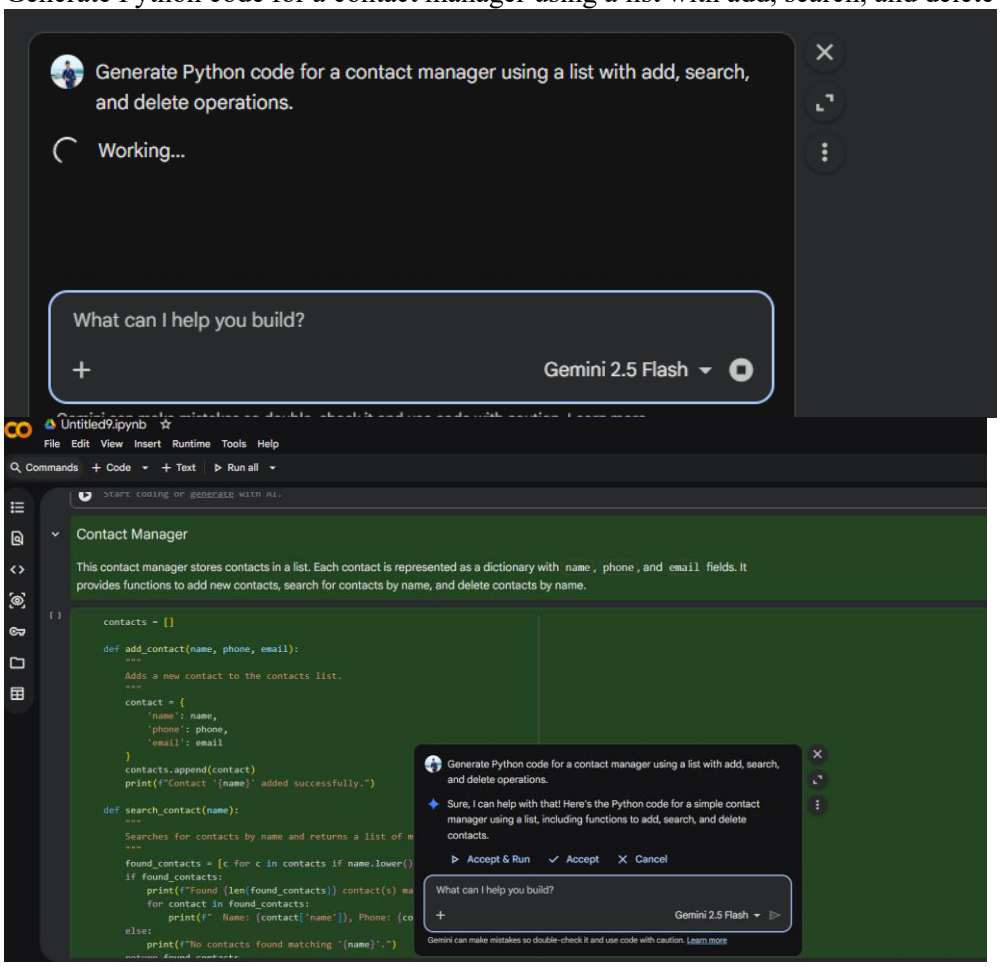
CH .VASU
2303a51618
Batch 25

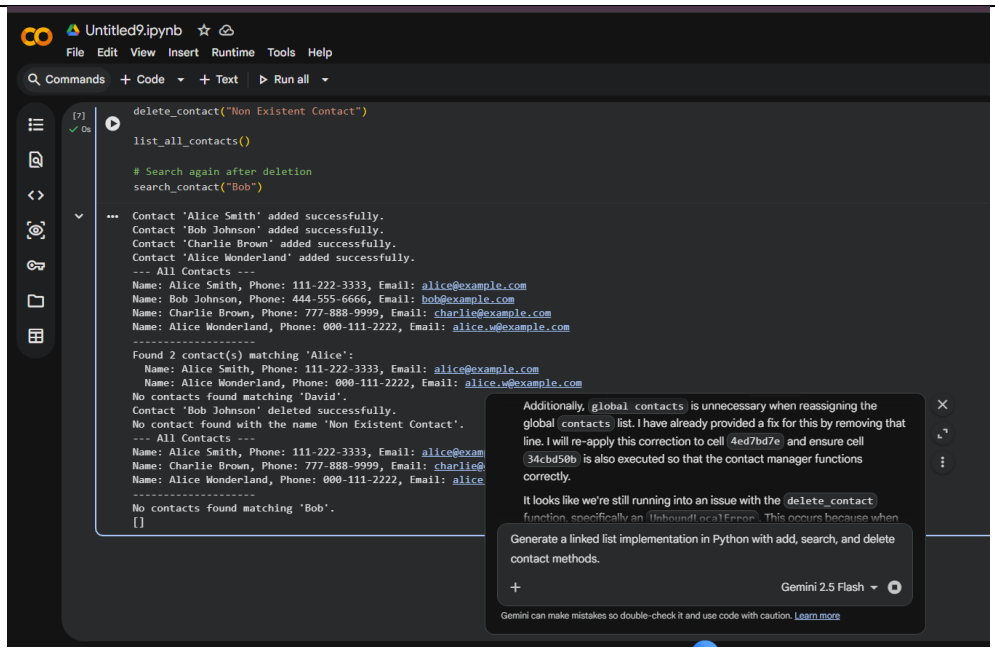
SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF C	
Program Name: B. Tech		**Assignment Type: Lab**	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
Course Code	23CS002PC304	**Course Title**	AI Assisted
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week6 – Wednesday**	**Time(s)**	23CSBTB0
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number:11.3(Present assignment number)/**24**(Total number of assignments)			

	Question
1	**Lab 11: Data Structures with AI Implementing Fundamental Data Structures using AI Assistance** **Lab Objectives:** By the end of this lab, students will be able to: • Design and implement fundamental data structures in Python using AI assistance. • Effectively prompt AI tools (e.g., GitHub Copilot) for code generation, optimization, and documentation. • Understand and compare core data structures: Arrays, Linked Lists, Stacks, Queues, Priority Queues, Tree • Improve code readability, efficiency, and maintainability using AI-generated suggestions. **Learning Outcomes** After completing this lab, students will be able to: • Apply appropriate data structures to solve real-world problems. • Analyze time and space complexity of different data structure operations. • Use AI tools responsibly to assist (not replace) logical thinking and problem-solving. • Validate, test, and refine AI-generated code. --- **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 pho 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. ---

Prompt
Generate Python code for a contact manager using a list with add, search, and delete operations.



Generate a linked list implementation in Python with add, search, and delete contact methods.



Conclusion:

Linked lists are more efficient for frequent insertions and deletions, while arrays are simpler and faster for index

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 mu** requests.

