

ASSIGNMENT- 11.3

Name:E.POOJITHA

HT.No: 2303A51356

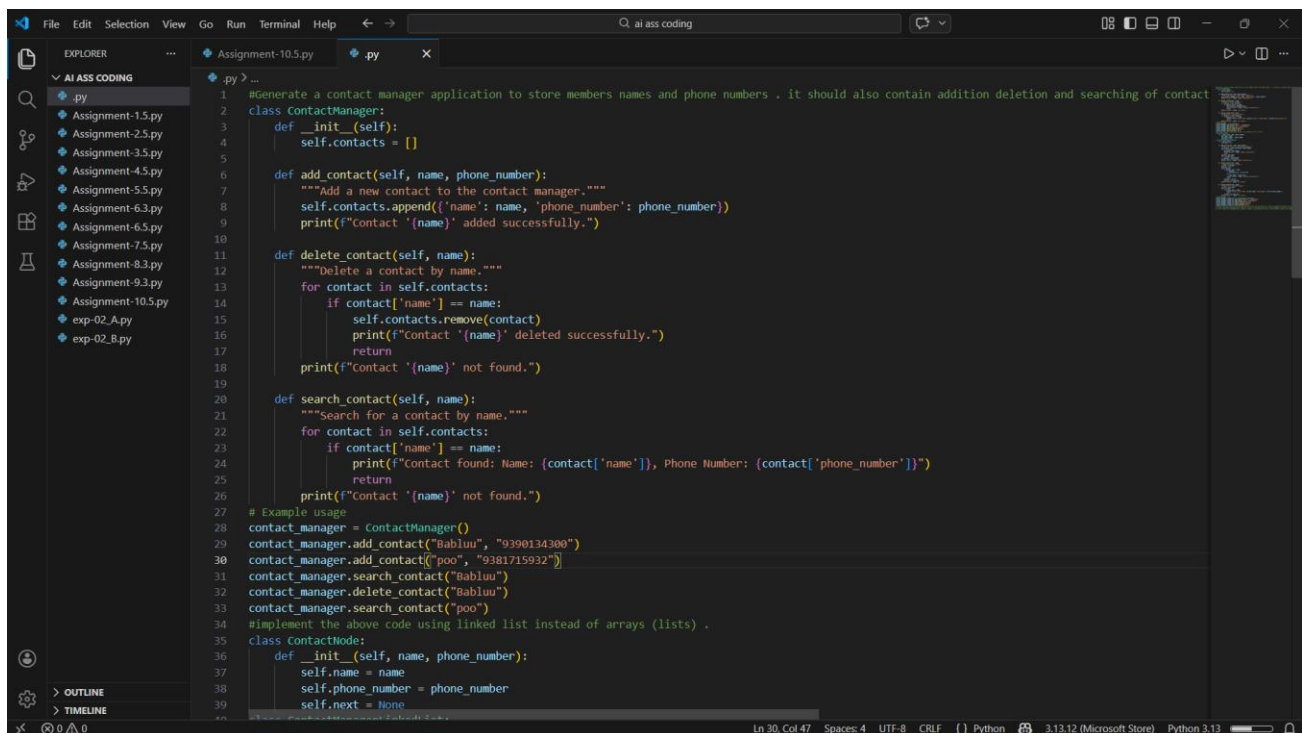
Batch: 20

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

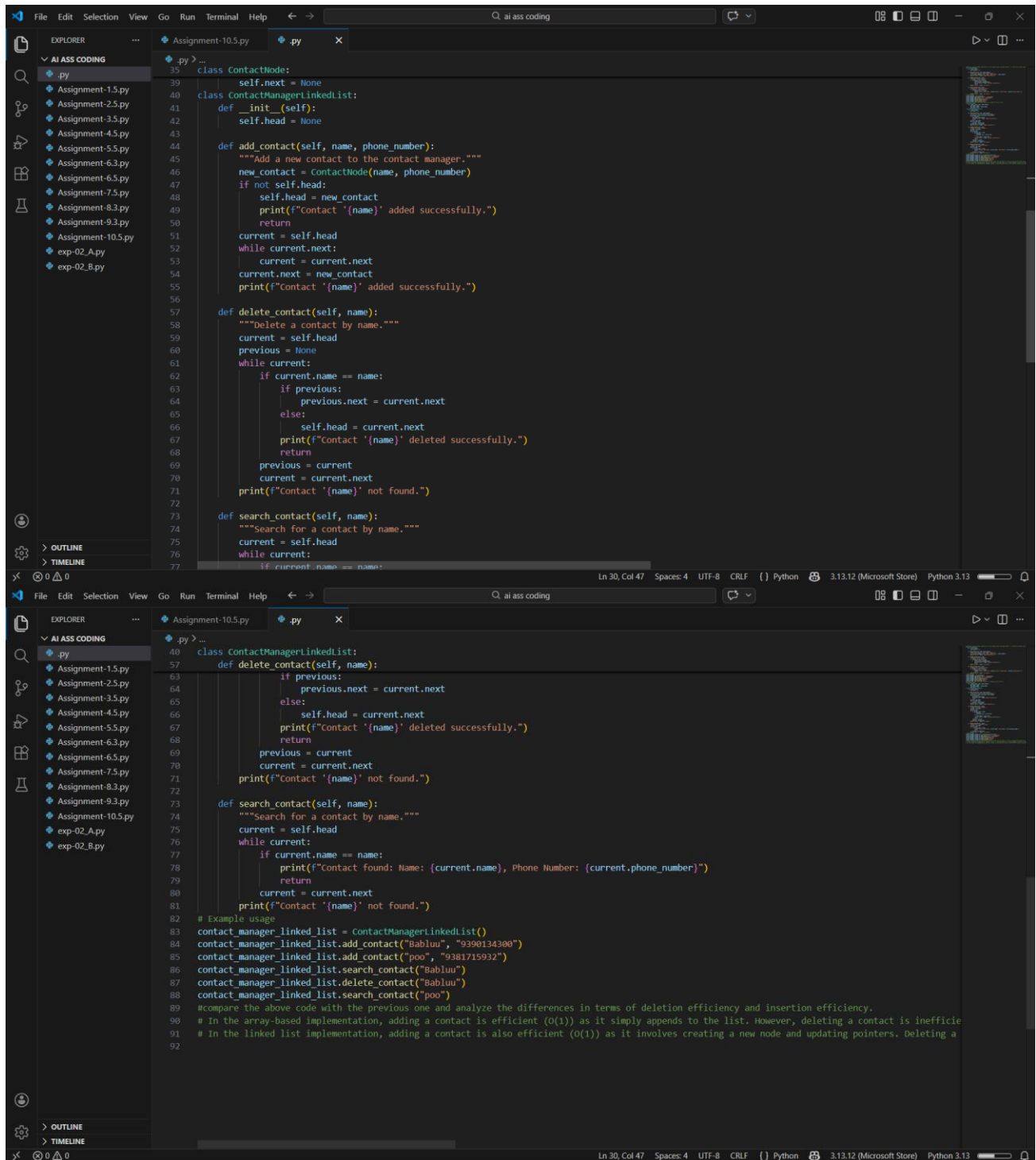
Sample input

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

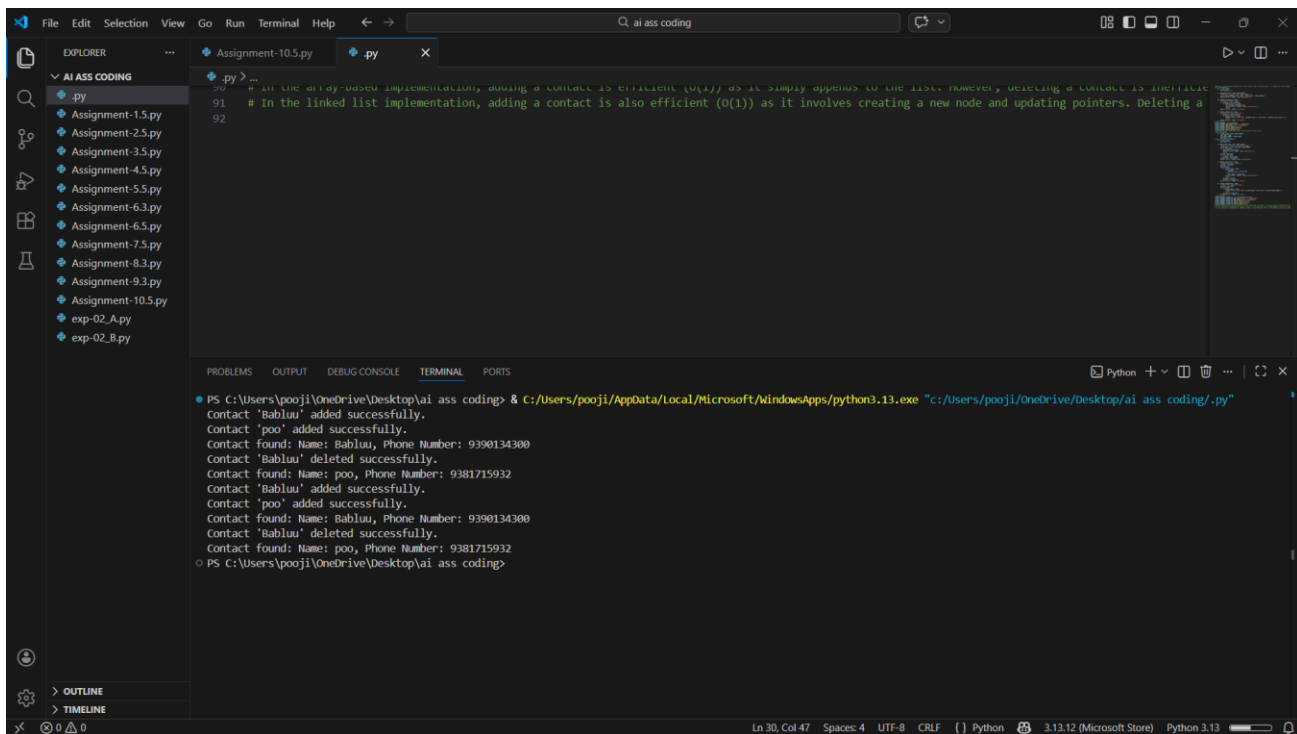
Code & probmt



```
1 #Generate a contact manager application to store members names and phone numbers . it should also contain addition deletion and searching of contact
2 class ContactManager:
3     def __init__(self):
4         self.contacts = []
5
6     def add_contact(self, name, phone_number):
7         """Add a new contact to the contact manager."""
8         self.contacts.append({'name': name, 'phone number': phone_number})
9         print(f"Contact '{name}' added successfully.")
10
11     def delete_contact(self, name):
12         """Delete a contact by name."""
13         for contact in self.contacts:
14             if contact['name'] == name:
15                 self.contacts.remove(contact)
16                 print(f"Contact '{name}' deleted successfully.")
17                 return
18         print(f"Contact '{name}' not found.")
19
20     def search_contact(self, name):
21         """Search for a contact by name."""
22         for contact in self.contacts:
23             if contact['name'] == name:
24                 print(f"Contact found: Name: {contact['name']}, Phone Number: {contact['phone number']}")
25                 return
26         print(f"Contact '{name}' not found.")
27
28 # Example usage
29 contact_manager = ContactManager()
30 contact_manager.add_contact("Babluu", "9390134300")
31 contact_manager.add_contact("poo", "9381715932")
32 contact_manager.search_contact("Babluu")
33 contact_manager.delete_contact("Babluu")
34 contact_manager.search_contact("poo")
35
36 # Implement the above code using linked list instead of arrays (lists) .
37 class ContactNode:
38     def __init__(self, name, phone_number):
39         self.name = name
40         self.phone_number = phone_number
41         self.next = None
```



