

ASSIGNMENT-11

**NAME: B.MANISHWAR
HALL TICKET:2303A51276
BATCH:05**

Task 1

PROMPT:

Generate a Python Smart Contact Manager application for a student club. Implement it using both an array (Python list) and a linked list. Each contact should store a name and phone number. Include methods to add, search, and delete contacts in both implementations.

CODE:

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Array-based contact manager > Untitled50.ipynb - Colab
- Header:** colab.research.google.com/drive/1aEYboE_Vq8RhMZimmsHytm3vNUT8gedj
- Toolbar:** File Edit View Insert Runtime Tools Help
- Code Cell:** Contains Python code for a ContactManagerArray class. The code includes methods for adding, searching, deleting, and listing contacts.
- Output Cell:** Displays the execution of the code, creating a manager_array object and performing various operations like adding contacts "Boda Manishwar" and "chaithu", searching for "Boda Manishwar", and deleting "chaithu".
- Bottom Navigation:** Buttons for "How can I install Python libraries?", "Load data from Google Drive?", and "Show an example of training?"
- Bottom Status Bar:** Gemini 2.5 Flash

OUTPUT:

The screenshot shows a Google Colab interface with two tabs open: "Array-based contact manager" and "Untitled50.ipynb - Colab". The "Untitled50.ipynb" tab is active, displaying a Python script for managing contacts using an array. The code includes methods for adding, searching, and deleting contacts, along with a function to list all contacts. The output pane shows the execution results, including successful contact additions, a list of all contacts, and a search result for an unknown person.

```
phone = manager_array.search_contact("Bodla Manishwar")
print("Phone for Bodla Manishwar : {phone}\n")
manager_array.delete_contact("chaithu")
manager_array.list_all_contacts()
manager_array.delete_contact("Unknown Person")

=====
Array-Based Contact Manager
=====
Contact 'Bodla Manishwar' added successfully.
Contact 'chaithu' added successfully.
Contact 'Hitesh' added successfully.

--- All Contacts (Array-based) ---
1. Bodla Manishwar: 9391457697
2. chaithu: 9876543210
3. Hitesh: 5678902130

Phone for Bodla Manishwar : 9391457697
Contact 'chaithu' deleted successfully.

--- All Contacts (Array-based) ---
1. Bodla Manishwar: 9391457697
2. Hitesh: 5678902130

Contact 'Unknown Person' not found.
```

CODE EXPLANATION:

- 1) The program creates a Contact Manager using a Python list to store contacts as dictionaries with name and phone number.
- 2) The “add_contact()” method adds a new contact to the list using the “append()” function.
- 3) The “search_contact()” method checks each contact one by one to find a matching name.
- 4) The “delete_contact()” method finds the contact and removes it from the list using the `del` statement.
- 5) Adding is fast ($O(1)$), but searching and deleting take more time ($O(n)$) because the list must be scanned.

TASK 2:

PROMPT:

Generate a Python Library Book Request System using Queue and Priority Queue. Implement enqueue() and dequeue() methods and prioritize faculty requests over student requests with test cases.

CODE:

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains Python code for creating two types of queues: a normal queue using `collections.deque` and a priority queue using `heapq`. The code demonstrates enqueueing ("Mani", "Student") and ("Dr. Rao", "Faculty") into both queues, and dequeuing them. The output shows the normal queue processing them in FIFO order, while the priority queue processes them based on their priority levels (0 for Faculty, 1 for Student).

```
[2]: from collections import deque
import heapq
class LibraryQueue:
    def __init__(self):
        self.queue = deque()

    def enqueue(self, name, role):
        self.queue.append((name, role))
        print(f"{name} added to normal queue.")

    def dequeue(self):
        if self.queue:
            return self.queue.popleft()
        return "Queue is empty"
class LibraryPriorityQueue:
    def __init__(self):
        self.pq = []

    def enqueue(self, name, role):
        # Faculty gets higher priority (0), Student gets lower priority (1)
        priority = 0 if role == "Faculty" else 1
        heapq.heappush(self.pq, (priority, name, role))
        print(f"{name} added to priority queue.")

    def dequeue(self):
        if self.pq:
            return heapq.heappop(self.pq)
        return "Priority Queue is empty"
print("Normal Queue:")
q = LibraryQueue()
q.enqueue("Mani", "Student")
q.enqueue("Dr. Rao", "Faculty")
print("Served:", q.dequeue())
print("Served:", q.dequeue())
print("\nPriority Queue:")
pq = LibraryPriorityQueue()
pq.enqueue("Mani", "Student")
pq.enqueue("Dr. Rao", "Faculty")
print("Served:", pq.dequeue())
print("Served:", pq.dequeue())
```

The right side of the interface features the Gemini AI sidebar, which includes a greeting, a search bar, and buttons for "How can I install Python libraries?", "Load data from Google Drive", and "Show an example of training a simple ML model".

