

ASSIGNMENT-11.3

Name: T.Swetha

Ht.no: 2303A51317

Batch: 05

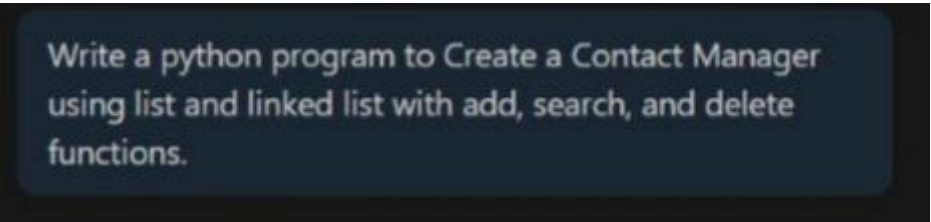
Task 1: Smart Contact Manager (Arrays & Linked Lists)

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.

Prompt:

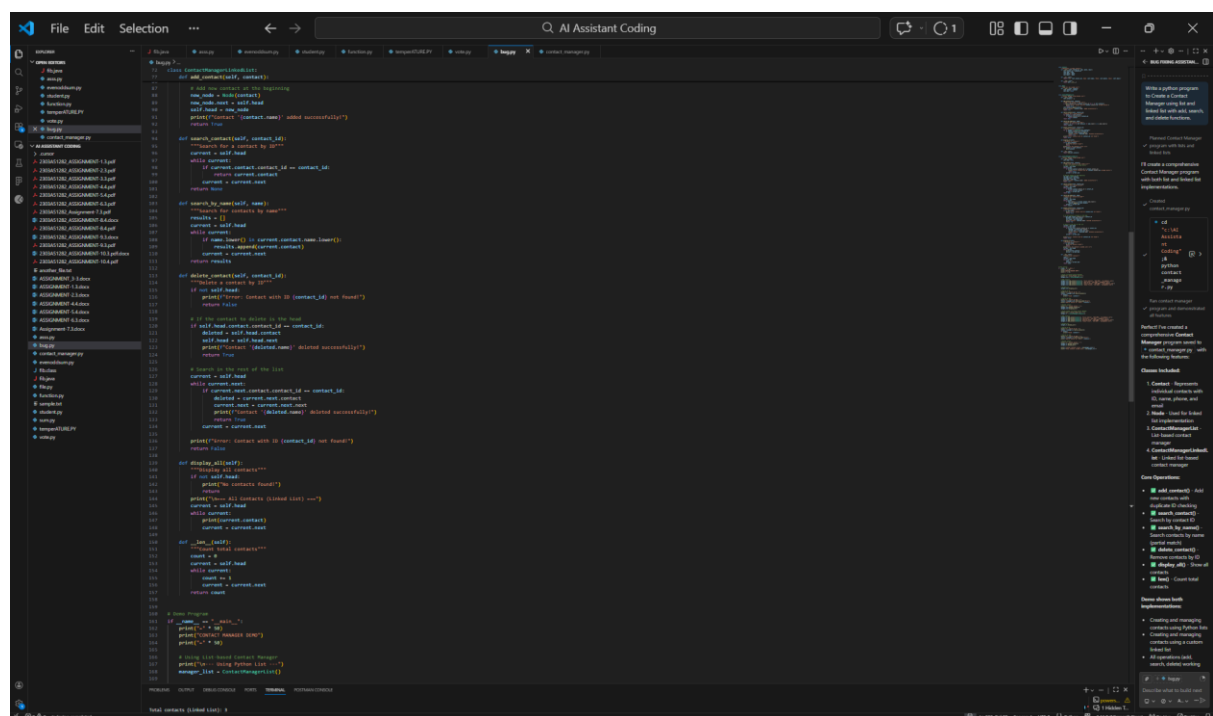
Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.

