

# 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:

The screenshot shows a code editor window with a dark theme. The title bar has tabs for "Welcome", "11.3\_5.py", "11.3\_4.py", "11.3\_3.py", "11.3\_2.py", "11.3\_1.py", and "X". The main area displays Python code for a class named "ContactManagerArray". The code includes methods for initializing the contact list, adding new contacts, searching by name, deleting contacts, and displaying all contacts. The code uses a list to store contacts, where each contact is represented as a dictionary with "name" and "phone" keys.

```
1  # -----
2  # Array Based Contact Manager
3  # -----
4
5  class ContactManagerArray:
6      def __init__(self):
7          self.contacts = [] # List to store contacts
8
9
10     def add_contact(self, name, phone):
11         """Add a new contact"""
12         self.contacts.append({"name": name, "phone": phone})
13         print("Contact added successfully!")
14
15     def search_contact(self, name):
16         """Search contact by name"""
17         for contact in self.contacts:
18             if contact["name"] == name:
19                 return contact
20         return None
21
22     def delete_contact(self, name):
23         """Delete contact by name"""
24         for i in range(len(self.contacts)):
25             if self.contacts[i]["name"] == name:
26                 del self.contacts[i]
27                 print("Contact deleted successfully!")
28                 return True
29         return False
30
31     def display_contacts(self):
32         """Display all contacts"""
33         if not self.contacts:
34             print("No contacts found.")
35             return
36
37         for contact in self.contacts:
38             print(contact["name"], "-", contact["phone"])
```

Users &gt; nishithamarepally &gt; Desktop &gt; 11.3\_1.py &gt; ContactManagerLinkedList

```
41  # -----
42  # Linked List Contact Manager
43  # -----
44
45  class Node:
46
47      def __init__(self, name, phone):
48          self.name = name
49          self.phone = phone
50          self.next = None
51
52
53  class ContactManagerLinkedList:
54
55      def __init__(self):
56          self.head = None
57
58      def add_contact(self, name, phone):
59          """Add contact at beginning"""
60          new_node = Node(name, phone)
61          new_node.next = self.head
62          self.head = new_node
63          print("Contact added successfully!")
64
65  def search_contact(self, name):
66      """Search contact by name"""
67      current = self.head
68
69      while current:
70          if current.name == name:
71              return {"name": current.name, "phone": current.phone}
72          current = current.next
73
74      return None
75
76  def delete_contact(self, name):
77      """Delete contact by name"""
78      current = self.head
```

```
users > nishithamarepally > Desktop > 11.3_1.py > ContactManagerLinkedList
```

```
53     class ContactManagerLinkedList:
54         return None
55
56     def delete_contact(self, name):
57         """Delete contact by name"""
58         current = self.head
59         prev = None
60
61         while current:
62
63             if current.name == name:
64
65                 # If first node
66                 if prev is None:
67                     self.head = current.next
68                 else:
69                     prev.next = current.next
70
71                 print("Contact deleted successfully!")
72                 return True
73
74             prev = current
75             current = current.next
76
77         return False
78
79     def display_contacts(self):
80         """Display all contacts"""
81         if self.head is None:
82             print("No contacts found.")
83             return
84
85         current = self.head
86
87         while current:
88             print(current.name, "-", current.phone)
89             current = current.next
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
```

```
1
2 # -----
3 # Main Program (Testing)
4 # -----
5
6 if __name__ == "__main__":
7
8     print("=====")
9     print("  SMART CONTACT MANAGER SYSTEM")
0     print("=====")
1
2
3 # ----- Array Version -----
4 print("\n---- Array Contact Manager ----")
5
6 array_manager = ContactManagerArray()
7
8 array_manager.add_contact("Ravi", "9876543210")
9 array_manager.add_contact("Anita", "9123456789")
0 array_manager.add_contact("Kiran", "9988776655")
1
2 print("\nSearching for Ravi:")
3 print(array_manager.search_contact("Ravi"))
4
5 print("\nAll Contacts:")
6 array_manager.display_contacts()
7
8 print("\nDeleting Anita:")
9 array_manager.delete_contact("Anita")
0
1 print("\nContacts After Deletion:")
2 array_manager.display_contacts()
3
4
5 # ----- Linked List Version -----
6 print("\n---- Linked List Contact Manager ----")
7
8 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:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/1  
1.3_1.py  
=====  
      SMART CONTACT MANAGER SYSTEM  
=====  
---- Array Contact Manager ----  
Contact added successfully!  
Contact added successfully!  
Contact added successfully!  
  
Searching for Ravi:  
{'name': 'Ravi', 'phone': '9876543210'}  
  
All Contacts:  
Ravi - 9876543210  
Anita - 9123456789  
Kiran - 9988776655  
  
Deleting Anita:  
Contact deleted successfully!  
  
Contacts After Deletion:  
Ravi - 9876543210  
Kiran - 9988776655  
  
---- Linked List Contact Manager ----  
Contact added successfully!  
Contact added successfully!  
Contact added successfully!  
  
Searching for Ravi:  
{'name': 'Ravi', 'phone': '9876543210'}  
  
All Contacts:  
Kiran - 9988776655  
Anita - 9123456789  
Ravi - 9876543210  
  
Deleting Anita:  
Contact deleted successfully!  
  
Contacts After Deletion:  
Kiran - 9988776655  
Ravi - 9876543210  
  
=====  
      PROGRAM COMPLETED  
=====
```