OUTPUT:

The screenshot shows the execution output of the code cell from the previous screenshot. The output displays the results of the enqueue and dequeue operations for both the normal queue and the priority queue.

```
... Normal Queue:  
Mani added to normal queue.  
Dr. Rao added to normal queue.  
Served: ('Mani', 'Student')  
Served: ('Dr. Rao', 'Faculty')  
  
Priority Queue:  
Mani added to priority queue.  
Dr. Rao added to priority queue.  
Served: (0, 'Dr. Rao', 'Faculty')  
Served: (1, 'Mani', 'Student')
```

The right side of the interface features the Gemini AI sidebar, which includes a greeting, a search bar, and buttons for "How can I install Python libraries?", "Load data from Google Drive", and "Show an example of training a simple ML model".

CODE EXPLANATION:

- 1) The normal queue uses FIFO (First-In-First-Out), so requests are served in the order they arrive.
- 2) The enqueue() method adds requests to the queue, and dequeue() removes the first request.
- 3) The priority queue uses the heapq module to assign priority values.
- 4) Faculty requests are given higher priority (0) than student requests (1), so they are served first.
- 5) This ensures correct prioritization while maintaining efficient insertion and removal operations.

TASK 3:

PROMPT:

Generate a Python Emergency Help Desk system using a Stack data structure. Implement push(), pop(), peek(), is_empty(), and is_full() methods and simulate five support tickets.

CODE:

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains the following Python code for a stack-based help desk system:

```
class HelpDeskStack:  
    def __init__(self, capacity=5):  
        self.stack = []  
        self.capacity = capacity  
    def push(self, ticket):  
        if self.is_full():  
            print("Stack is full. Cannot add ticket.")  
        else:  
            self.stack.append(ticket)  
            print(f"Ticket '{ticket}' added.")  
    def pop(self):  
        if self.is_empty():  
            return "No tickets to resolve."  
        return f"Resolved Ticket: {self.stack.pop()}"  
    def peek(self):  
        if self.is_empty():  
            return "No tickets available."  
        return f"Latest Ticket: {self.stack[-1]}"  
    def is_empty(self):  
        return len(self.stack) == 0  
    def is_full(self):  
        return len(self.stack) == self.capacity  
helpdesk = HelpDeskStack()  
helpdesk.push("Wifi not working")  
helpdesk.push("Email login issue")  
helpdesk.push("System crash")  
helpdesk.push("Printer not responding")  
helpdesk.push("Software installation error")  
print(helpdesk.peek())  
print(helpdesk.pop())  
print(helpdesk.pop())  
print(helpdesk.pop())
```

To the right of the code cell, there is a Gemini AI interface window. It displays a greeting: "Hello, MANISHWAR". Below it are three buttons: "How can I install Python libraries?", "Load data from Google Drive", and "Show an example of training a simple ML model". At the bottom of the Gemini window, there is a text input field with the placeholder "What can I help you build?" and a "Gemini 2.5 Flash" button.

OUTPUT:

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". On the left, there's a code editor window with the following Python code:

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)

stack = Stack()
for ticket in ["WiFi not working", "Email login issue", "System crash", "Printer not responding", "Software installation error"]:
    stack.push(ticket)
print(stack.pop())
print(stack.pop())
print(stack.pop())
print(stack.pop())
print(stack.pop())
print(stack.is_empty())
```

The output of this code is displayed in the terminal window:

```
...  
Ticket 'WiFi not working' added.  
Ticket 'Email login issue' added.  
Ticket 'System crash' added.  
Ticket 'Printer not responding' added.  
Ticket 'Software installation error' added.  
Latest Ticket: Software installation error  
Resolved Ticket: Software installation error  
Resolved Ticket: Printer not responding  
Resolved Ticket: System crash
```

On the right, there's a Gemini AI interface window. It says "Hello, MANISHWAR" and "How can I help you today?". It has three buttons: "How can I install Python libraries?", "Load data from Google Drive", and "Show an example of training a simple ML model". Below that, it says "What can I help you build?" and "Gemini 2.5 Flash".

EXPLANATION:

- 1) The stack follows the LIFO (Last-In, First-Out) principle, meaning the most recent ticket is resolved first.
- 2) The `push()` method adds a new ticket to the top of the stack if it is not full.
- 3) The `pop()` method removes and returns the most recently added ticket.
- 4) The `peek()` method shows the latest ticket without removing it from the stack.
- 5) The `is_empty()` and `is_full()` methods check the stack status to prevent errors during operations.

TASK 4:

PROMPT:

Generate a Python Hash Table implementation using chaining for collision handling. Include insert, search, and delete methods with proper comments and test cases.

CODE:

The screenshot shows a Google Colab notebook titled "Untitled31.ipynb". The code cell contains the following Python code for a HashTable class:

```
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]
    def hash_function(self, key):
        return hash(key) % self.size
    def insert(self, key, value):
        index = self.hash_function(key)
        bucket = self.table[index]
        # Check if key already exists
        for i, (k, v) in enumerate(bucket):
            if k == key:
                bucket[i] = (key, value) # Update value
                return
        bucket.append((key, value)) # Add new key-value pair
    def search(self, key):
        index = self.hash_function(key)
        bucket = self.table[index]
        for k, v in bucket:
            if k == key:
                return v
        return None
    def delete(self, key):
        index = self.hash_function(key)
        bucket = self.table[index]
        for i, (k, v) in enumerate(bucket):
            if k == key:
                del bucket[i]
                return "Key deleted"
        return "Key not found"
ht = HashTable()
ht.insert("Mani", "9876543210")
ht.insert("Rahul", "9123456780")
print("Search Mani:", ht.search("Mani"))
print(ht.delete("Rahul"))
print("Search Rahul:", ht.search("Rahul"))
```

To the right of the code cell is a Gemini AI interface window. It displays a greeting "Hello, MANISHWAR" and several buttons for interacting with the AI.

OUTPUT:

The screenshot shows the execution output of the code in the previous screenshot. The code cell now displays the results of the print statements:

```
... Search Mani: 9876543210
Key deleted
Search Rahul: None
```

The Gemini AI interface remains on the right side of the screen, showing the same greeting and interaction options as before.

EXPLANATION:

- 1) The hash table stores data in buckets (lists) to handle collisions using chaining.
- 2) The hash_function() converts a key into an index within the table size.
- 3) The insert() method adds key-value pairs and updates the value if the key already exists.
- 4) The search() method checks the correct bucket and returns the value if found.
- 5) The delete() method removes the key-value pair from the bucket, ensuring proper collision handling.

TASK 5:

PROMPT:

Design a Campus Resource Management System and suggest appropriate data structures for each feature with justification. Generate Python code to implement one selected feature with proper methods and test cases.

CODE:

The screenshot shows a Google Colab interface with a dark theme. On the left, there's a sidebar with various icons for file operations. The main area contains a code cell with the following Python code:

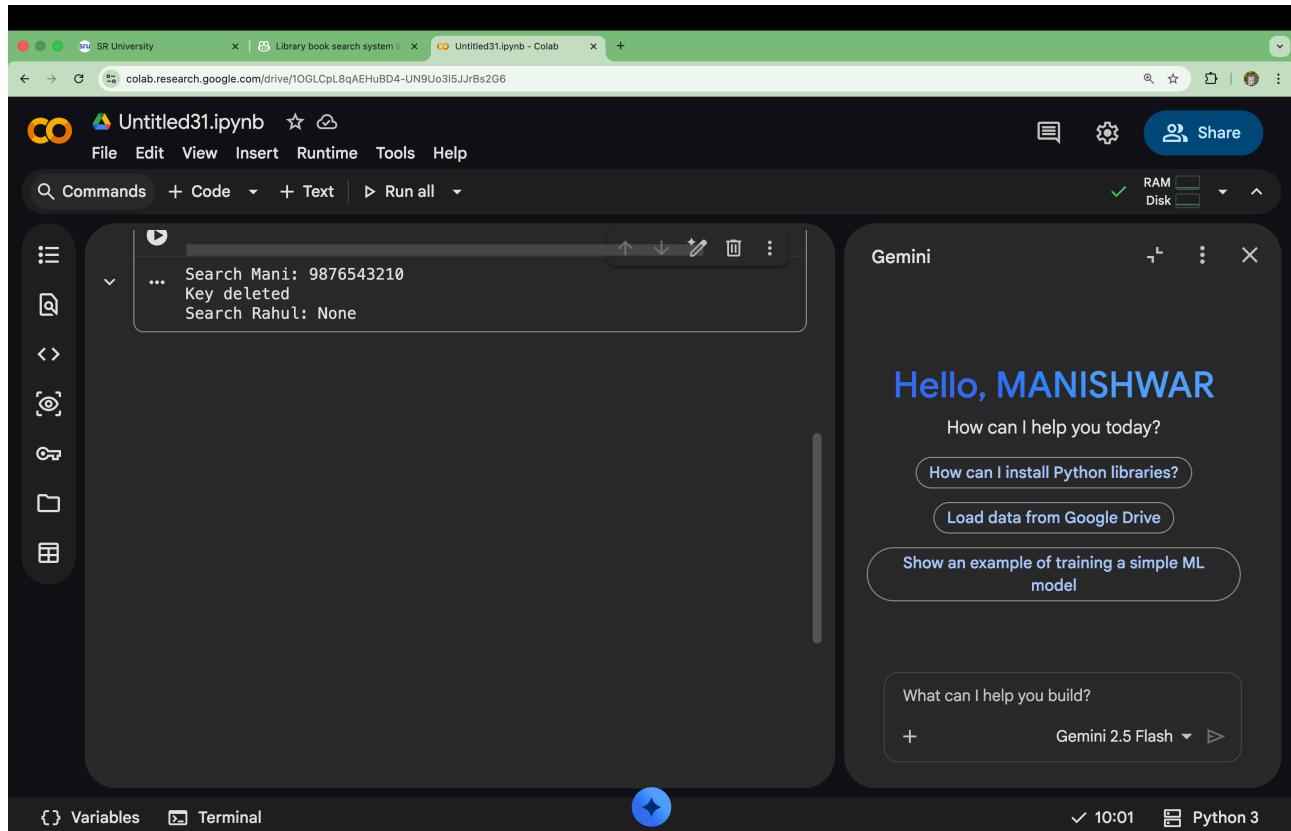
```
[6] class AttendanceSystem:  
    def __init__(self):  
        self.attendance = {} # Dictionary to store roll_no : status  
  
    def mark_attendance(self, roll_no, status):  
        self.attendance[roll_no] = status  
        print(f"Attendance marked for {roll_no}")  
  
    def check_attendance(self, roll_no):  
        return self.attendance.get(roll_no, "Record not found")  
  
    def delete_record(self, roll_no):  
        if roll_no in self.attendance:  
            del self.attendance[roll_no]  
            return "Record deleted"  
        return "Student not found"  
  
system = AttendanceSystem()  
system.mark_attendance("101", "Present")  
system.mark_attendance("102", "Absent")  
print("Student 101:", system.check_attendance("101"))  
print(system.delete_record('102'))  
print("Student 102:", system.check_attendance("102"))
```

To the right of the code cell is a Gemini AI sidebar. It displays a greeting message: "Hello, MANISHWAR". Below it are several AI-generated suggestions or related topics:

- How can I help you today?
- How can I install Python libraries?
- Load data from Google Drive
- Show an example of training a simple ML model

At the bottom of the sidebar, there are buttons for "What can I help you build?", "+", and "Gemini 2.5 Flash".

OUTPUT:



EXPLANATION:

- 1) The system uses a dictionary where roll numbers are keys and attendance status is values.
- 2) The `mark_attendance()` method inserts or updates attendance in O(1) time.
- 3) The `check_attendance()` method quickly retrieves attendance using the key lookup feature of dictionaries.
- 4) The `delete_record()` method removes a student record safely after checking existence.
- 5) Dictionary is efficient because it provides fast insertion, deletion, and searching operations.