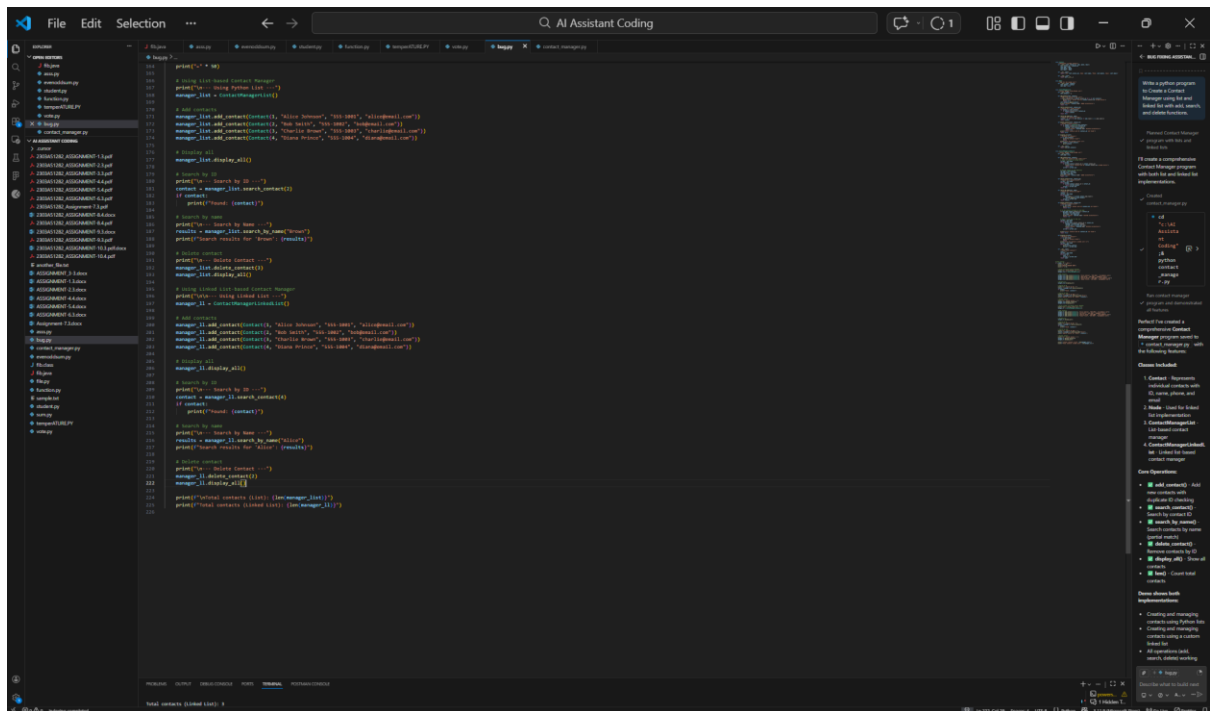


Write a python program to Create a Contact Manager using list and linked list with add, search, and delete functions.

```

1 class Contact:
2     """Represents a single contact"""
3     def __init__(self, contact_id, name, phone, email):
4         self.contact_id = contact_id
5         self.name = name
6         self.phone = phone
7         self.email = email
8
9     def __str__(self):
10         return f'({self.contact_id}, {name} ({self.phone}, {self.email})'
11
12     def __repr__(self):
13         return self.__str__()
14
15     def __eq__(self, other):
16         """Make for the linked list"""
17         return self.name == other.name
18
19     def __lt__(self, other):
20         """Make for the linked list"""
21         return self.name < other.name
22
23     def __gt__(self, other):
24         """Make for the linked list"""
25         return self.name > other.name
26
27     def __le__(self, other):
28         """Make for the linked list"""
29         return self.name <= other.name
30
31     def __ge__(self, other):
32         """Make for the linked list"""
33         return self.name >= other.name
34
35     def __ne__(self, other):
36         """Make for the linked list"""
37         return self.name != other.name
38
39     def __hash__(self):
40         """Make for the linked list"""
41         return hash(self.name)
42
43     def __iter__(self):
44         """Make for the linked list"""
45         return self
46
47     def __getitem__(self, index):
48         """Make for the linked list"""
49         return self
50
51     def __setitem__(self, index, value):
52         """Make for the linked list"""
53         return self
54
55     def __delitem__(self, index):
56         """Make for the linked list"""
57         return self
58
59     def __len__(self):
60         """Make for the linked list"""
61         return 1
62
63     def __contains__(self, item):
64         """Make for the linked list"""
65         return self.name == item
66
67     def __getitem__(self, item):
68         """Make for the linked list"""
69         return self
70
71     def __setitem__(self, item, value):
72         """Make for the linked list"""
73         return self
74
75     def __delitem__(self, item):
76         """Make for the linked list"""
77         return self
78
79     def __len__(self):
80         """Make for the linked list"""
81         return 1
82
83     def __contains__(self, item):
84         """Make for the linked list"""
85         return self.name == item
86
87     def __getitem__(self, item):
88         """Make for the linked list"""
89         return self
90
91     def __setitem__(self, item, value):
92         """Make for the linked list"""
93         return self
94
95     def __delitem__(self, item):
96         """Make for the linked list"""
97         return self
98
99     def __len__(self):
100         """Make for the linked list"""
101         return 1
102
103     def __contains__(self, item):
104         """Make for the linked list"""
105         return self.name == item
106
107     def __getitem__(self, item):
108         """Make for the linked list"""
109         return self
110
111     def __setitem__(self, item, value):
112         """Make for the linked list"""
113         return self
114
115     def __delitem__(self, item):
116         """Make for the linked list"""
117         return self
118
119     def __len__(self):
120         """Make for the linked list"""
121         return 1
122
123     def __contains__(self, item):
124         """Make for the linked list"""
125         return self.name == item
126
127     def __getitem__(self, item):
128         """Make for the linked list"""
129         return self
130
131     def __setitem__(self, item, value):
132         """Make for the linked list"""
133         return self
134
135     def __delitem__(self, item):
136         """Make for the linked list"""
137         return self
138
139     def __len__(self):
140         """Make for the linked list"""
141         return 1
142
143     def __contains__(self, item):
144         """Make for the linked list"""
145         return self.name == item
146
147     def __getitem__(self, item):
148         """Make for the linked list"""
149         return self
150
151     def __setitem__(self, item, value):
152         """Make for the linked list"""
153         return self
154
155     def __delitem__(self, item):
156         """Make for the linked list"""
157         return self
158
159     def __len__(self):
160         """Make for the linked list"""
161         return 1
162
163     def __contains__(self, item):
164         """Make for the linked list"""
165         return self.name == item
166
167     def __getitem__(self, item):
168         """Make for the linked list"""
169         return self
170
171     def __setitem__(self, item, value):
172         """Make for the linked list"""
173         return self
174
175     def __delitem__(self, item):
176         """Make for the linked list"""
177         return self
178
179     def __len__(self):
180         """Make for the linked list"""
181         return 1
182
183     def __contains__(self, item):
184         """Make for the linked list"""
185         return self.name == item
186
187     def __getitem__(self, item):
188         """Make for the linked list"""
189         return self
190
191     def __setitem__(self, item, value):
192         """Make for the linked list"""
193         return self
194
195     def __delitem__(self, item):
196         """Make for the linked list"""
197         return self
198
199     def __len__(self):
200         """Make for the linked list"""
201         return 1
202
203     def __contains__(self, item):
204         """Make for the linked list"""
205         return self.name == item
206
207     def __getitem__(self, item):
208         """Make for the linked list"""
209         return self
210
211     def __setitem__(self, item, value):
212         """Make for the linked list"""
213         return self
214
