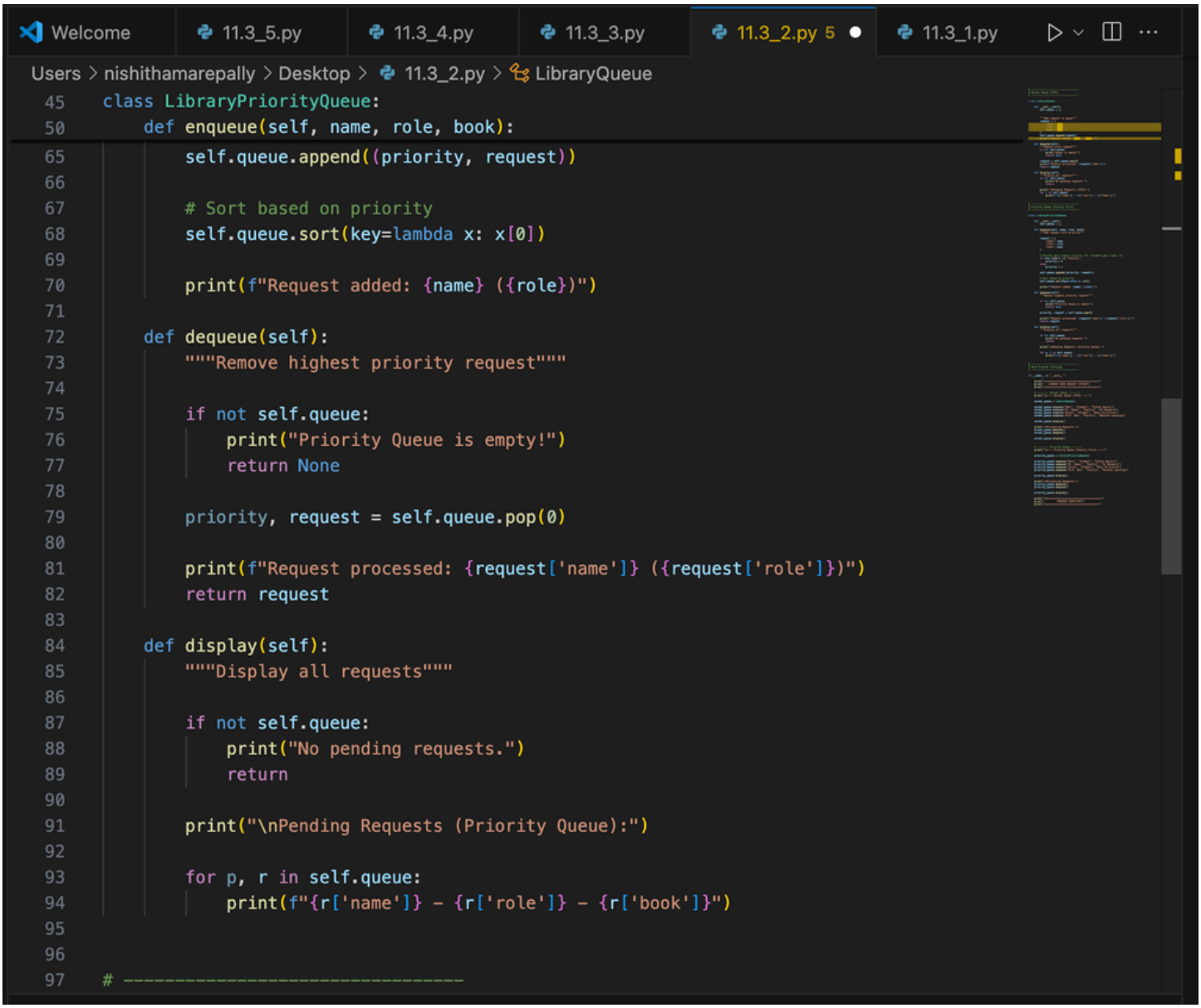
Working queue and priority queue implementations.

• Correct prioritization of faculty requests.

CODE:




Welcome
11.3_5.py
11.3_4.py
11.3_3.py
11.3_2.py 5
11.3_1.py
Users > nishithamarepally > Desktop > 11.3_2.py > LibraryQueue
class LibraryQueue:
def display(self):
return
print("\nPending Requests (FIFO):")
for r in self.queue:
print(f"{r['name']} - {r['role']} - {r['book']}")
# ------------------------------------
# Priority Queue (Faculty First)
# ------------------------------------
class LibraryPriorityQueue:
def __init__(self):
self.queue = []
def enqueue(self, name, role, book):
"""Add request with priority"""
request = {
"name": name,
"role": role,
"book": book
}
# Faculty gets higher priority (0), Student gets lower (1)
if role.lower() == "faculty":
priority = 0
else:
priority = 1
self.queue.append((priority, request))