○ nishithamarepally@nishithas-MacBook-Air ~ % []

## 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

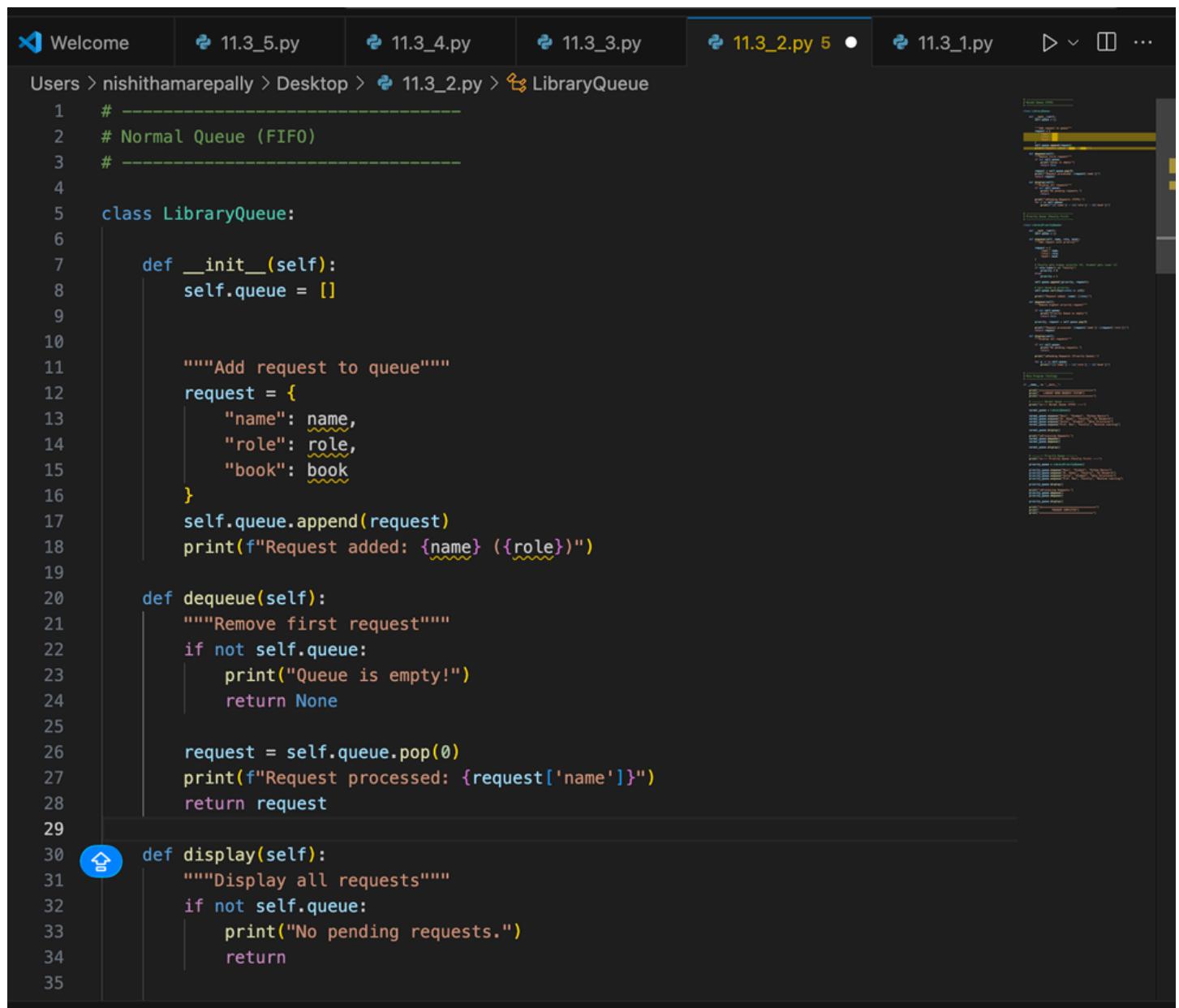
4. Test the system with a mix of student and faculty requests.