Result:



```
File Edit Selection View Go Run Terminal Help
ai ass coding
EXPLORER
AI ASS CODING
py
Assignment-1.5.py
Assignment-2.5.py
Assignment-3.5.py
Assignment-4.5.py
Assignment-5.5.py
Assignment-6.5.py
Assignment-7.5.py
Assignment-8.3.py
Assignment-9.3.py
Assignment-10.5.py
exp-02_A.py
exp-02_B.py
Assignment-10.5.py
.py
90 # In the array-based implementation, adding a contact is efficient (O(1)) as it simply appends to the list. However, deleting a contact is inefficient
91 # In the linked list implementation, adding a contact is also efficient (O(1)) as it involves creating a new node and updating pointers. Deleting a
92
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:\Users\pooji\OneDrive\Desktop\ai ass coding\."
Contact 'Babluu' added successfully.
Contact 'poo' added successfully.
Contact found: Name: Babluu, Phone Number: 9390134300
Contact 'Babluu' deleted successfully.
Contact found: Name: poo, Phone Number: 9381715932
Contact 'Babluu' added successfully.
Contact 'poo' added successfully.
Contact found: Name: Babluu, Phone Number: 9390134300
Contact 'Babluu' deleted successfully.
Contact found: Name: poo, Phone Number: 9381715932
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation:

- The array-based contact manager allows simple and fast appending ($O(1)$) but requires shifting elements during deletion, making removals $O(n)$.
- The linked list version enables efficient insertion at the head ($O(1)$) without shifting on deletion, though searching remains $O(n)$ in both approaches.

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

Sample Input:

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.

Prompt & Code:

```
1 #Generate code for library book search syste, . students and faculty submit requests , but faculty requests have higher priority than student requests . Implement a queue to manage the
2 class Request:
3     def __init__(self, name, request_type):
4         self.name = name
5         self.request_type = request_type # 'faculty' or 'student'
6 class RequestQueue:
7     def __init__(self):
8         self.faculty_queue = []
9         self.student_queue = []
10    def enqueue(self, request):
11        if request.request_type == 'faculty':
12            self.faculty_queue.append(request)
13        elif request.request_type == 'student':
14            self.student_queue.append(request)
15    def dequeue(self):
16        if self.faculty_queue:
17            return self.faculty_queue.pop(0) # Process faculty requests first
18        elif self.student_queue:
19            return self.student_queue.pop(0) # Process student requests if no faculty requests
20        else:
21            return None # No requests to process
22 def main():
23     request_queue = RequestQueue()
24     while True:
25         print("Enter a request (type 'exit' to quit):")
26         name = input("Name: ")
27         if name.lower() == 'exit':
28             break
29         request_type = input("Request Type (faculty/student): ").lower()
30         if request_type in ['faculty', 'student']:
31             request = Request(name, request_type)
32             request_queue.enqueue(request)
33             print("Request from {name} added to the queue.")
34         else:
35             print("Invalid request type. Please enter 'faculty' or 'student'.")
36         # Process requests
37         processed_request = request_queue.dequeue()
38         if processed_request:
39             print(f"Processing {processed_request.request_type} request from {processed_request.name}.")
40         else:
41             print("No requests to process.")
42 if __name__ == "__main__":
43     main()
44
45
46
```

Result:

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
Enter a request (type 'exit' to quit):
Name: poojitha
Request Type (faculty/student): student
Request from poojitha added to the queue.
Processing student request from poojitha.
Enter a request (type 'exit' to quit):
Name: pooo
Request Type (faculty/student): student
Request from pooo added to the queue.
Processing student request from pooo.
Enter a request (type 'exit' to quit):
Name: akhila
Request Type (faculty/student): student
Request from akhila added to the queue.
Processing student request from akhila.
Enter a request (type 'exit' to quit):
Name: exit
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation:

- The priority queue ensures faculty requests are always processed before student requests, regardless of arrival order.
- Within each category (faculty or student), requests are handled in FIFO order, preserving fairness among the same requester type
-

Task 3: Emergency Help Desk (Stack Implementation)

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)

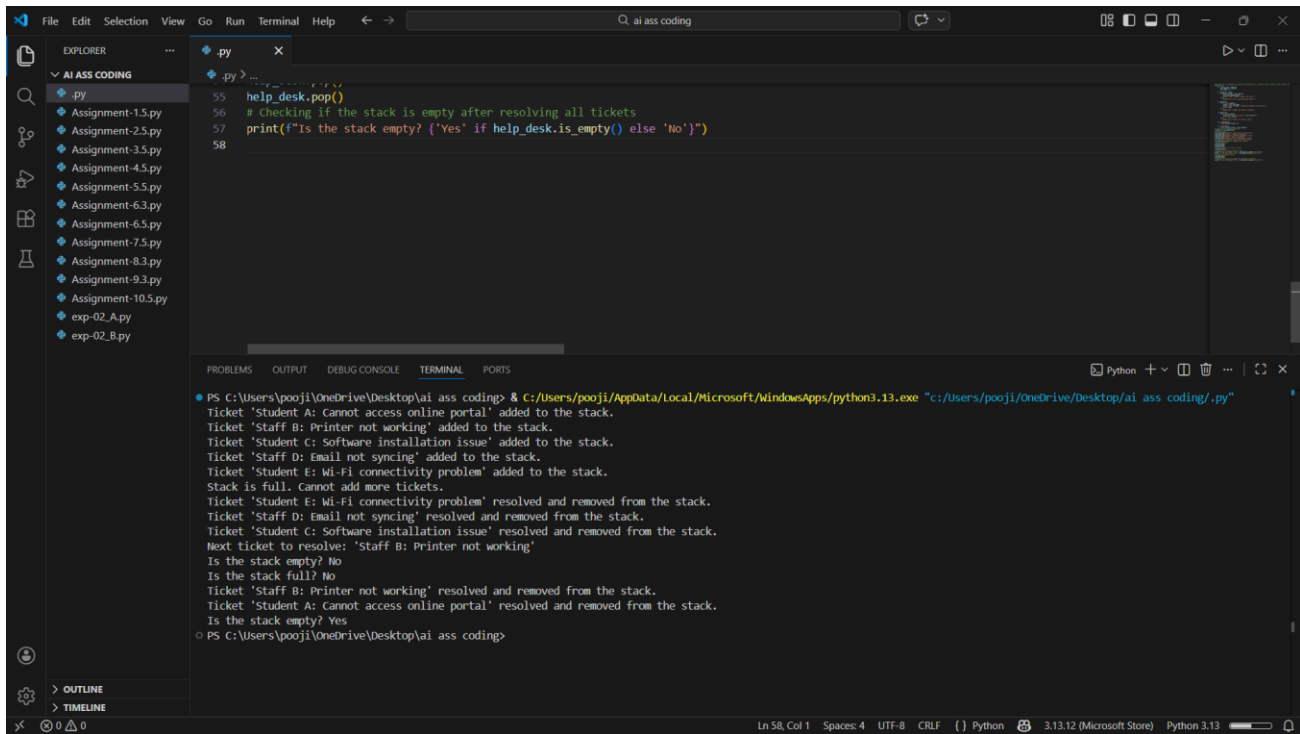
Prompt & Code:

```

1 # Generate code for emergency help desk(stack implementation) . It should receive technical support tickets from students and staff . while tickets are received they are followed by last in first out approach . Implement a stack to
2 class Stack:
3     def __init__(self, capacity):
4         self.capacity = capacity
5         self.stack = []
6
7     def push(self, ticket):
8         if len(self.stack) < self.capacity:
9             self.stack.append(ticket)
10            print(f"Ticket '{ticket}' added to the stack.")
11        else:
12            print("Stack is full. Cannot add more tickets.")
13
14    def pop(self):
15        if not self.is_empty():
16            ticket = self.stack.pop()
17            print(f"Ticket '{ticket}' resolved and removed from the stack.")
18            return ticket
19        else:
20            print("Stack is empty. No tickets to resolve.")
21
22    def peek(self):
23        if not self.is_empty():
24            print(f"Next ticket to resolve: '{self.stack[-1}]'"
25            return self.stack[-1]
26        else:
27            print("Stack is empty. No tickets to peek.")
28
29    def is_empty(self):
30        return len(self.stack) == 0
31
32    def is_full(self):
33        return len(self.stack) == self.capacity
34
35    # Simulating the emergency help desk
36    help_desk = Stack(capacity=5)
37
38    # Raising tickets
39    help_desk.push("Student A: Cannot access online portal")
40    help_desk.push("Staff B: Printer not working")
41    help_desk.push("Student C: Software installation issue")
42    help_desk.push("Staff D: Email not syncing")
43    help_desk.push("Student E: Wi-Fi connectivity problem")
44
45    # Attempting to add another ticket when stack is full
46    help_desk.push("Student F: Computer won't start")
47
48    # Resolving tickets
49    help_desk.pop()
50    help_desk.pop()
51    help_desk.pop()
52
53    # Peeking at the next ticket to resolve
54    help_desk.peek()
55
56    # Checking if the stack is empty or full
57    print(f"Is the stack empty? ('Yes' if help_desk.is_empty() else 'No')")
58    print(f"Is the stack full? ('Yes' if help_desk.is_full() else 'No')")
59
60    # Resolving remaining tickets
61    help_desk.pop()
62    help_desk.pop()
63
64    # Checking if the stack is empty after resolving all tickets
65    print(f"Is the stack empty? ('Yes' if help_desk.is_empty() else 'No')")

```

Result:



The screenshot shows a Visual Studio Code editor window with a file explorer on the left and a terminal at the bottom. The file explorer shows a folder named 'AI ASS CODING' containing several Python files. The main editor window displays a Python script with the following code:

```
55 help_desk.pop()
56 # checking if the stack is empty after resolving all tickets
57 print(f'Is the stack empty? {'Yes' if help_desk.is_empty() else 'No'})
58
```

The terminal window shows the output of the script, which simulates a ticket support system using a stack. The output is as follows:

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "c:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
Ticket 'Student A: Cannot access online portal' added to the stack.
Ticket 'Staff B: Printer not working' added to the stack.
Ticket 'Student C: Software installation issue' added to the stack.
Ticket 'Staff D: Email not syncing' added to the stack.
Ticket 'Student E: Wi-Fi connectivity problem' added to the stack.
Stack is full. Cannot add more tickets.
Ticket 'Student E: Wi-Fi connectivity problem' resolved and removed from the stack.
Ticket 'Staff D: Email not syncing' resolved and removed from the stack.
Ticket 'Student C: Software installation issue' resolved and removed from the stack.
Next ticket to resolve: 'Staff B: Printer not working'
Is the stack empty? No
Is the stack full? No
Ticket 'Staff B: Printer not working' resolved and removed from the stack.
Ticket 'Student A: Cannot access online portal' resolved and removed from the stack.
Is the stack empty? Yes
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation:

The stack follows the LIFO (Last In, First Out) principle, so the most recently added support ticket is processed first.

The `peek()` method allows viewing the top ticket without removing it, and attempting to `pop()` from an empty stack safely returns `None`

Task 4: Hash Table

Objective

To implement a Hash Table and understand collision handling

Sample input:

Task Description

Use AI to generate a hash table with:

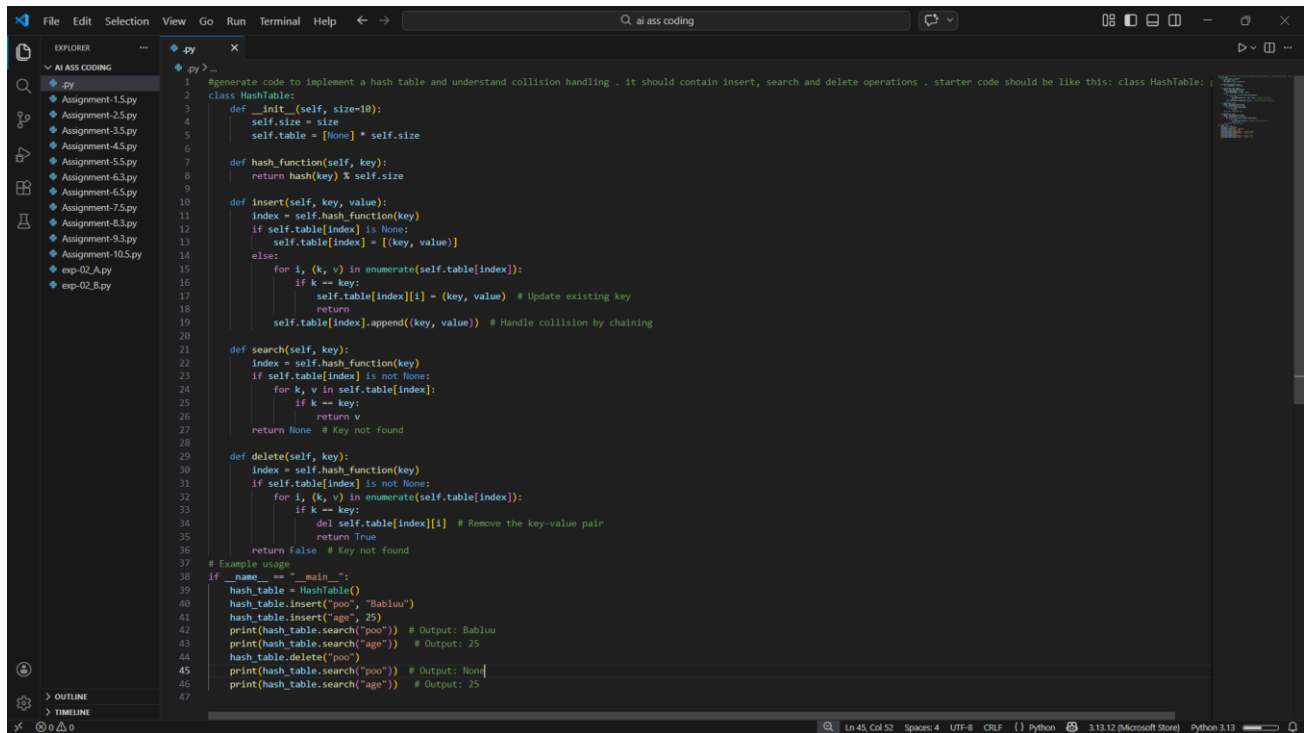
- Insert
- Search
- Delete

Starter Code

class HashTable:

pass

Prompt & Code:



```
1 #generate code to implement a hash table and understand collision handling . it should contain insert, search and delete operations . starter code should be like this: class HashTable:
2 class HashTable:
3     def __init__(self, size=10):
4         self.size = size
5         self.table = [None] * self.size
6
7     def hash_function(self, key):
8         return hash(key) % self.size
9
10    def insert(self, key, value):
11        index = self.hash_function(key)
12        if self.table[index] is None:
13            self.table[index] = [(key, value)]
14        else:
15            for i, (k, v) in enumerate(self.table[index]):
16                if k == key:
17                    self.table[index][i] = (key, value) # Update existing key
18            return
19            self.table[index].append((key, value)) # Handle collision by chaining
20
21    def search(self, key):
22        index = self.hash_function(key)
23        if self.table[index] is not None:
24            for k, v in self.table[index]:
25                if k == key:
26                    return v
27        return None # Key not found
28
29    def delete(self, key):
30        index = self.hash_function(key)
31        if self.table[index] is not None:
32            for i, (k, v) in enumerate(self.table[index]):
33                if k == key:
34                    del self.table[index][i] # Remove the key-value pair
35            return True
36        return False # Key not found
37
38 # Example usage
39 if __name__ == "__main__":
40     hash_table = HashTable()
41     hash_table.insert("poo", "Babluu")
42     print(hash_table.search("poo")) # Output: Babluu
43     print(hash_table.search("age")) # Output: 25
44     hash_table.delete("poo")
45     print(hash_table.search("poo")) # Output: None
46     print(hash_table.search("age")) # Output: 25
47
```

Result:

The screenshot shows a VS Code editor with a Python file named `ai ass coding.py`. The code implements a `HashTable` class with methods for `__init__`, `hash_function`, `insert`, and `search`. The `insert` method uses chaining to handle collisions. The `search` method returns `None` if a key is not found. The terminal output shows the execution of the code, which prints `None` for a search operation.