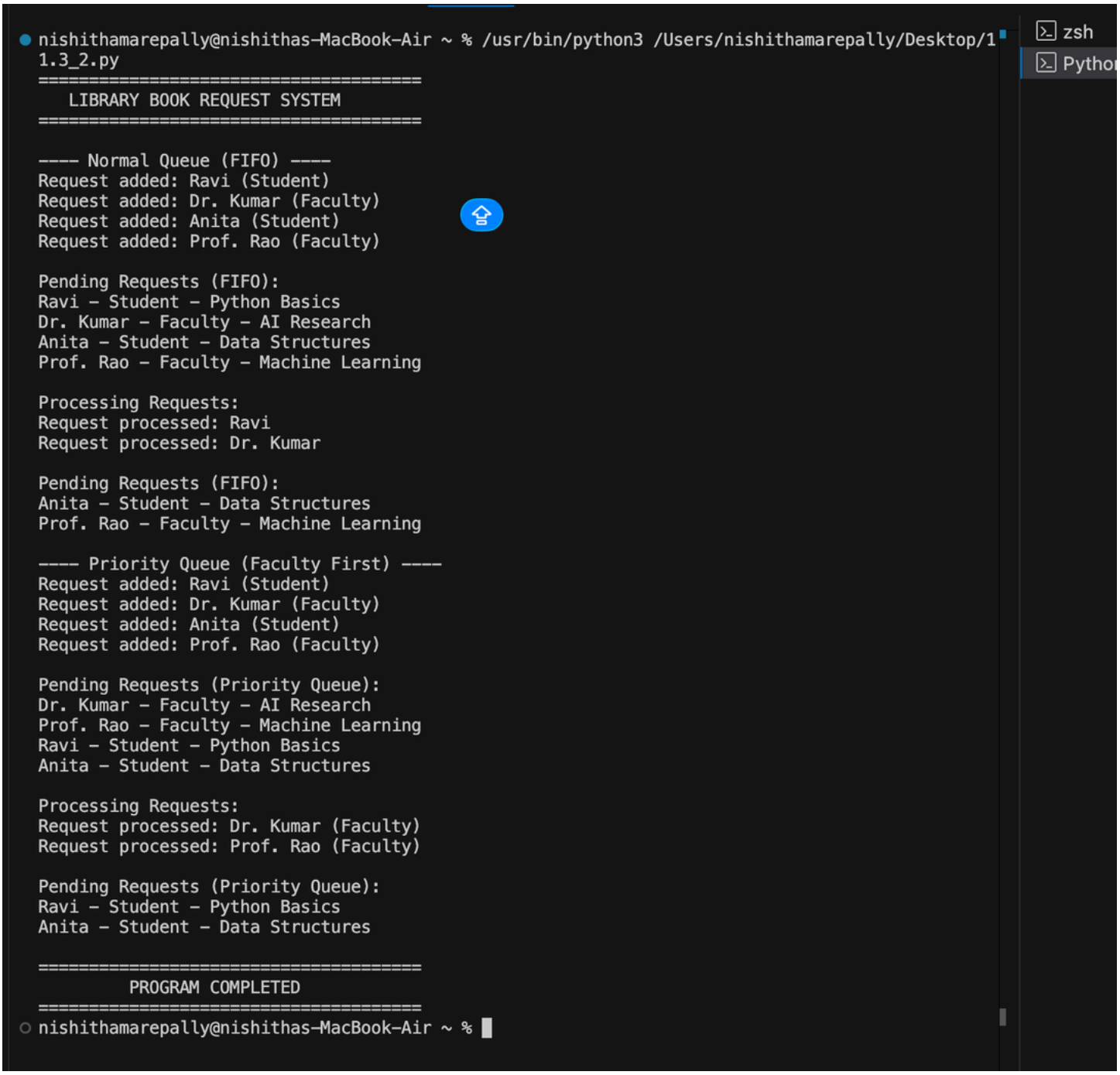

Welcome
11.3_5.py
11.3_4.py
11.3_3.py
11.3_2.py 5
11.3_1.py
Users > nishithamarepally > Desktop > 11.3_2.py > LibraryQueue
45 class LibraryPriorityQueue:
50 def enqueue(self, name, role, book):
65 self.queue.append((priority, request))
66
67 # Sort based on priority
68 self.queue.sort(key=lambda x: x[0])
69
70 print(f"Request added: {name} ({role})")
71
72 def dequeue(self):
73 """Remove highest priority request"""
74
75 if not self.queue:
76 print("Priority Queue is empty!")
77 return None
78
79 priority, request = self.queue.pop(0)
80
81 print(f"Request processed: {request['name']} ({request['role']})")
82 return request
83
84 def display(self):
85 """Display all requests"""
86
87 if not self.queue:
88 print("No pending requests.")
89 return
90
91 print("\nPending Requests (Priority Queue):")
92
93 for p, r in self.queue:
94 print(f"{r['name']} - {r['role']} - {r['book']}")
95
96
97 # ----------

```python
# ------------------------------------
# Main Program (Testing)
# ------------------------------------

if __name__ == "__main__":

    print("=======================================")
    print("   LIBRARY BOOK REQUEST SYSTEM")
    print("=======================================")

    # -------- Normal Queue --------
    print("\n---- Normal Queue (FIFO) ----")

    normal_queue = LibraryQueue()

    normal_queue.enqueue("Ravi", "Student", "Python Basics")
    normal_queue.enqueue("Dr. Kumar", "Faculty", "AI Research")
    normal_queue.enqueue("Anita", "Student", "Data Structures")
    normal_queue.enqueue("Prof. Rao", "Faculty", "Machine Learning")

    normal_queue.display()

    print("\nProcessing Requests:")
    normal_queue.dequeue()
    normal_queue.dequeue()

    normal_queue.display()


    # -------- Priority Queue --------
    print("\n---- Priority Queue (Faculty First) ----")

    priority_queue = LibraryPriorityQueue()

    priority_queue.enqueue("Ravi", "Student", "Python Basics")
```