215     def __delitem__(self, item):
216         """Make for the linked list"""
217         return self
218
219     def __len__(self):
220         """Make for the linked list"""
221         return 1
222
223     def __contains__(self, item):
224         """Make for the linked list"""
225         return self.name == item
226
227     def __getitem__(self, item):
228         """Make for the linked list"""
229         return self
230
231     def __setitem__(self, item, value):
232         """Make for the linked list"""
233         return self
234
235     def __delitem__(self, item):
236         """Make for the linked list"""
237         return self
238
239     def __len__(self):
240         """Make for the linked list"""
241         return 1
242
243     def __contains__(self, item):
244         """Make for the linked list"""
245         return self.name == item
246
247     def __getitem__(self, item):
248         """Make for the linked list"""
249         return self
250
251     def __setitem__(self, item, value):
252         """Make for the linked list"""
253         return self
254
255     def __delitem__(self, item):
256         """Make for the linked list"""
257         return self
258
259     def __len__(self):
260         """Make for the linked list"""
261         return 1
262
263     def __contains__(self, item):
264         """Make for the linked list"""
265         return self.name == item
266
267     def __getitem__(self, item):
268         """Make for the linked list"""
269         return self
270
271     def __setitem__(self, item, value):
272         """Make for the linked list"""
273         return self
274
275     def __delitem__(self, item):
276         """Make for the linked list"""
277         return self
278
279     def __len__(self):
280         """Make for the linked list"""
281         return 1
282
283     def __contains__(self, item):
284         """Make for the linked list"""
285         return self.name == item
286
287     def __getitem__(self, item):
288         """Make for the linked list"""
289         return self
290
291     def __setitem__(self, item, value):
292         """Make for the linked list"""
293         return self
294
295     def __delitem__(self, item):
296         """Make for the linked list"""
297         return self
298
299     def __len__(self):
300         """Make for the linked list"""
301         return 1
302
303     def __contains__(self, item):
304         """Make for the linked list"""
305         return self.name == item
306
307     def __getitem__(self, item):
308         """Make for the linked list"""
309         return self
310
311     def __setitem__(self, item, value):
312         """Make for the linked list"""
313         return self
314
315     def __delitem__(self, item):
316         """Make for the linked list"""
317         return self
318
319     def __len__(self):
320         """Make for the linked list"""
321         return 1
322
323     def __contains__(self, item):
324         """Make for the linked list"""
325         return self.name == item
326
327     def __getitem__(self, item):
328         """Make for the linked list"""
329         return self
330
331     def __setitem__(self, item, value):
332         """Make for the linked list"""
333         return self
334
335     def __delitem__(self, item):
336         """Make for the linked list"""
337         return self
338
339     def __len__(self):
340         """Make for the linked list"""
341         return 1
342
343     def __contains__(self, item):
344         """Make for the linked list"""
345         return self.name == item
346
347     def __getitem__(self, item):
348         """Make for the linked list"""
349         return self
350
351     def __setitem__(self, item, value):
352         """Make for the linked list"""
353         return self
354
355     def __delitem__(self, item):
356         """Make for the linked list"""
357         return self
358
359     def __len__(self):
360         """Make for the linked list"""
361         return 1
362
363     def __contains__(self, item):
364         """Make for the linked list"""
365         return self.name == item
366
367     def __getitem__(self, item):
368         """Make for the linked list"""
369         return self
370
371     def __setitem__(self, item, value):
372         """Make for the linked list"""
373         return self
374
375     def __delitem__(self, item):
376         """Make for the linked list"""
377         return self
378
379     def __len__(self):
380         """Make for the linked list"""
381         return 1
382
383     def __contains__(self, item):
384         """Make for the linked list"""
385         return self.name == item
386
387     def __getitem__(self, item):
388         """Make for the linked list"""
389         return self
390
391     def __setitem__(self, item, value):
392         """Make for the linked list"""
393         return self
394
395     def __delitem__(self, item):
396         """Make for the linked list"""
397         return self
398
399     def __len__(self):
400         """Make for the linked list"""
401         return 1
402
403     def __contains__(self, item):
404         """Make for the linked list"""
405         return self.name == item
406
407     def __getitem__(self, item):
408         """Make for the linked list"""
409         return self
410
411     def __setitem__(self, item, value):
412         """Make for the linked list"""
413         return self
414
415     def __delitem__(self, item):
416         """Make for the linked list"""
417         return self
418
419     def __len__(self):
420         """Make for the linked list"""
421         return 1
422
423     def __contains__(self, item):
424         """Make for the linked list"""
425         return self.name == item
426
427     def __getitem__(self, item):
428         """Make for the linked list"""
429         return self
430
431     def __setitem__(self, item, value):
432         """Make for the linked list"""
433         return self
434
435     def
```