```
1 #generate code to implement a hash table and understand collision handling . it should contain insert, search and delete operations . starter code should be like this: class HashTable:
2 class HashTable:
3     def __init__(self, size=10):
4         self.size = size
5         self.table = [None] * self.size
6
7     def hash_function(self, key):
8         return hash(key) % self.size
9
10    def insert(self, key, value):
11        index = self.hash_function(key)
12        if self.table[index] is None:
13            self.table[index] = [(key, value)]
14        else:
15            for i, (k, v) in enumerate(self.table[index]):
16                if k == key:
17                    self.table[index][i] = (key, value) # Update existing key
18            return
19            self.table[index].append((key, value)) # Handle collision by chaining
20
21    def search(self, key):
22        index = self.hash_function(key)
23        if self.table[index] is not None:
24            for k, v in self.table[index]:
25                if k == key:
26                    return v
27        return None
```

Terminal Output:

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:/Users/pooji/AppData/Local/Microsoft/WindowsApps/python3.13.exe "C:/Users/pooji/OneDrive/Desktop/ai ass coding/.py"
None
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Observation

The hash table provides fast average-case time complexity of $O(1)$ for insertion, search, and deletion operations using key-based access.

After deleting a key, searching for it returns `None`, while other existing keys remain unaffected in the table.

Task 5: Real-Time Application Challenge

Sample input:

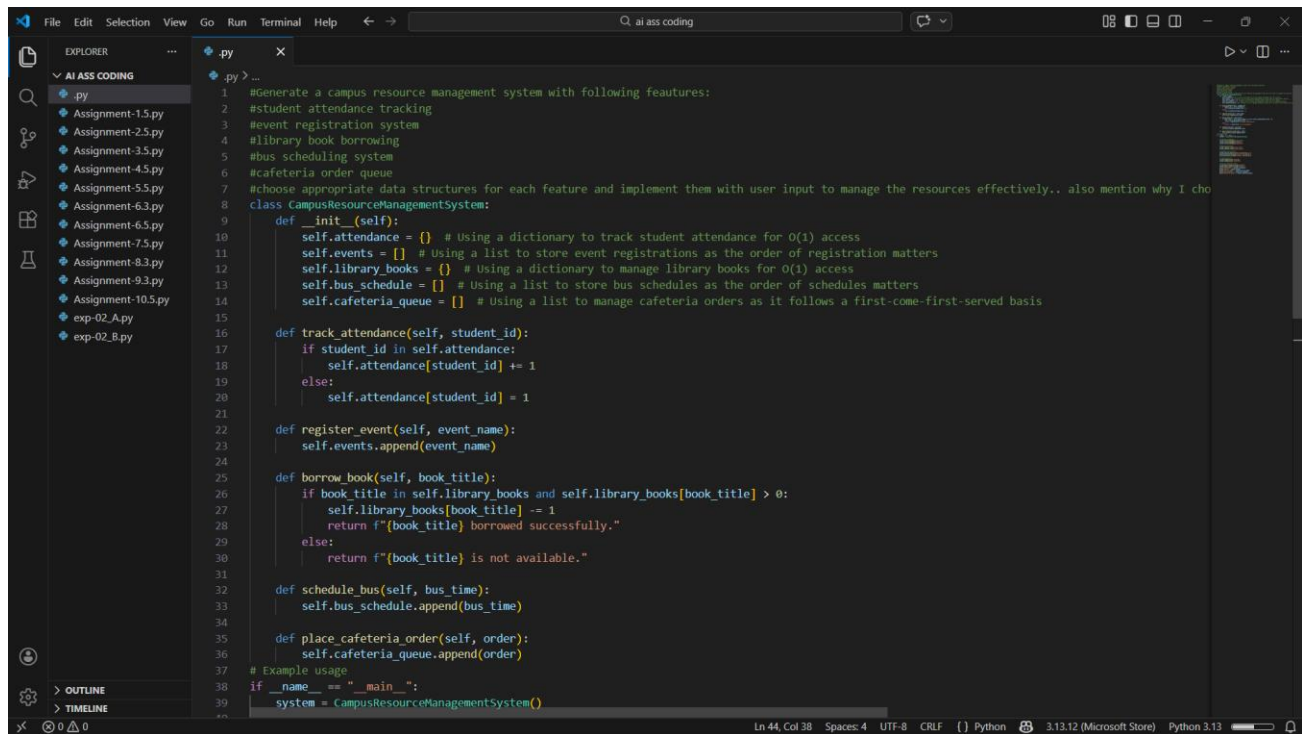
Design a Campus Resource Management System with the following features:

- 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.

Prompt & Code:



```
1 #Generate a campus resource management system with following feautres:
2 #student attendance tracking
3 #event registration system
4 #library book borrowing
5 #bus scheduling system
6 #cafeteria order queue
7 #choose appropriate data structures for each feature and implement them with user input to manage the resources effectively.. also mention why I cho
8 class CampusResourceManagementSystem:
9     def __init__(self):
10         self.attendance = {} # Using a dictionary to track student attendance for O(1) access
11         self.events = [] # Using a list to store event registrations as the order of registration matters
12         self.library_books = {} # Using a dictionary to manage library books for O(1) access
13         self.bus_schedule = [] # Using a list to store bus schedules as the order of schedules matters
14         self.cafeteria_queue = [] # Using a list to manage cafeteria orders as it follows a first-come-first-served basis
15
16     def track_attendance(self, student_id):
17         if student_id in self.attendance:
18             self.attendance[student_id] += 1
19         else:
20             self.attendance[student_id] = 1
21
22     def register_event(self, event_name):
23         self.events.append(event_name)
24
25     def borrow_book(self, book_title):
26         if book_title in self.library_books and self.library_books[book_title] > 0:
27             self.library_books[book_title] -= 1
28             return f"{book_title} borrowed successfully."
29         else:
30             return f"{book_title} is not available."
31
32     def schedule_bus(self, bus_time):
33         self.bus_schedule.append(bus_time)
34
35     def place_cafeteria_order(self, order):
36         self.cafeteria_queue.append(order)
37
38 # Example usage
39 if __name__ == "__main__":
40     system = CampusResourceManagementSystem()
```