Users > nishithamarepally > Desktop > 11.3_2.py > LibraryQueue

```python
    priority_queue.enqueue("Ravi", "Student", "Python Basics")
    priority_queue.enqueue("Dr. Kumar", "Faculty", "AI Research")
    priority_queue.enqueue("Anita", "Student", "Data Structures")
    priority_queue.enqueue("Prof. Rao", "Faculty", "Machine Learning")

    priority_queue.display()

    print("\nProcessing Requests:")
    priority_queue.dequeue()
    priority_queue.dequeue()

    priority_queue.display()

    print("\n=======================================")
    print("          PROGRAM COMPLETED")
    print("=======================================")
```

OUTPUT:



Task 3: Emergency Help Desk (Stack Implementation)

Scenario

SR University's IT Help Desk receives technical support tickets from students and staff. While tickets are received sequentially, issue escalation follows a Last-In, First-Out (LIFO) approach.

Tasks

1. Implement a Stack to manage support tickets.

2. Provide the following operations:

o push(ticket)

o pop()

o peek()

3. Simulate at least five tickets being raised and resolved.

4. Use GitHub Copilot to suggest additional stack operations such as:

o Checking whether the stack is empty

o Checking whether the stack is full (if applicable)

Expected Outcome

• Functional stack-based ticket management system.

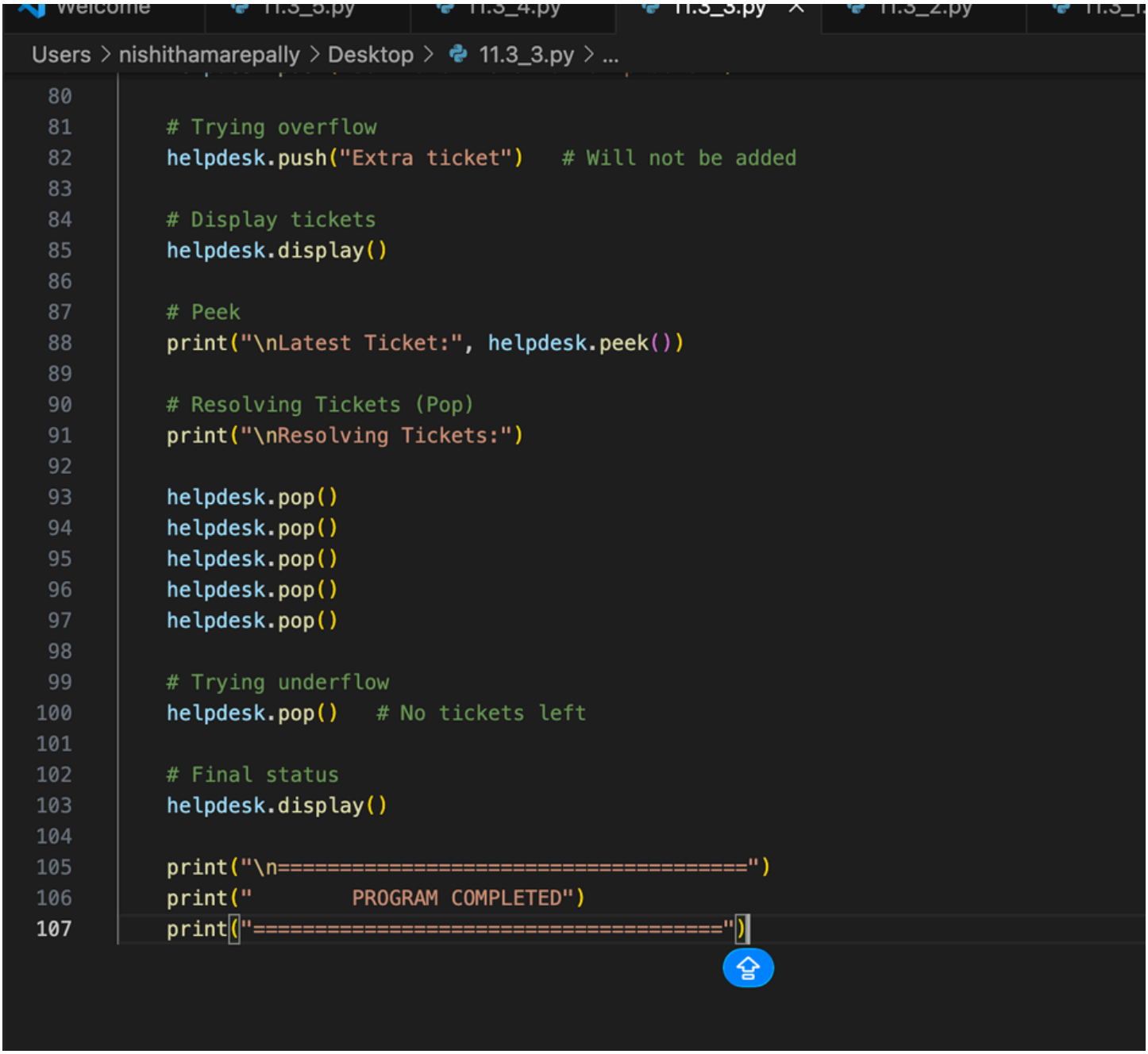
• Clear demonstration of LIFO behavior.