Tasks

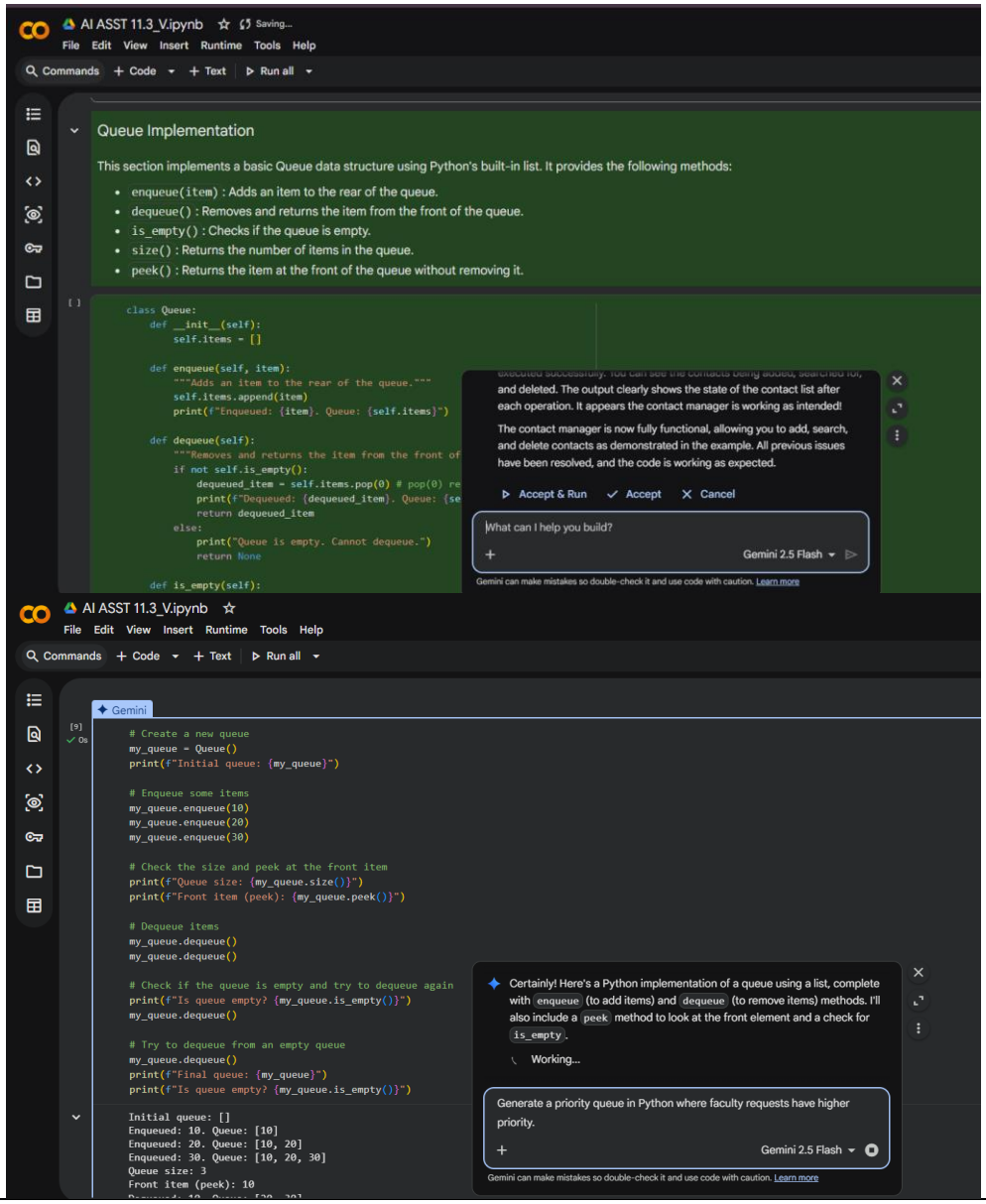
1. Implement a **Queue (FIFO)** to manage book requests.
2. Extend the system to a **Priority Queue**, prioritizing faculty requests.
3. Use **GitHub Copilot** to assist in generating:
    - enqueue() method
    - dequeue() method
4. Test the system with a mix of student and faculty requests.

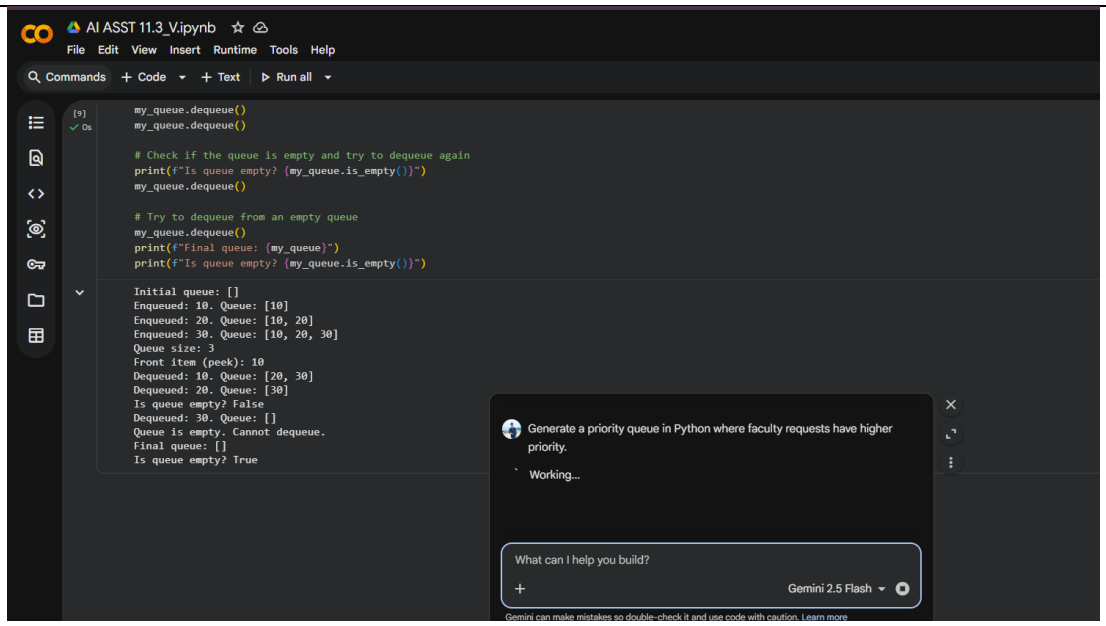
Expected Outcome

- Working queue and priority queue implementations.
- Correct prioritization of faculty requests.