- In an array, adding at the end is fast, but inserting in the middle is slow because elements must shift.
- In a linked list, insertion is fast because no shifting is needed.
- Searching takes the same time in both (you must check each element).
- Deleting in an array is slower due to shifting elements.
- Linked list is better for frequent insertions and deletions.

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.

Prompt:

Write a Python program for a library book request system. First, make a normal queue where requests are handled in the order they come. Then, make another version where faculty requests are given first priority over student requests. Include functions to add a request and remove a request.

Write a Python program for a library book request system.
First, make a normal queue where requests are handled in the order they come.
Then, make another version where faculty requests are given first priority over student requests.
Include functions to add a request and remove a request.

Code:

```
File Edit Selection ... Q AI Assistant Coding
buggy > % NormalQueue > remove_request
1 from collections import deque
2
3 # ===== NORMAL QUEUE IMPLEMENTATION =====
4 class NormalQueue:
5     """represents a book request"""
6     def __init__(self, request_id, requester_name, book_title):
7         self.request_id = request_id
8         self.requester_name = requester_name
9         self.book_title = book_title
10
11     def __str__(self):
12         return f"ID: {self.request_id}, Requester: {self.requester_name}, Book: {self.book_title}"
13
14
15 class NormalQueue:
16     """Simple FIFO queue for book requests"""
17     def __init__(self):
18         self.queue = deque()
19
20     def add_request(self, request):
21         """add a request to the queue"""
22         self.queue.append(request)
23         print("Request added: (request)")
24
25     def remove_request(self):
26         """remove and return the first request"""
27         if not self.queue:
28             print("Queue is empty!")
29             return None
30         request = self.queue.popleft()
31         print("Request processed: (request)")
32         return request
33
34     def display_queue(self):
35         """display all requests in queue"""
36         if not self.queue:
37             print("Queue is empty!")
38             return
39         print("===== Normal Queue =====")
40         for i, req in enumerate(self.queue, 1):
41             print(f"({i}). {req}")
42
43     def size(self):
44         return len(self.queue)
45
46 # ===== PRIORITY QUEUE IMPLEMENTATION =====
47 class PriorityBookRequest:
48     """book request with priority (Faculty > Student)"""
49     def __init__(self, request_id, requester_name, book_title, user_type):
50         self.request_id = request_id
51         self.requester_name = requester_name
52         self.book_title = book_title
53         self.user_type = user_type # "Faculty" or "Student"
54
55     def __str__(self):
56         return f"ID: {self.request_id}, Requester: {self.requester_name} ({self.user_type}), Book: {self.book_title}"
57
58     def get_priority(self):
59         """return priority (lower number = higher priority)"""
60         if self.user_type.lower() == "faculty":
61             return 1
62         else:
63             return 2
```

```
File Edit Selection ... Q AI Assistant Coding
buggy > % NormalQueue > remove_request
64
65 class PriorityBookRequest:
66     """book request with priority (Faculty > Student)"""
67     def __init__(self, request_id, requester_name, book_title, user_type):
68         self.request_id = request_id
69         self.requester_name = requester_name
70         self.book_title = book_title
71         self.user_type = user_type # "Faculty" or "Student"
72
73     def __str__(self):
74         return f"ID: {self.request_id}, Requester: {self.requester_name} ({self.user_type}), Book: {self.book_title}"
75
76     def get_priority(self):
77         """return priority (lower number = higher priority)"""
78         if self.user_type.lower() == "faculty":
79             return 1
80         else:
81             return 2
82
83     def add_request(self, request):
84         """add a request with priority sorting"""
85         self.queue.append(request)
86         self.queue.sort(key=lambda x: x.get_priority())
87         print("Request added: (request)")
88
89     def remove_request(self):
90         """remove and return the highest priority request"""
91         if not self.queue:
92             print("Queue is empty!")
93             return None
94         request = self.queue.pop(0)
95         print("Request processed: (request)")
96         return request
97
98     def display_queue(self):
99         """display all requests in priority order"""
100         if not self.queue:
101             print("Queue is empty!")
102             return
103         print("===== Priority Queue =====")
104         for i, req in enumerate(self.queue, 1):
105             print(f"({i}). {req}")
106
107     def size(self):
108         return len(self.queue)
109
110 # ===== DEMO PROGRAM =====
111 if __name__ == "__main__":
112     print("===== LIBRARY BOOK REQUEST SYSTEM =====")
113     # Create Normal Queue (FIFO)
114     normal_q = NormalQueue()
115
116     # Add requests
117     normal_q.add_request(BookRequest(1, "Alice", "Python Programming"))
118     normal_q.add_request(BookRequest(2, "Bob", "Data Science"))
119     normal_q.add_request(BookRequest(3, "Charlie", "Web Development"))
120     normal_q.add_request(BookRequest(4, "Diana", "Machine Learning"))
121
122     normal_q.display_queue()
123     print(f"Queue size: {normal_q.size()}")
124
125     # Process requests
126     print("===== Processing Requests (Normal Queue) =====")
127     normal_q.remove_request()
128     normal_q.remove_request()
129     normal_q.remove_request()
130
131     normal_q.display_queue()
132     print(f"Queue size: {normal_q.size()}")
133
134 # ===== Priority Queue Demo =====
135 if __name__ == "__main__":
136     print("===== LIBRARY BOOK REQUEST SYSTEM =====")
137     # Create Priority Queue
138     priority_q = PriorityBookRequest()
139
140     # Add requests
141     priority_q.add_request(BookRequest(1, "Alice", "Python Programming", "Student"))
142     priority_q.add_request(BookRequest(2, "Bob", "Data Science", "Faculty"))
143     priority_q.add_request(BookRequest(3, "Charlie", "Web Development", "Student"))
144     priority_q.add_request(BookRequest(4, "Diana", "Machine Learning", "Faculty"))
145
146     priority_q.display_queue()
147     print(f"Queue size: {priority_q.size()}")
148
149     # Process requests
150     print("===== Processing Requests (Priority Queue) =====")
151     priority_q.remove_request()
152     priority_q.remove_request()
153     priority_q.remove_request()
154     priority_q.remove_request()
155
156     priority_q.display_queue()
157     print(f"Queue size: {priority_q.size()}")
```

```

101 class PriorityQueue:
102     def __init__(self):
103         self.queue = []
104         return
105     def display_queue(self):
106         print("Priority Queue ---")
107         for i, req in enumerate(self.queue, 1):
108             print(f"{i}. {req}")
109     def size(self):
110         return len(self.queue)
111
112 # ===== DEMO PROGRAM =====
113 if __name__ == "__main__":
114     print("*****")
115     print("LIBRARY BOOK REQUEST SYSTEM")
116     print("*****")
117
118     # ===== Normal Queue Demo =====
119     print("\n--- NORMAL QUEUE (FIFO) ---")
120     normal_q = NormalQueue()
121
122     # Add requests
123     normal_q.add_request(BookRequest(1, "Alice", "Python Programming"))
124     normal_q.add_request(BookRequest(2, "Bob", "Data Science"))
125     normal_q.add_request(BookRequest(3, "Charlie", "Web Development"))
126     normal_q.add_request(BookRequest(4, "Diana", "Machine Learning"))
127
128     normal_q.display_queue()
129     print(f"Queue size: {normal_q.size()}")
130
131     # Process requests
132     print("\n--- Processing Requests (Normal Queue) ---")
133     normal_q.remove_request()
134     normal_q.remove_request()
135
136     normal_q.display_queue()
137     print(f"Queue size: {normal_q.size()}")
138
139     # ===== Priority Queue Demo =====
140     print("\n--- PRIORITY QUEUE (Faculty First) ---")
141     priority_q = PriorityQueue()
142
143     # Add requests ( mixes faculty and student )
144     priority_q.add_request(PriorityBookRequest(1, "Alice", "Python Programming", "Student"))
145     priority_q.add_request(PriorityBookRequest(2, "Prof. Bob", "Data Science", "Faculty"))
146     priority_q.add_request(PriorityBookRequest(3, "Charlie", "Web Development", "Student"))
147     priority_q.add_request(PriorityBookRequest(4, "Prof. Diana", "Machine Learning", "Faculty"))
148     priority_q.add_request(PriorityBookRequest(5, "Eve", "Databases", "Student"))
149
150     priority_q.display_queue()
151     print(f"Queue size: {priority_q.size()}")
152
153     # Process requests
154     print("\n--- Processing Requests (Priority Queue) ---")
155     priority_q.remove_request()
156     priority_q.remove_request()
157     priority_q.remove_request()
158
159     priority_q.display_queue()
160     print(f"Queue size: {priority_q.size()}")
161

```

Output:

```

PS C:\AI Assistant Coding> & C:\Users\edula\AppData\Local\Microsoft\WindowsApps\python3.11.exe "c:/AI Assistant Coding/bug.py"

=== Priority Queue ===
1. ID: 3, Requester: Charlie (Student), Book: Web Development
2. ID: 5, Requester: Eve (Student), Book: Databases
Queue size: 2

=====
PS C:\AI Assistant Coding>

```

Explanation:

- Queue (FIFO) → First request comes, first served.(If a student requests first, they get the book first.)
- Priority Queue → Faculty requests are served before students, even if they come later.
- enqueue() → Adds a request to the system.