CODE:

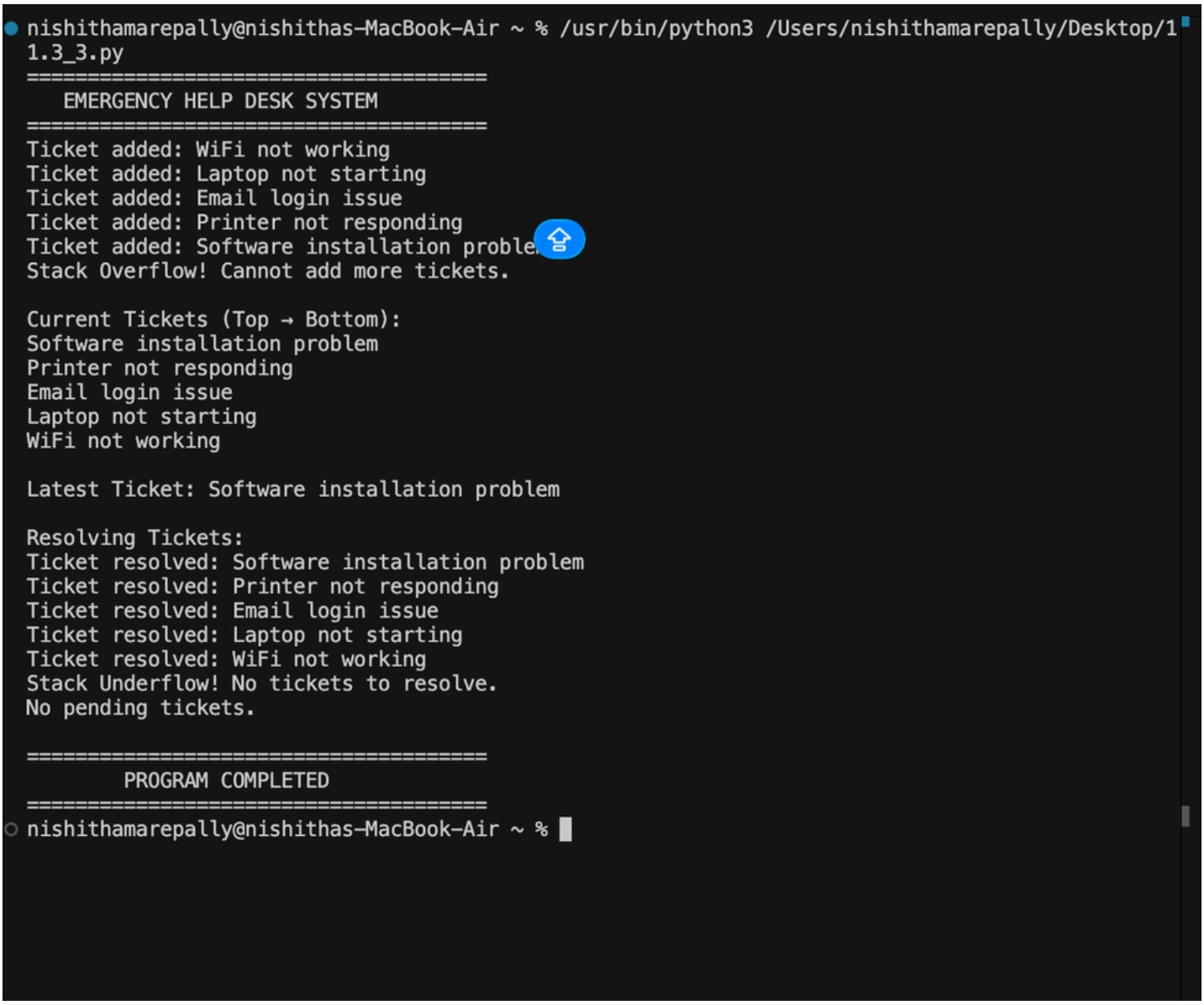

Welcome
11.3_5.py
11.3_4.py
11.3_3.py
11.3_2.py
Users > nishithamarepally > Desktop > 11.3_3.py > ...
# ---------------------------------
# Emergency Help Desk - Stack
# ---------------------------------
class HelpDeskStack:
def __init__(self, max_size=10):
self.stack = []
self.max_size = max_size
def push(self, ticket):
"""Add new ticket (Push)"""
if self.is_full():
print("Stack Overflow! Cannot add more tickets.")
return
self.stack.append(ticket)
print(f"Ticket added: {ticket}")
def pop(self):
"""Resolve last ticket (Pop)"""
if self.is_empty():
print("Stack Underflow! No tickets to resolve.")
return None
ticket = self.stack.pop()
print(f"Ticket resolved: {ticket}")
return ticket
def peek(self):
"""View latest ticket"""
if self.is_empty():

```
        if self.is_empty():
            print("No tickets available.")
            return None

        return self.stack[-1]

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

    def is_full(self):
        """Check if stack is full"""
        return len(self.stack) == self.max_size

    def display(self):
        """Display all tickets"""

        if self.is_empty():
            print("No pending tickets.")
            return

        print("\nCurrent Tickets (Top → Bottom):")

        for ticket in reversed(self.stack):
            print(ticket)


# ------------------------------------
# Main Program (Testing)
# ------------------------------------

if __name__ == "__main__":
```

```
class HelpDeskStack:
    def display(self):
        """Display all tickets"""

        if self.is_empty():
            print("No pending tickets.")
            return

        print("\nCurrent Tickets (Top → Bottom):")

        for ticket in reversed(self.stack):
            print(ticket)


# ---------------------------------
# Main Program (Testing)
# ---------------------------------

if __name__ == "__main__":

    print("=======================================")
    print("   EMERGENCY HELP DESK SYSTEM")
    print("=======================================")

    helpdesk = HelpDeskStack(max_size=5)

    # Raising Tickets (Push)
    helpdesk.push("WiFi not working")
    helpdesk.push("Laptop not starting")
    helpdesk.push("Email login issue")
    helpdesk.push("Printer not responding")
    helpdesk.push("Software installation problem")

    # Trying overflow
    helpdesk.push("Extra ticket")   # Will not be added
```


Users > nishithamarepally > Desktop > 11.3_3.py > ...
80
81 # Trying overflow
82 helpdesk.push("Extra ticket") # Will not be added
83
84 # Display tickets
85 helpdesk.display()
86
87 # Peek
88 print("\nLatest Ticket:", helpdesk.peek())
89
90 # Resolving Tickets (Pop)
91 print("\nResolving Tickets:")
92
93 helpdesk.pop()
94 helpdesk.pop()
95 helpdesk.pop()
96 helpdesk.pop()
97 helpdesk.pop()
98
99 # Trying underflow
100 helpdesk.pop() # No tickets left
101
102 # Final status
103 helpdesk.display()
104
105 print("\n==")
106 print(" PROGRAM COMPLETED")
107 print("==")

OUTPUT:



Task 4: Hash Table

Objective

To implement a Hash Table and understand collision handling.

Task Description

Use AI to generate a hash table with:

• Insert

• Search

• Delete

Starter Code

class HashTable:

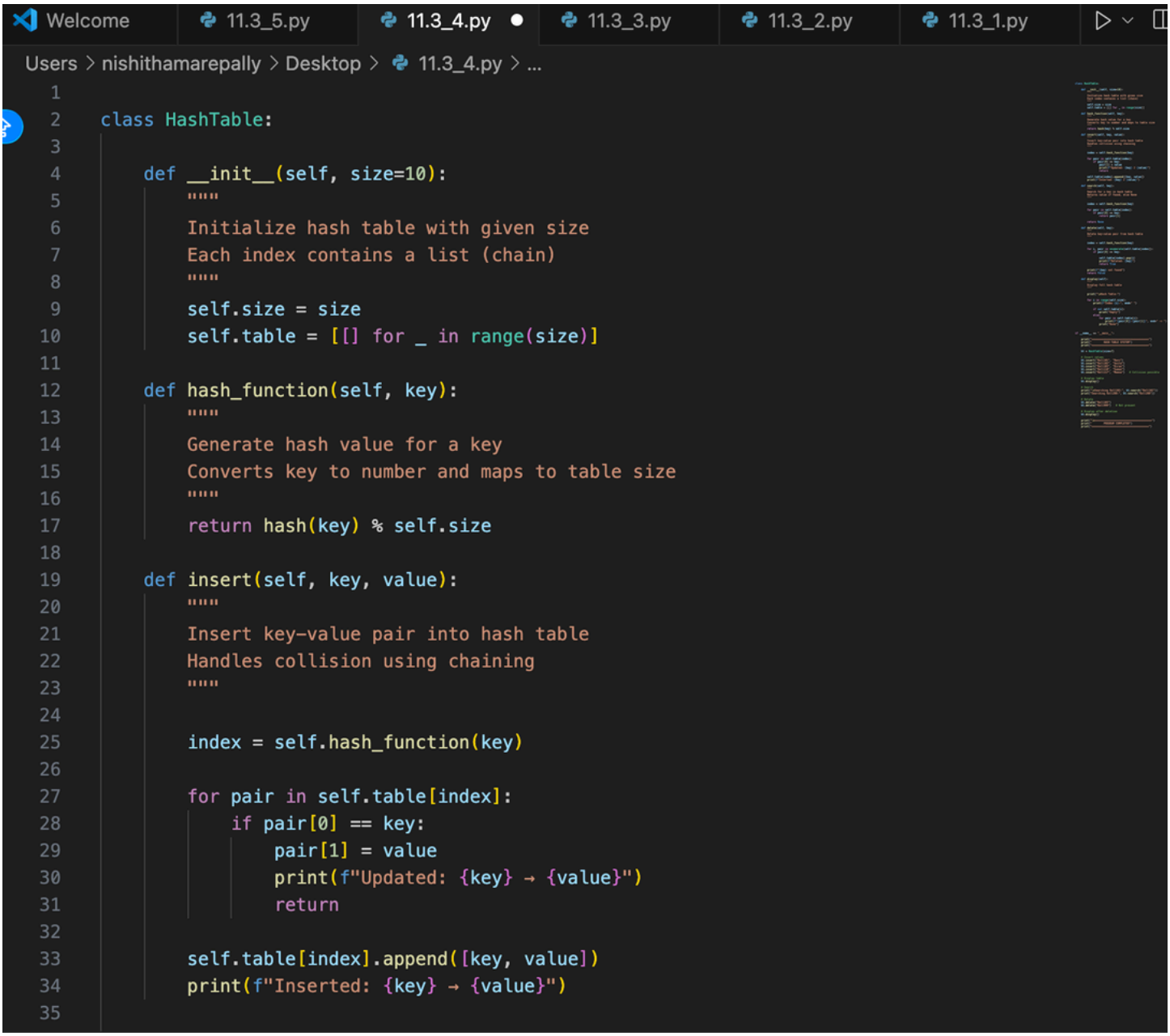
pass

Expected Outcome

• Collision handling using chaining

• Well-commented methods

CODE:



Users > nishithamarepally > Desktop > 11.3_4.py > ...

```
class HashTable:
        print(f"Inserted: {key} → {value}")

    def search(self, key):
        """
        Search for a key in hash table
        Returns value if found, else None
        """

        index = self.hash_function(key)

        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]

        return None

    def delete(self, key):
        """
        Delete key-value pair from hash table
        """

        index = self.hash_function(key)

        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:

                self.table[index].pop(i)
                print(f"Deleted: {key}")
                return True

        print(f"{key} not found")
        return False

    def display(self):
```

11.3_4.py ● | 11.3_3.py | 11.3_2.py | 11.3_1.py | 10.5_5.py | 10.5_4.py

Users > nishithamarepally > Desktop > 11.3_4.py > ...