Prompt

Generate Python queue implementation with enqueue and dequeue methods.





Faculty requests are processed first due to higher priority (1).
Priority Queue improves fairness and urgency handling.

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 receive
follows a **Last-In, First-Out (LIFO)** approach.

Tasks

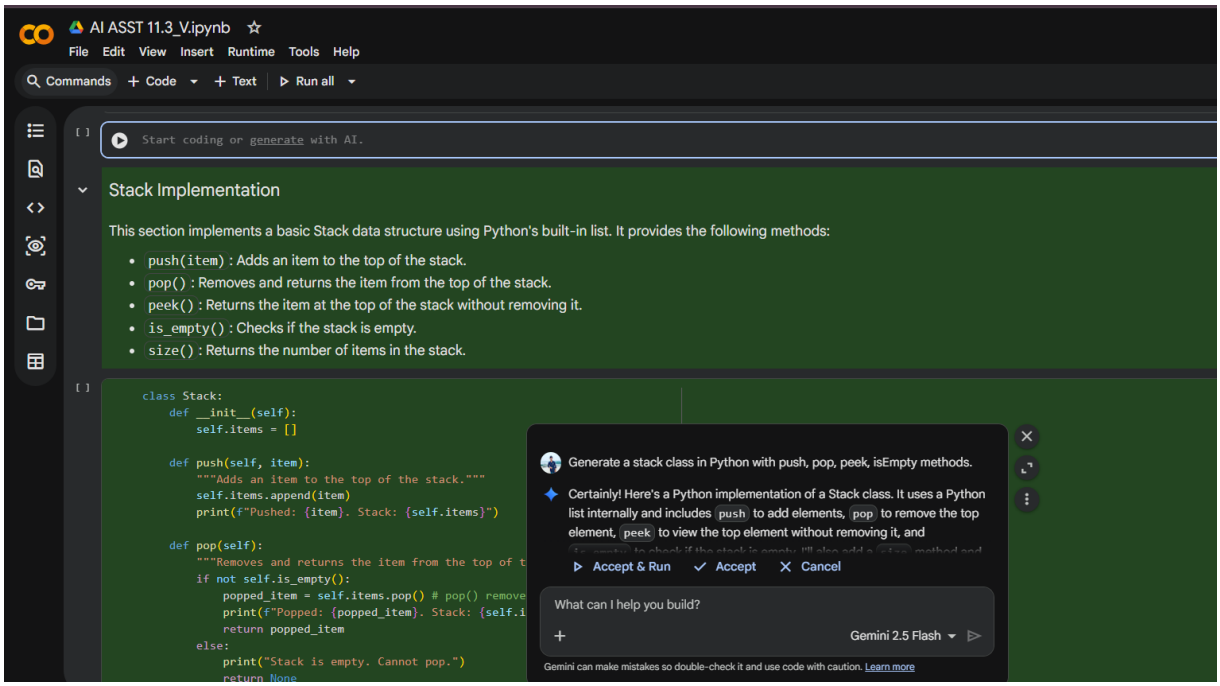
1. Implement a **Stack** to manage support tickets.
2. Provide the following operations:
   - push(ticket)
   - pop()
   - peek()
3. Simulate at least **five tickets** being raised and resolved.
4. Use **GitHub Copilot** to suggest additional stack operations such as:
   - Checking whether the stack is empty
   - Checking whether the stack is full (if applicable)