- `dequeue()` → Removes and processes the next request.

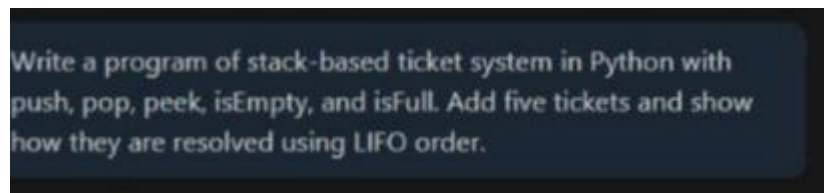
Task 3: Emergency Help Desk (Stack Implementation)

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.

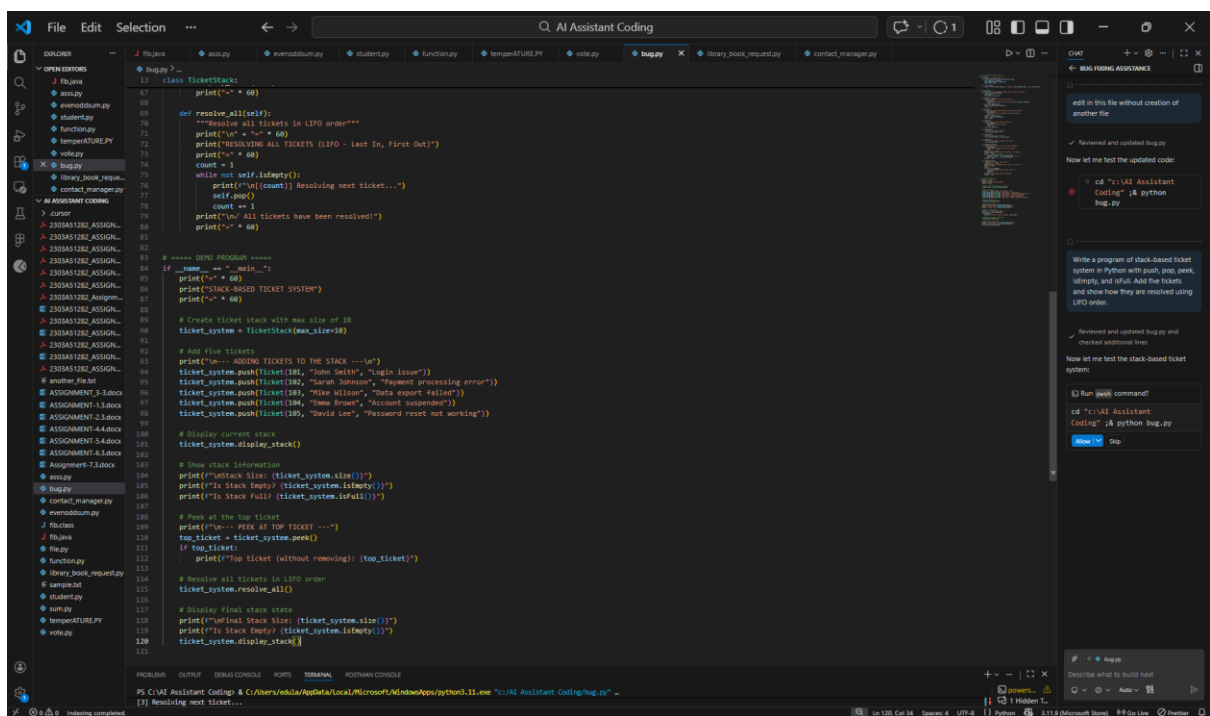
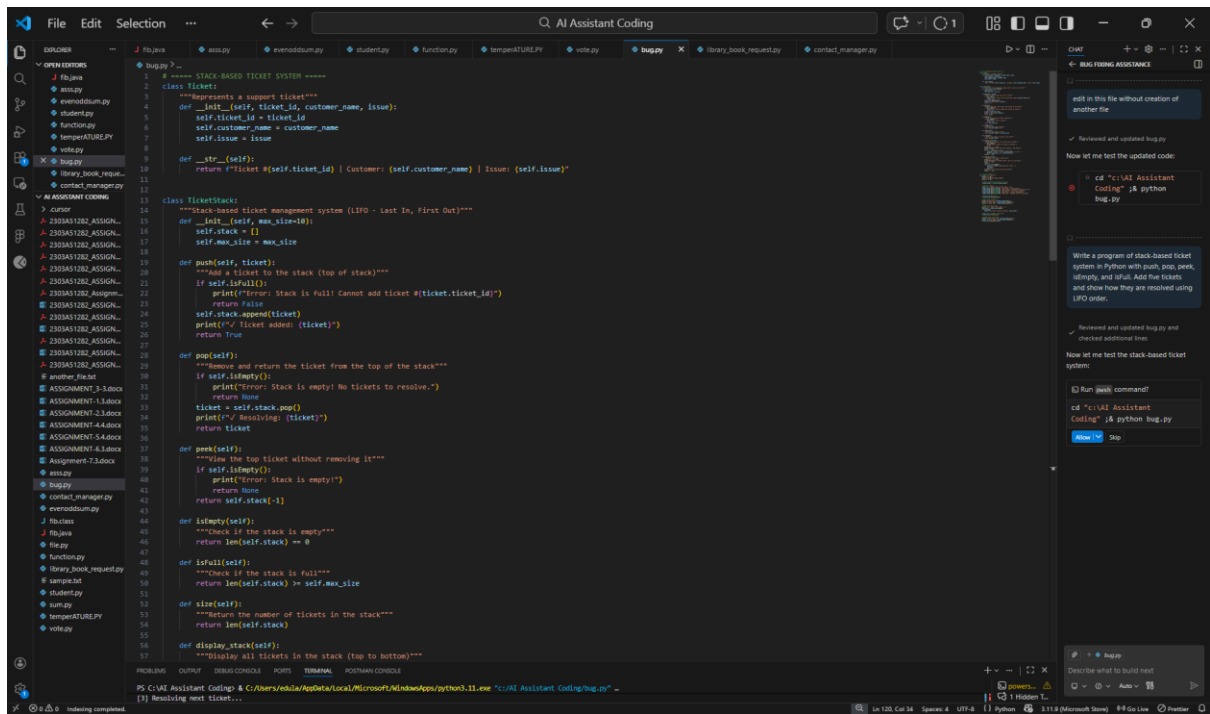
Prompt:

Write a program of stack-based ticket system in Python with `push`, `pop`, `peek`, `isEmpty`, and `isFull`. Add five tickets and show how they are resolved using LIFO order.

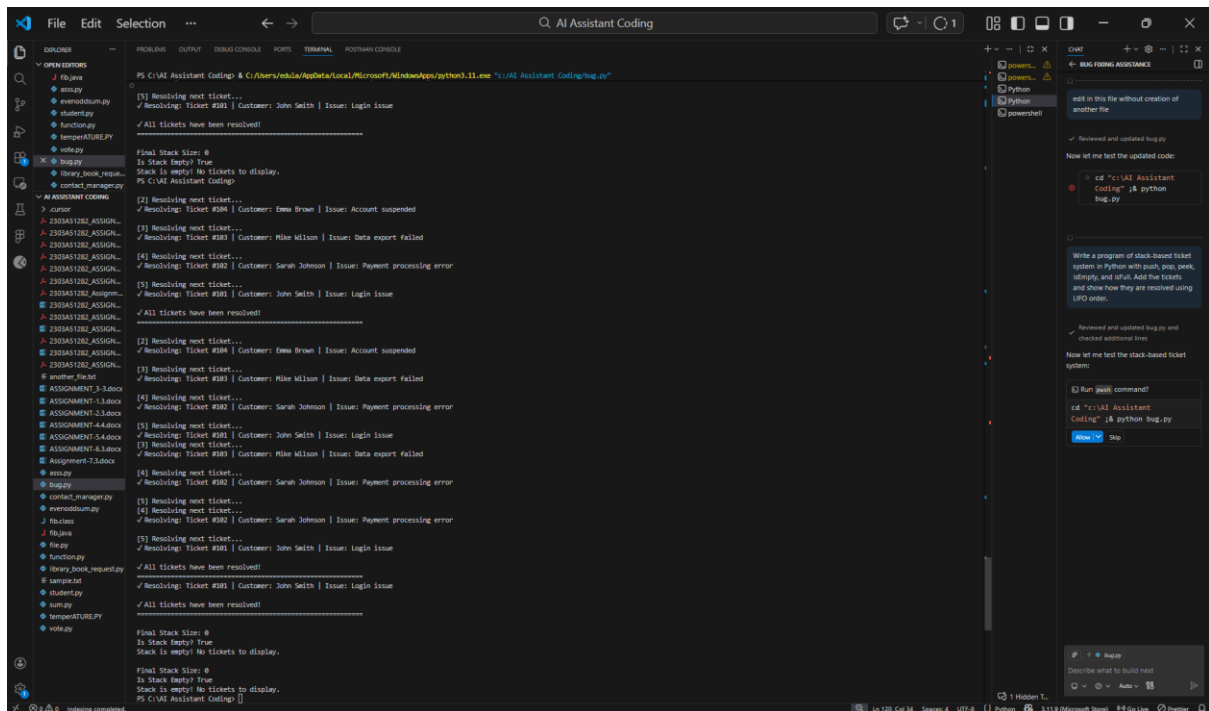


Write a program of stack-based ticket system in Python with `push`, `pop`, `peek`, `isEmpty`, and `isFull`. Add five tickets and show how they are resolved using LIFO order.

Code:



Output:



Explanation:

The program uses a stack to manage help desk tickets.

A stack works in last in, first solved order.

When a new ticket is raised, it is added to the top.

When solving a ticket, the most recent one is handled first.

The program can also check if there are no tickets left or if the stack is full.

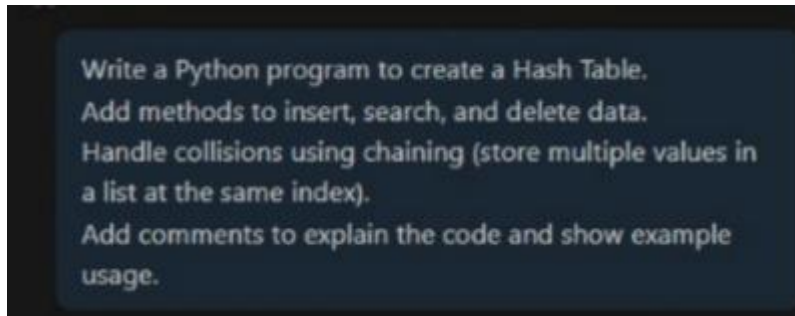
Task 4: Hash Table

Objective

To implement a Hash Table and understand collision handling.

Prompt:

Write a Python program to create a Hash Table.
Add methods to insert, search, and delete data.
Handle collisions using chaining (store multiple values in a list at the same index).
Add comments to explain the code and show example usage.