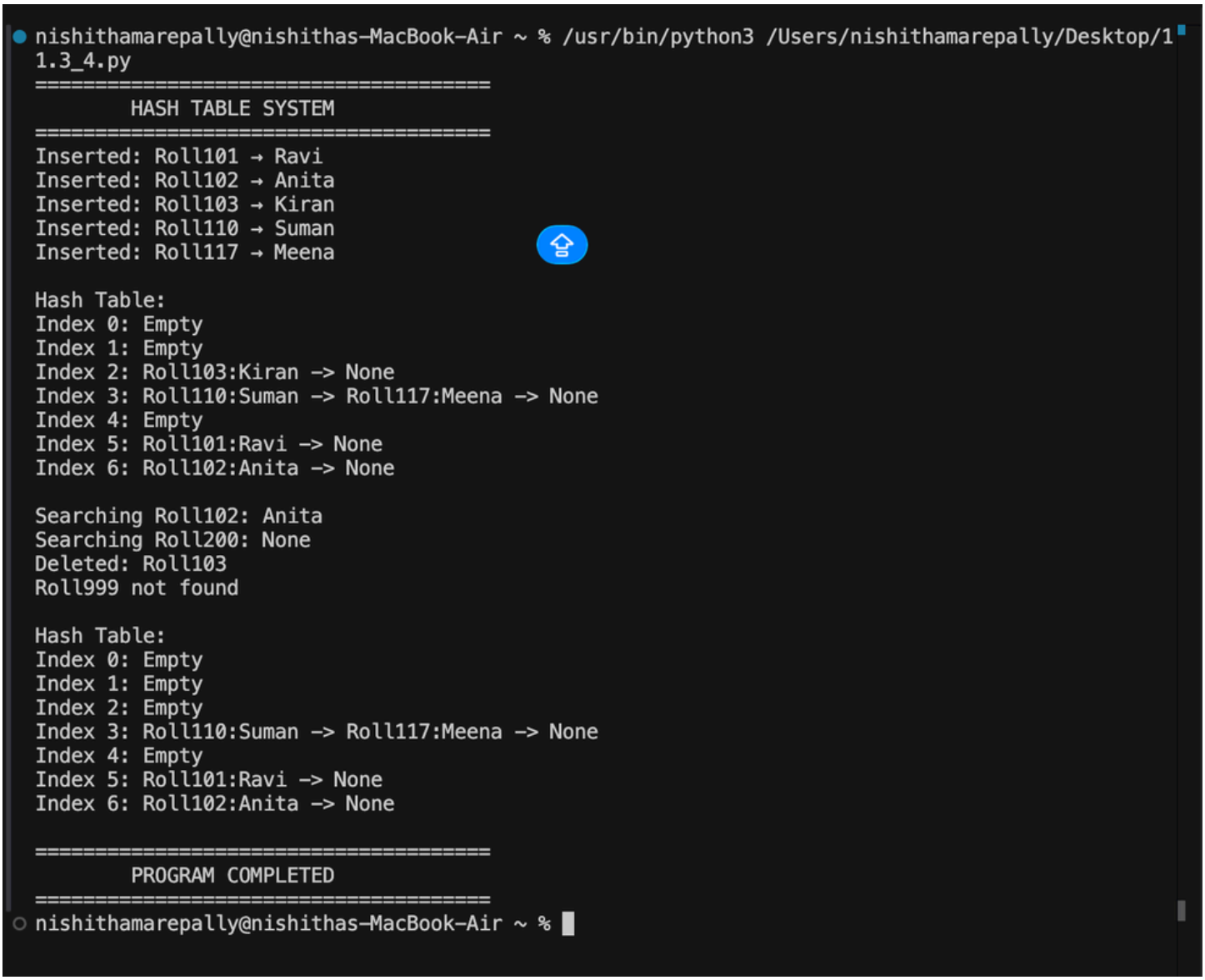
```
2   class HashTable:
67      def display(self):
69          Display full hash table
70          """
71
72          print("\nHash Table:")
73
74          for i in range(self.size):
75              print(f"Index {i}:", end=" ")
76
77              if not self.table[i]:
78                  print("Empty")
79              else:
80                  for pair in self.table[i]:
81                      print(f"{pair[0]}:{pair[1]}", end=" -> ")
82                  print("None")
83
84
85  if __name__ == "__main__":
86
87      print("=====================================")
88      print("        HASH TABLE SYSTEM")
89      print("=====================================")
90
91      ht = HashTable(size=7)
92
93      # Insert values
94      ht.insert("Roll101", "Ravi")
95      ht.insert("Roll102", "Anita")
96      ht.insert("Roll103", "Kiran")
97      ht.insert("Roll110", "Suman")
98      ht.insert("Roll117", "Meena")   # Collision possible
99
100     # Display table
101     ht.display()
```