Expected Outcome

Working queue and priority queue implementations.

- Correct prioritization of faculty requests.

**CODE:**



The screenshot shows a code editor interface with multiple tabs open. The active tab is '11.3\_2.py' at line 5. The code implements a 'LibraryQueue' class with methods for enqueueing, dequeuing, and displaying requests. The code uses GitHub Copilot's AI assistance feature, indicated by a blue circular icon with a white question mark in the bottom-left corner of the code area.

```
1  # -----
2  # Normal Queue (FIFO)
3  # -----
4
5  class LibraryQueue:
6
7      def __init__(self):
8          self.queue = []
9
10
11     """Add request to queue"""
12     request = {
13         "name": name,
14         "role": role,
15         "book": book
16     }
17     self.queue.append(request)
18     print(f"Request added: {name} ({role})")
19
20     def dequeue(self):
21         """Remove first request"""
22         if not self.queue:
23             print("Queue is empty!")
24             return None
25
26         request = self.queue.pop(0)
27         print(f"Request processed: {request['name']}")
```

⚡ 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

```
5   class LibraryQueue:
30      def display(self):
31          print("No pending requests. ")
32          return
33
34
35
36      print("\nPending Requests (FIFO):")
37      for r in self.queue:
38          print(f"{r['name']} - {r['role']} - {r['book']}")
```

40

41 # -----
42 # Priority Queue (Faculty First)
43 # -----

44

```
45 class LibraryPriorityQueue:
46
47     def __init__(self):
48         self.queue = []
49
50     def enqueue(self, name, role, book):
51         """Add request with priority"""
52
53         request = {
54             "name": name,
55             "role": role,
56             "book": book
57         }
58
59         # Faculty gets higher priority (0), Student gets lower (1)
60         if role.lower() == "faculty":
61             priority = 0
62         else:
63             priority = 1
64
65         self.queue.append((priority, request))
66
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

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

```
45 class LibraryPriorityQueue:
46     def enqueue(self, name, role, book):
47         self.queue.append((priority, request))
48
49         # Sort based on priority
50         self.queue.sort(key=lambda x: x[0])
51
52         print(f"Request added: {name} ({role})")
53
54     def dequeue(self):
55         """Remove highest priority request"""
56
57         if not self.queue:
58             print("Priority Queue is empty!")
59             return None
60
61         priority, request = self.queue.pop(0)
62
63         print(f"Request processed: {request['name']} ({request['role']})")
64         return request
65
66     def display(self):
67         """Display all requests"""
68
69         if not self.queue:
70             print("No pending requests.")
71             return
72
73         print("\nPending Requests (Priority Queue):")
74
75         for p, r in self.queue:
76             print(f"{r['name']} - {r['role']} - {r['book']}")
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97 # -----
```

```
Users > nishithamarepally > Desktop > 11.3_2.py > LibraryQueue
97     # -----
98     # Main Program (Testing)
99     # -----
100
101    if __name__ == "__main__":
102
103        print("====")
104        print("  LIBRARY BOOK REQUEST SYSTEM")
105        print("====")
106
107        # ----- Normal Queue -----
108        print("\n---- Normal Queue (FIFO) ----")
109
110        normal_queue = LibraryQueue()
111
112        normal_queue.enqueue("Ravi", "Student", "Python Basics")
113        normal_queue.enqueue("Dr. Kumar", "Faculty", "AI Research")
114        normal_queue.enqueue("Anita", "Student", "Data Structures")
115        normal_queue.enqueue("Prof. Rao", "Faculty", "Machine Learning")
116
117        normal_queue.display()
118
119        print("\nProcessing Requests:")
120        normal_queue.dequeue()
121        normal_queue.dequeue()
122
123        normal_queue.display()
124
125
126        # ----- Priority Queue -----
127        print("\n---- Priority Queue (Faculty First) ----")
128
129        priority_queue = LibraryPriorityQueue()
130
131        priority_queue.enqueue("Ravi", "Student", "Python Basics")
```