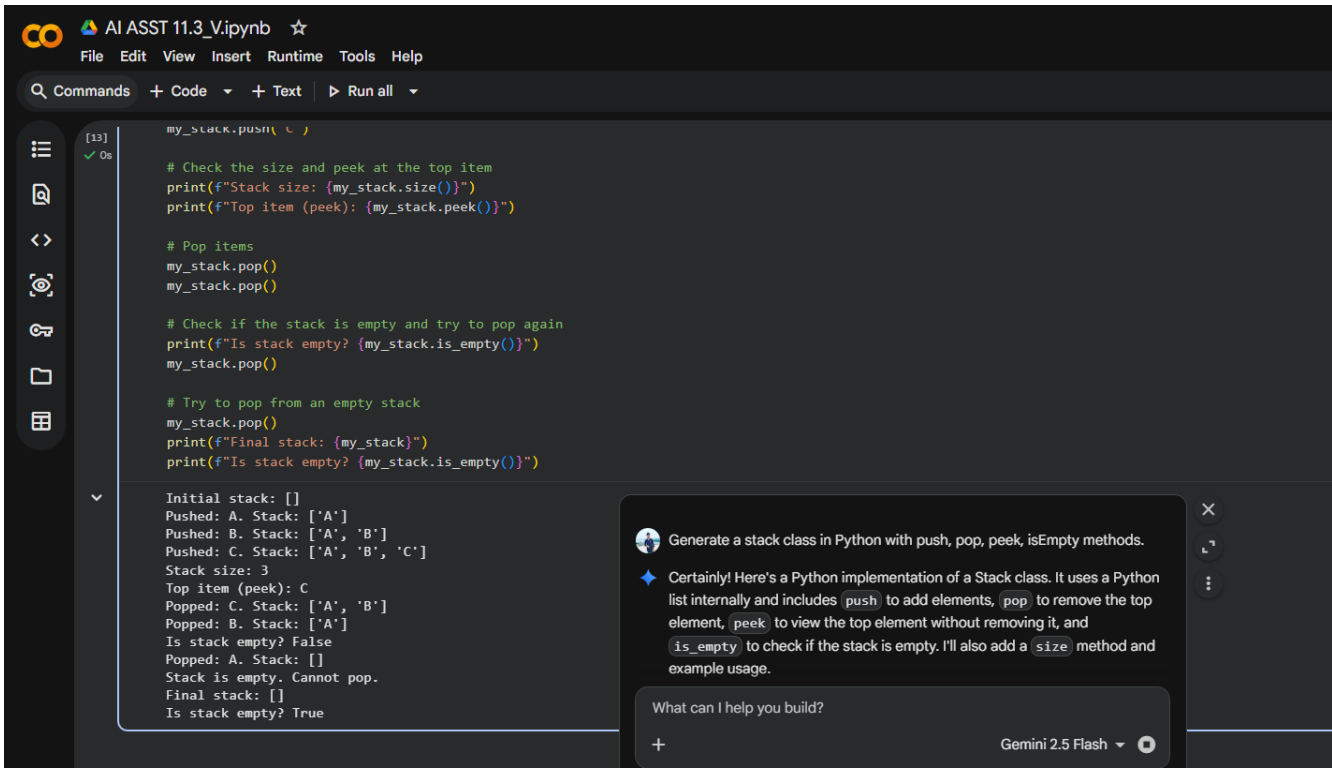
Expected Outcome

- Functional stack-based ticket management system.
- Clear demonstration of LIFO behavior.

Prompt

Generate a stack class in Python with push, pop, peek, isEmpty methods.





Stack follows **LIFO**. Latest ticket is resolved first.

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:

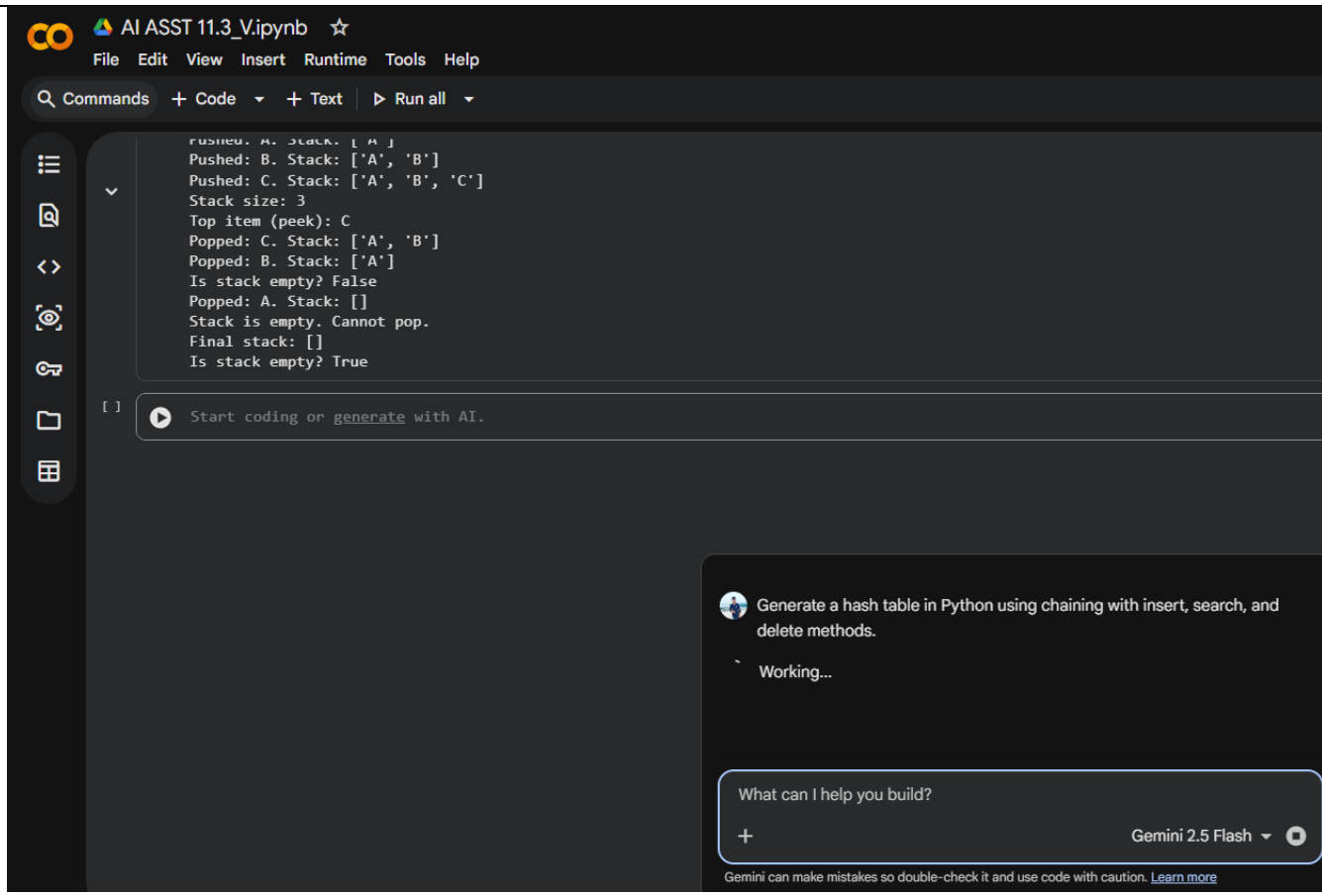
pass

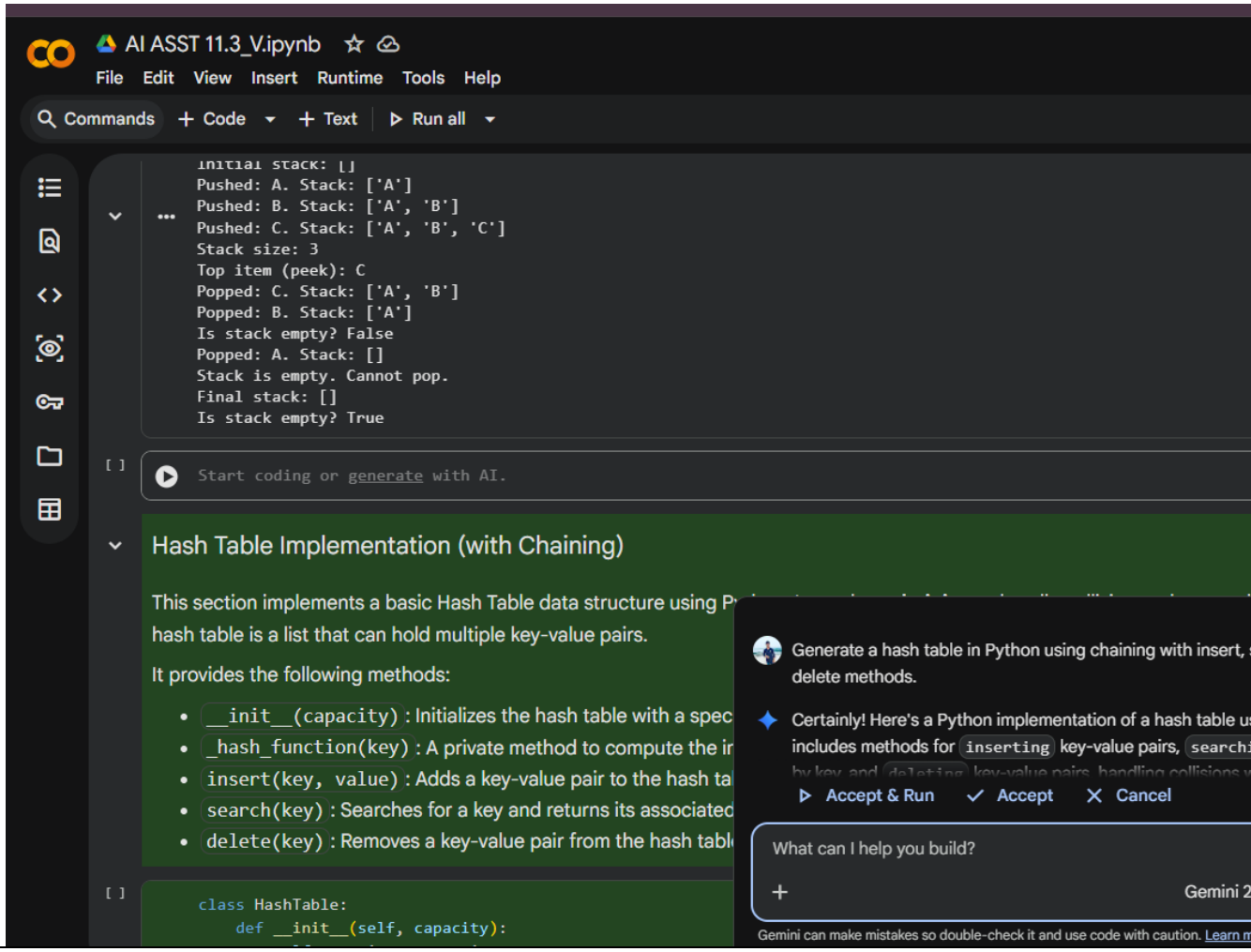