11.3_4.py ● | 11.3_3.py | 11.3_2.py | 11.3_1.py | 10.5_5.py

Users > nishithamarepally > Desktop > 11.3_4.py > ...

```
100     # Display table
101     ht.display()
102
103     # Search
104     print("\nSearching Roll102:", ht.search("Roll102"))
105     print("Searching Roll200:", ht.search("Roll200"))
106
107     # Delete
108     ht.delete("Roll103")
109     ht.delete("Roll999")   # Not present
110
111     # Display after deletion
112     ht.display()
113
114     print("\n====================================")
115     print("        PROGRAM COMPLETED")
116     print("====================================")
```

OUTPUT:



Task 5: Real-Time Application Challenge

Scenario

Design a Campus Resource Management System with the following

features:

• Student Attendance Tracking

• Event Registration System

• Library Book Borrowing

• Bus Scheduling System

• Cafeteria Order Queue

Student Tasks

1. Choose the most appropriate data structure for each feature.

2. Justify your choice in 2–3 sentences.

3. Implement one selected feature using AI-assisted code generation.

Expected Outcome

• Mapping table: Feature → Data Structure → Justification

• One fully working Python implementation

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

CODE:

Welcome | 11.3_5.py | 11.3_4.py | 11.3_3.py | 11.3_2.py | 11.3

Users > nishithamarepally > Desktop > 11.3_5.py > ...

```python
# ---------------------------------
# Cafeteria Order Queue System
# ---------------------------------

class CafeteriaQueue:

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

    def enqueue(self, student_name, item):
        """Add new order"""
        order = {
            "name": student_name,
            "item": item
        }

        self.queue.append(order)
        print(f"Order placed: {student_name} - {item}")

    def dequeue(self):
        """Serve first order"""

        if self.is_empty():
            print("No orders to serve.")
            return None

        order = self.queue.pop(0)
        print(f"Order served: {order['name']} - {order['item']}")
        return order

    def peek(self):
        """View first order"""

        if self.is_empty():
```