```
130
131        priority_queue.enqueue("Dr. Kumar", "Faculty", "AI Research")
132        priority_queue.enqueue("Anita", "Student", "Data Structures")
133        priority_queue.enqueue("Prof. Rao", "Faculty", "Machine Learning")
134
135
136        priority_queue.display()
137
138        print("\nProcessing Requests:")
139        priority_queue.dequeue()
140        priority_queue.dequeue()
141
142        priority_queue.display()
143
144        print("====")
145        print("      PROGRAM COMPLETED")
146        print("====")
```

**OUTPUT:**

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/1.3_2.py
=====
LIBRARY BOOK REQUEST SYSTEM
=====

---- Normal Queue (FIFO) ----
Request added: Ravi (Student)
Request added: Dr. Kumar (Faculty)
Request added: Anita (Student)
Request added: Prof. Rao (Faculty)

Pending Requests (FIFO):
Ravi - Student - Python Basics
Dr. Kumar - Faculty - AI Research
Anita - Student - Data Structures
Prof. Rao - Faculty - Machine Learning

Processing Requests:
Request processed: Ravi
Request processed: Dr. Kumar

Pending Requests (FIFO):
Anita - Student - Data Structures
Prof. Rao - Faculty - Machine Learning

---- Priority Queue (Faculty First) ----
Request added: Ravi (Student)
Request added: Dr. Kumar (Faculty)
Request added: Anita (Student)
Request added: Prof. Rao (Faculty)

Pending Requests (Priority Queue):
Dr. Kumar - Faculty - AI Research
Prof. Rao - Faculty - Machine Learning
Ravi - Student - Python Basics
Anita - Student - Data Structures

Processing Requests:
Request processed: Dr. Kumar (Faculty)
Request processed: Prof. Rao (Faculty)

Pending Requests (Priority Queue):
Ravi - Student - Python Basics
Anita - Student - Data Structures
=====

PROGRAM COMPLETED
=====
```

### 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:**

Users &gt; nishithamarepally &gt; Desktop &gt; 11.3\_3.py &gt; ...

```
1  # -----
2  # Emergency Help Desk - Stack
3  # -----
4
5  class HelpDeskStack:
6
7      def __init__(self, max_size=10):
8          self.stack = []
9          self.max_size = max_size
10
11     def push(self, ticket):
12         """Add new ticket (Push)"""
13
14         if self.is_full():
15             print("Stack Overflow! Cannot add more tickets.")
16             return
17
18         self.stack.append(ticket)
19         print(f"Ticket added: {ticket}")
20
21     def pop(self):
22         """Resolve last ticket (Pop)"""
23
24         if self.is_empty():
25             print("Stack Underflow! No tickets to resolve.")
26             return None
27
28         ticket = self.stack.pop()
29         print(f"Ticket resolved: {ticket}")
30         return ticket
31
32     def peek(self):
33         """View latest ticket"""
34
35         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__":
    pass
```

```
5   class HelpDeskStack:
49     def display(self):
50       |  Display all tickets
51
52       if self.is_empty():
53         print("No pending tickets.")
54         return
55
56       print("\nCurrent Tickets (Top → Bottom):")
57
58       for ticket in reversed(self.stack):
59         print(ticket)
60
61
62 # -----
63 # Main Program (Testing)
64 # -----
65
66 if __name__ == "__main__":
67
68   print("===== ")
69   print("  EMERGENCY HELP DESK SYSTEM")
70   print("===== ")
71
72 helpdesk = HelpDeskStack(max_size=5)
73
74 # Raising Tickets (Push)
75 helpdesk.push("WiFi not working")
76 helpdesk.push("Laptop not starting")
77 helpdesk.push("Email login issue")
78 helpdesk.push("Printer not responding")
79 helpdesk.push("Software installation problem")
80
81 # Trying overflow
82 helpdesk.push("Extra ticket")  # Will not be added
83
```

```
~ 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_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:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/1.3_3.py
=====
    EMERGENCY HELP DESK SYSTEM
=====
Ticket added: WiFi not working
Ticket added: Laptop not starting
Ticket added: Email login issue
Ticket added: Printer not responding
Ticket added: Software installation problem
Stack Overflow! Cannot add more tickets.

Current Tickets (Top → Bottom):
Software installation problem
Printer not responding
Email login issue
Laptop not starting
WiFi not working

Latest Ticket: Software installation problem

Resolving Tickets:
Ticket resolved: Software installation problem
Ticket resolved: Printer not responding
Ticket resolved: Email login issue
Ticket resolved: Laptop not starting
Ticket resolved: WiFi not working
Stack Underflow! No tickets to resolve.
No pending tickets.

=====
        PROGRAM COMPLETED
=====
```