```
File Edit Selection View Go Run Terminal Help
ai ass coding

EXPLORER
AI ASS CODING
.py
Assignment-1.5.py
Assignment-2.5.py
Assignment-3.5.py
Assignment-4.5.py
Assignment-5.5.py
Assignment-6.3.py
Assignment-6.5.py
Assignment-7.5.py
Assignment-8.3.py
Assignment-9.3.py
Assignment-10.5.py
exp-02_A.py
exp-02_B.py

.py
class CampusResourceManagementSystem:
    def schedule_bus(self, bus_time):
        self.bus_schedule.append(bus_time)

    def place_cafeteria_order(self, order):
        self.cafeteria_queue.append(order)

# Example usage
if __name__ == "__main__":
    system = CampusResourceManagementSystem()

    # Tracking attendance
    system.track_attendance("poojitha")
    system.track_attendance("akhila")
    system.track_attendance("poojitha")

    # Registering events
    system.register_event("Tech Talk")
    system.register_event("Sports Day")

    # Managing library books
    system.library_books["Python Programming"] = 5
    print(system.borrow_book("Python Programming"))
    print(system.borrow_book("Python Programming"))

    # Scheduling buses
    system.schedule_bus("8:00 AM")
    system.schedule_bus("12:00 PM")

    # Placing cafeteria orders
    system.place_cafeteria_order("Burger")
    system.place_cafeteria_order("Pizza")
    print("Attendance:", system.attendance)
    print("Events:", system.events)
    print("Library Books:", system.library_books)
    print("Bus Schedule:", system.bus_schedule)
    print("Cafeteria Queue:", system.cafeteria_queue)
```

Ln 44, Col 38 Spaces: 4 UTF-8 CRLF Python 3.13.12 (Microsoft Store) Python 3.13

Result:

```
File Edit Selection View Go Run Terminal Help
ai ass coding

EXPLORER
AI ASS CODING
.py
Assignment-1.5.py
Assignment-2.5.py
Assignment-3.5.py
Assignment-4.5.py
Assignment-5.5.py
Assignment-6.3.py
Assignment-6.5.py
Assignment-7.5.py
Assignment-8.3.py
Assignment-9.3.py
Assignment-10.5.py
exp-02_A.py
exp-02_B.py

.py
class CampusResourceManagementSystem:
    def schedule_bus(self, bus_time):
        self.bus_schedule.append(bus_time)

    def place_cafeteria_order(self, order):
        self.cafeteria_queue.append(order)

# Example usage
if __name__ == "__main__":
    system = CampusResourceManagementSystem()

    # Tracking attendance
    system.track_attendance("poojitha")
    system.track_attendance("akhila")
    system.track_attendance("poojitha")

    # Registering events
    system.register_event("Tech Talk")
    system.register_event("Sports Day")

    # Managing library books
    system.library_books["Python Programming"] = 5
    print(system.borrow_book("Python Programming"))
    print(system.borrow_book("Python Programming"))

    # Scheduling buses
    system.schedule_bus("8:00 AM")
    system.schedule_bus("12:00 PM")

    # Placing cafeteria orders
    system.place_cafeteria_order("Burger")
    system.place_cafeteria_order("Pizza")
    print("Attendance:", system.attendance)
    print("Events:", system.events)
    print("Library Books:", system.library_books)
    print("Bus Schedule:", system.bus_schedule)
    print("Cafeteria Queue:", system.cafeteria_queue)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding> & C:\Users\pooji\AppData\Local\Microsoft\WindowsApps\python3.13.exe "C:\Users\pooji\OneDrive\Desktop\ai ass coding\ai ass coding.py"
Python Programming borrowed successfully.
Python Programming borrowed successfully.
Attendance: {'poojitha': 2, 'akhila': 1}
Events: ['Tech Talk', 'Sports Day']
Library Books: {'Python Programming': 3}
Bus Schedule: ['8:00 AM', '12:00 PM']
Cafeteria Queue: ['Burger', 'Pizza']
PS C:\Users\pooji\OneDrive\Desktop\ai ass coding>
```

Ln 44, Col 38 Spaces: 4 UTF-8 CRLF Python 3.13.12 (Microsoft Store) Python 3.13

Observation:

The dictionary efficiently tracks attendance using student IDs as keys, allowing quick $O(1)$ access to each student's status.

The attendance report is generated by counting boolean values, making it simple to calculate total, present, and absent students dynamically.