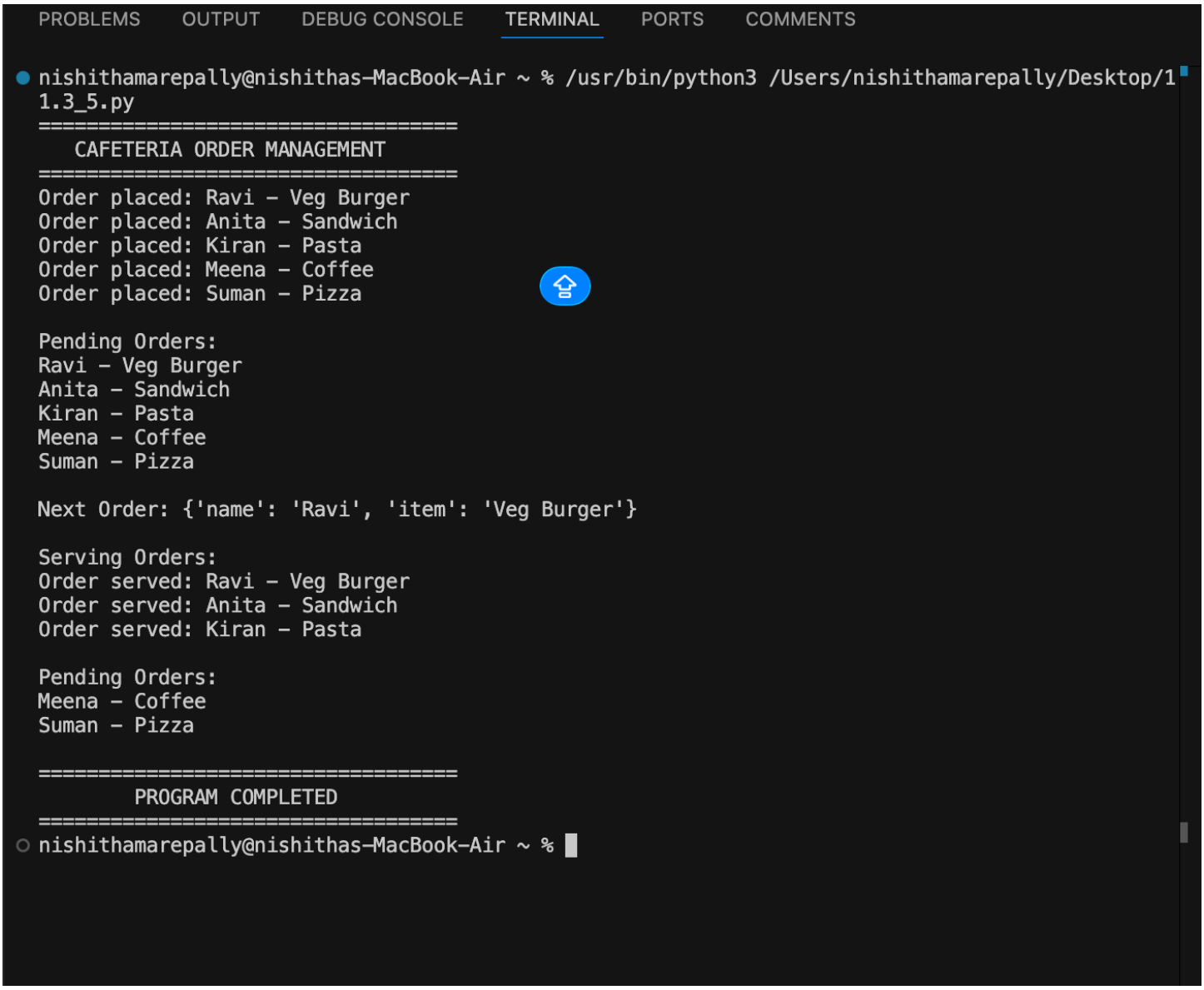
Users > nishithamarepally > Desktop > 11.3_5.py > ...

```
5   class CafeteriaQueue:
32      def peek(self):
35          if self.is_empty():
36              print("No pending orders.")
37              return None
38
39          return self.queue[0]
40
41      def is_empty(self):
42          """Check if queue is empty"""
43          return len(self.queue) == 0
44
45      def display_orders(self):
46          """Display all orders"""
47
48          if self.is_empty():
49              print("No pending orders.")
50              return
51
52          print("\nPending Orders:")
53
54          for order in self.queue:
55              print(f"{order['name']} - {order['item']}")
56
57
58  # ----------------------------------
59  # Main Program (Testing)
60  # ----------------------------------
61
62  if __name__ == "__main__":
63
64      print("==================================")
65      print("   CAFETERIA ORDER MANAGEMENT")
66      print("==================================")
67
```

Users > nishithamarepally > Desktop > 11.3_5.py > ...

```
67
68      cafeteria = CafeteriaQueue()
69
70      # Placing Orders
71      cafeteria.enqueue("Ravi", "Veg Burger")
72      cafeteria.enqueue("Anita", "Sandwich")
73      cafeteria.enqueue("Kiran", "Pasta")
74      cafeteria.enqueue("Meena", "Coffee")
75      cafeteria.enqueue("Suman", "Pizza")
76
77      # Display Orders
78      cafeteria.display_orders()
79
80      # Peek First Order
81      print("\nNext Order:", cafeteria.peek())
82
83      # Serving Orders
84      print("\nServing Orders:")
85
86      cafeteria.dequeue()
87      cafeteria.dequeue()
88      cafeteria.dequeue()
89
90      # Remaining Orders
91      cafeteria.display_orders()
92
93      print("\n====================================")
94      print("        PROGRAM COMPLETED")
95      print("====================================")
```

OUTPUT:


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
COMMENTS
nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/11.3_5.py
==================================
CAFETERIA ORDER MANAGEMENT
==================================
Order placed: Ravi - Veg Burger
Order placed: Anita - Sandwich
Order placed: Kiran - Pasta
Order placed: Meena - Coffee
Order placed: Suman - Pizza
Pending Orders:
Ravi - Veg Burger
Anita - Sandwich
Kiran - Pasta
Meena - Coffee
Suman - Pizza
Next Order: {'name': 'Ravi', 'item': 'Veg Burger'}
Serving Orders:
Order served: Ravi - Veg Burger
Order served: Anita - Sandwich
Order served: Kiran - Pasta
Pending Orders:
Meena - Coffee
Suman - Pizza
==================================
PROGRAM COMPLETED
==================================
nishithamarepally@nishithas-MacBook-Air ~ %