Code:

```
File Edit Selection ... Q AI Assistant Coding
# Create Hash Table with Chaining Collision Handling
1 # Create key-value pair class for storing data
2 class KeyValuePair:
3     """Represents a key-value pair in the hash table"""
4     def __init__(self, key, value):
5         self.key = key
6         self.value = value
7
8     def __str__(self):
9         return f"({self.key} | {self.value})"
10
11 class HashTable:
12     """Hash Table Implementation using chaining to handle collisions.
13     Chaining stores multiple key-value pairs that hash to the same index in a linked list.
14     """
15     def __init__(self, size=10):
16         """Initialize the hash table
17         Args:
18             size (int): Number of buckets in the hash table
19         """
20         self.size = size
21         # Create an array of empty lists (chains) for collision handling
22         self.table = [[] for _ in range(size)]
23         self.total_items = 0
24
25     def _hash_function(self, key):
26         """Hash function to determine the index for a given key
27         Uses simple modulo operation: hash(key) % table_size
28         Args:
29             key: The key to hash
30         Returns:
31             int: Index in the hash table
32         """
33         # Convert key to string and sum ASCII values for different key types
34         key_str = str(key)
35         hash_value = sum(ord(char) for char in key_str)
36         return hash_value % self.size
37
38     def insert(self, key, value):
39         """Insert a key-value pair into the hash table
40         If key already exists, update its value
41         If collision occurs, add to the chain (list) at that index
42         Args:
43             key: The key to insert
44             value: The value associated with the key
45         Returns:
46             bool: True if insertion successful, False otherwise
47         """
48         # Get the hash index
49         index = self._hash_function(key)
50         chain = self.table[index]
51
52         # Check if key already exists in the chain and update if found
53         for i, pair in enumerate(chain):
54             if pair.key == key:
```

```
class HashTable:
    def __init__(self, size):
        self.size = size
        self.table = [[] for _ in range(self.size)]

    def insert(self, key, value):
        # Hash the key to find the bucket
        index = self._hash(key)
        # Check if the key already exists in the bucket
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                # Key exists, update the value
                pair[1] = value
                return
        # Key not found, add new key-value pair to the bucket
        self.table[index].append((key, value))

    def search(self, key):
        # Hash the key to find the bucket
        index = self._hash(key)
        # Search for the key in the bucket
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                return pair[1]
        # Key not found
        return None

    def delete(self, key):
        # Hash the key to find the bucket
        index = self._hash(key)
        # Search for the key in the bucket
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                # Key exists, delete it
                del self.table[index][i]
                return
        # Key not found
        return None

    def _hash(self, key):
        # Simple hash function
        return hash(key) % self.size
```

```
def main():
    # Create a hash table with 10 buckets
    hash_table = HashTable(10)

    # Insert some data
    hash_table.insert("name", "Alice")
    hash_table.insert("age", 30)
    hash_table.insert("email", "alice@example.com")
    hash_table.insert("phone", "123-456-7890")
    hash_table.insert("country", "USA")
    hash_table.insert("address", "123 Main St")

    # Display the hash table
    hash_table.display()

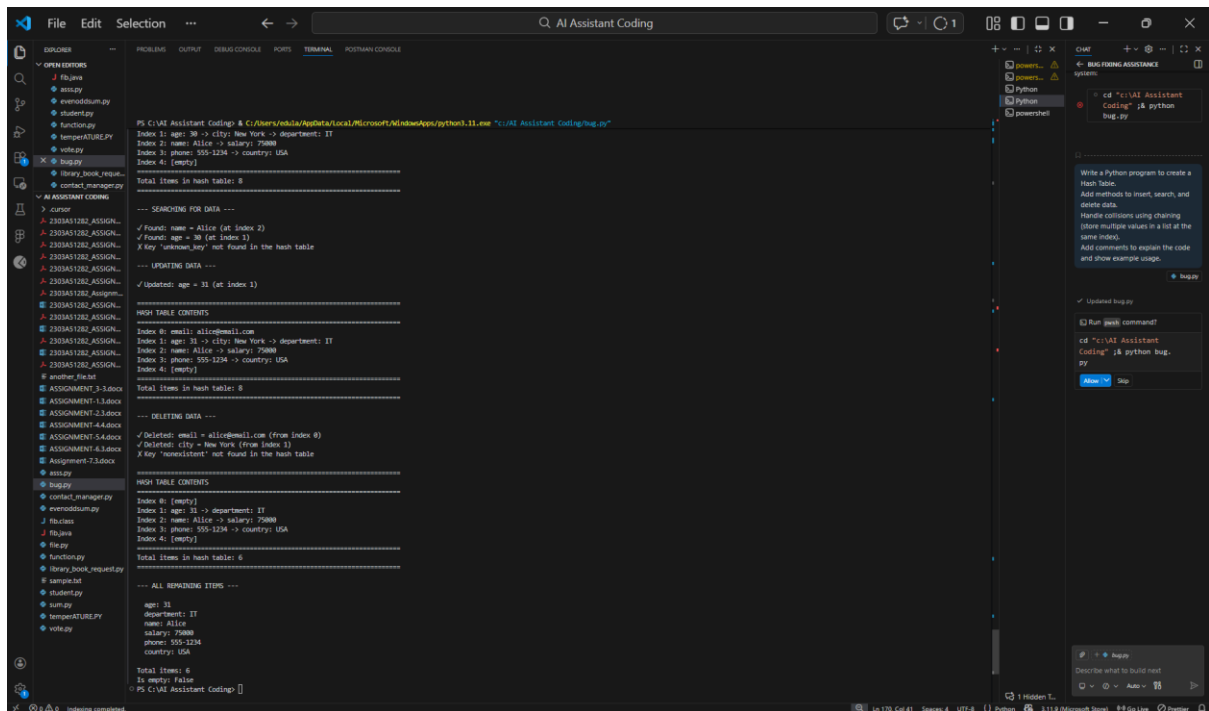
    # Search for data
    print("Searching for data...")
    hash_table.search("name")
    hash_table.search("age")
    hash_table.search("unknown_key") # Key that doesn't exist

    # Delete data
    print("Deleting data...")
    hash_table.delete("age")
    hash_table.delete("name")
    hash_table.delete("phone")
    hash_table.delete("nonexistent") # Try to delete non-existent key

    # Display the hash table after deletions
    hash_table.display()

    # Get all items
    print("Getting all items...")
    all_items = hash_table.get_all_items()
    for item in all_items:
        print(item)
```