Expected Outcome

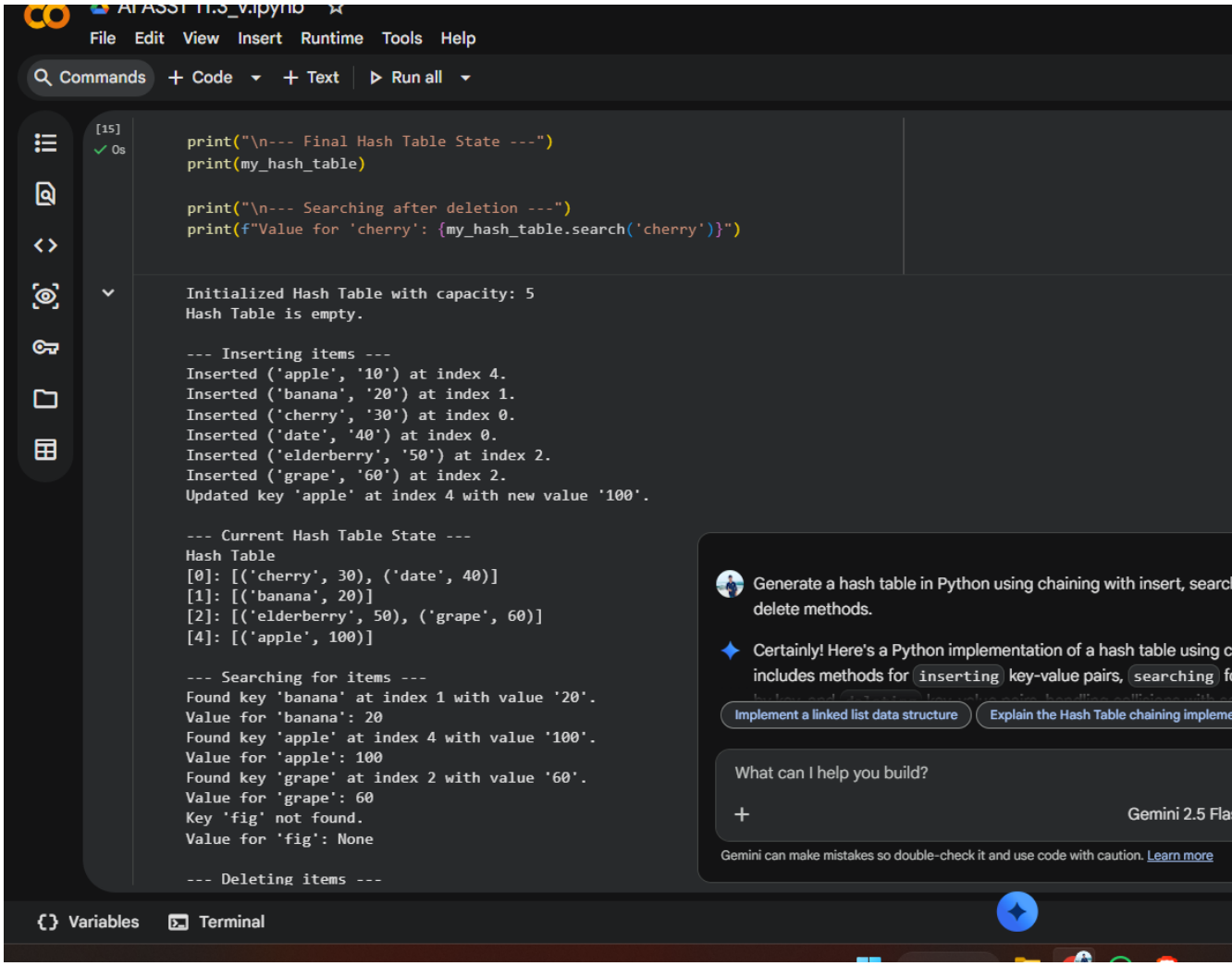
- Collision handling using chaining
- Well-commented methods