○ nishithamarepally@nishithas-MacBook-Air ~ %

## 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:

```
1  Welcome          2  11.3_5.py      3  11.3_4.py      4  ●  5  11.3_3.py      6  11.3_2.py      7  11.3_1.py      8  ▶ v  9  [ ]
```

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

```
1
2  class HashTable:
3
4      def __init__(self, size=10):
5          """
6              Initialize hash table with given size
7              Each index contains a list (chain)
8          """
9          self.size = size
10         self.table = [[] for _ in range(size)]
11
12     def hash_function(self, key):
13         """
14             Generate hash value for a key
15             Converts key to number and maps to table size
16         """
17         return hash(key) % self.size
18
19     def insert(self, key, value):
20         """
21             Insert key-value pair into hash table
22             Handles collision using chaining
23         """
24
25         index = self.hash_function(key)
26
27         for pair in self.table[index]:
28             if pair[0] == key:
29                 pair[1] = value
30                 print(f"Updated: {key} → {value}")
31                 return
32
33         self.table[index].append([key, value])
34         print(f"Inserted: {key} → {value}")
```

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

```
2     class HashTable:
34         print(f"Inserted: {key} → {value}")
35
36     def search(self, key):
37         """
38             Search for a key in hash table
39             Returns value if found, else None
40         """
41
42         index = self.hash_function(key)
43
44         for pair in self.table[index]:
45             if pair[0] == key:
46                 return pair[1]
47
48         return None
49
50     def delete(self, key):
51         """
52             Delete key-value pair from hash table
53         """
54
55         index = self.hash_function(key)
56
57         for i, pair in enumerate(self.table[index]):
58             if pair[0] == key:
59
60                 self.table[index].pop(i)
61                 print(f"Deleted: {key}")
62                 return True
63
64             print(f"{key} not found")
65         return False
66
67     def display(self):
```

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()
```

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("====")
115     print("      PROGRAM COMPLETED")
116     print("====")
```

OUTPUT:

```
● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/1  
1.3_4.py  
=====  
HASH TABLE SYSTEM  
=====  
Inserted: Roll101 -> Ravi  
Inserted: Roll102 -> Anita  
Inserted: Roll103 -> Kiran  
Inserted: Roll110 -> Suman  
Inserted: Roll117 -> Meena  
=====  
Hash Table:  
Index 0: Empty  
Index 1: Empty  
Index 2: Roll103:Kiran -> None  
Index 3: Roll110:Suman -> Roll117:Meena -> None  
Index 4: Empty  
Index 5: Roll101:Ravi -> None  
Index 6: Roll102:Anita -> None  
  
Searching Roll102: Anita  
Searching Roll200: None  
Deleted: Roll103  
Roll999 not found  
  
Hash Table:  
Index 0: Empty  
Index 1: Empty  
Index 2: Empty  
Index 3: Roll110:Suman -> Roll117:Meena -> None  
Index 4: Empty  
Index 5: Roll101:Ravi -> None  
Index 6: Roll102:Anita -> None  
  
=====  
PROGRAM COMPLETED  
=====
```

○ nishithamarepally@nishithas-MacBook-Air ~ %

## 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 > ...

```
1  # -----
2  # Cafeteria Order Queue System
3  # -----
4
5  class CafeteriaQueue:
6
7      def __init__(self):
8          """Initialize empty queue"""
9          self.queue = []
10
11     def enqueue(self, student_name, item):
12         """Add new order"""
13         order = {
14             "name": student_name,
15             "item": item
16         }
17
18         self.queue.append(order)
19         print(f"Order placed: {student_name} - {item}")
20
21     def dequeue(self):
22         """Serve first order"""
23
24         if self.is_empty():
25             print("No orders to serve.")
26             return None
27
28         order = self.queue.pop(0)
29         print(f"Order served: {order['name']} - {order['item']}")
30         return order
31
32     def peek(self):
33         """View first order"""
34
35         if self.is_empty():
36             print("No orders to view.")
```

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("====")
```

```
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:

● nishithamarepally@nishithas-MacBook-Air ~ % /usr/bin/python3 /Users/nishithamarepally/Desktop/1.3\_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 ~ %