Prompt

Generate a hash table in Python using chaining with insert, search, and delete methods.


AI ASST 11.3_V.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Pushed: B. Stack: ['A', 'B']
Pushed: C. Stack: ['A', 'B', 'C']
Stack size: 3
Top item (peek): C
Popped: C. Stack: ['A', 'B']
Popped: B. Stack: ['A']
Is stack empty? False
Popped: A. Stack: []
Stack is empty. Cannot pop.
Final stack: []
Is stack empty? True
[ ]
Start coding or generate with AI.
Generate a hash table in Python using chaining with insert, search, and delete methods.
Working...
What can I help you build?
Gemini 2.5 Flash
Gemini can make mistakes so double-check it and use code with caution. Learn more


AI ASST 11.3_V.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Initial stack: []
Pushed: A. Stack: ['A']
Pushed: B. Stack: ['A', 'B']
Pushed: C. Stack: ['A', 'B', 'C']
Stack size: 3
Top item (peek): C
Popped: C. Stack: ['A', 'B']
Popped: B. Stack: ['A']
Is stack empty? False
Popped: A. Stack: []
Stack is empty. Cannot pop.
Final stack: []
Is stack empty? True
Start coding or generate with AI.
Hash Table Implementation (with Chaining)
This section implements a basic Hash Table data structure using P
hash table is a list that can hold multiple key-value pairs.
It provides the following methods:
__init__(capacity) : Initializes the hash table with a spec
_hash_function(key) : A private method to compute the ir
insert(key, value) : Adds a key-value pair to the hash ta
search(key) : Searches for a key and returns its associated
delete(key) : Removes a key-value pair from the hash tabl
class HashTable:
def __init__(self, capacity):
Generate a hash table in Python using chaining with insert,
delete methods.
Certainly! Here's a Python implementation of a hash table u
includes methods for inserting key-value pairs, searchi
Accept & Run Accept Cancel
What can I help you build?
Gemini 2
Gemini can make mistakes so double-check it and use code with caution. Learn m



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 expla
screenshots.

Task 5: Real-Time Application Challenge

◇ Mapping Table

Feature	Data Structure	Justification
Attendance	Hash Table	Fast lookup
Event Registration	Queue	FIFO
Library Borrowing	Priority Queue	Faculty priority
Bus Scheduling	Queue	Sequential order
Cafeteria Orders	Queue	Order of arrival

Conclusion

AI tools like GitHub Copilot helped generate boilerplate code faster, but logical understanding and testing were essential to verify correctness and efficiency.