Output:



```
PS C:\AI Assistant Coding\8 C:\Users\eduar\AppData\Local\Microsoft\WindowsApps\python11.exe "C:\AI Assistant Coding\bug.py"
Index 1: age: 30 -> city: New York -> department: IT
Index 2: name: Alice -> salary: 7000
Index 3: phone: 555-1234 -> country: USA
Index 4: [empty]
Total items in hash table: 8

--- SEARCHING FOR DATA ---
✓ Found: name = Alice (at index 2)
✓ Found: age = 30 (at index 1)
✗ Key 'unknown_key' not found in the hash table
--- UPDATING DATA ---
✓ Updated: age = 31 (at index 1)

HIGH TABLE CONTENTS
Index 0: email: alice@gmail.com
Index 1: age: 31 -> city: New York -> department: IT
Index 2: name: Alice -> salary: 7000
Index 3: phone: 555-1234 -> country: USA
Index 4: [empty]
Total items in hash table: 8

--- DELETING DATA ---
✓ Deleted: email = alice@gmail.com (from index 0)
✓ Deleted: city = New York (from index 1)
✗ Key 'nonexistent' not found in the hash table

HIGH TABLE CONTENTS
Index 0: [empty]
Index 1: age: 31 -> department: IT
Index 2: name: Alice -> salary: 7000
Index 3: phone: 555-1234 -> country: USA
Index 4: [empty]
Total items in hash table: 6

--- ALL REPAIRING ITEMS ---
age: 31
dept: IT
name: Alice
salary: 7000
phone: 555-1234
country: USA
Total items: 6
Is repair: false
PS C:\AI Assistant Coding\
```

Explanation:

- A Hash Table stores data using a key and value.
- A hash function decides where to store the data.
- Sometimes two keys go to the same place. This is called a collision.
- To solve collisions, we use chaining, meaning we store multiple items in a list at the same index.
- The program should allow adding, finding, and removing data correctly.

Task 5: Real-Time Application Challenge Scenario

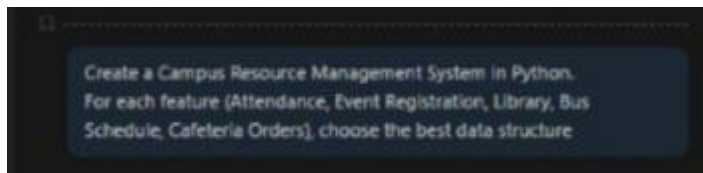
Design a Campus Resource Management System with the following features:

- Student Attendance Tracking

- Event Registration System
- Library Book Borrowing
- Bus Scheduling System
- Cafeteria Order Queue

Prompt:

Create a Campus Resource Management System in Python. For each feature (Attendance, Event Registration, Library, Bus Schedule, Cafeteria Orders), choose the best data structure



Code:

```

1  """
2  Campus Resource Management System
3  Features and chosen data structures:
4  - Attendance: set (O(1) add/check) to track present student IDs
5  - Event Registration: deque (FIFO queue) for fair registration processing
6  - Library: dict (hash table) mapping ISBN -> book record for fast lookup
7  - Bus Schedule: dict of route -> sorted list of departure times (list best sorted)
8  - Cafeteria Orders: heap (priority queue) to prioritize faculty over students while preserving arrival order
9
10 Run this file to see a small demo of each feature.
11 """
12
13 from collections import deque
14 import heapq
15 import random
16 from bisect import bisect
17 from datetime import datetime, timedelta
18
19 # ===== Attendance (set) =====
20 class Attendance:
21     """Track attendance using a set for O(1) add/remove/check."""
22     def __init__(self):
23         self.present = set()
24
25     def mark_present(self, student_id):
26         self.present.add(student_id)
27         print(f"Marked present: {student_id}")
28
29     def mark_absent(self, student_id):
30         self.present.discard(student_id)
31         print(f"Marked absent: {student_id}")
32
33     def is_present(self, student_id):
34         return student_id in self.present
35
36     def present_count(self):
37         return len(self.present)
38
39     def list_present(self):
40         return sorted(self.present)
41
42 # ===== Event Registration (FIFO queue) =====
43 class EventRegistration:
44     """Register attendees in arrival order using deque."""
45     def __init__(self):
46         self.queue = deque()
47
48     def add_registration(self, attendee_id, name):
49         self.queue.append((attendee_id, name))
50         print(f"Registered: {attendee_id} - {name}")
51
52     def process_registration(self):
53         if not self.queue:
54             print("No registrations to process.")
55             return None
56         attendee = self.queue.popleft()
57         print(f"Processed registration: {attendee[0]} - {attendee[1]}")
58         return attendee
59
60     def pending_count(self):
61         return len(self.queue)

```

```
class EventRegistration:
    def pending_count(self):
        return len(self.queue)

    def list_pending(self):
        return list(self.queue)

    # ----- Library (dict/hash table) -----
    class Library:
        """Simple library using a dict for O(1) lookups by ISBN."""
        def __init__(self):
            self.catalog = {}

        def add_book(self, isbn, title, author, copies=1):
            if isbn in self.catalog:
                self.catalog[isbn]['copies'] += copies
                print(f"Added {copies} more copy(ies) of {title} (ISBN: {isbn}).")
            else:
                self.catalog[isbn] = {
                    'title': title,
                    'author': author,
                    'copies': copies,
                    'borrowers': []
                }
                print(f"Added book: {title} (ISBN: {isbn}).")

        def search(self, isbn):
            return self.catalog.get(isbn)

        def borrow_book(self, isbn, user_id):
            book = self.catalog.get(isbn)
            if not book:
                print("Book not found.")
                return False
            if book['copies'] <= 0:
                print("No copies available.")
                return False
            book['copies'] -= 1
            book['borrowers'].append(user_id)
            print(f"{user_id} borrowed {book['title']}")
            return True

        def return_book(self, isbn, user_id):
            book = self.catalog.get(isbn)
            if not book:
                print("Book not found.")
                return False
            try:
                book['borrowers'].remove(user_id)
            except ValueError:
                print("This user did not borrow the book.")
                return False
            book['copies'] += 1
            print(f"{user_id} returned {book['title']}")
            return True

        def list_available(self):
            return [(isbn, info['title'], info['copies']) for isbn, info in self.catalog.items()]

    # ----- Bus Schedule (route -> sorted list of times) -----
```

```
    # ----- Bus Schedule (route -> sorted list of times) -----
    def __init__(self):
        self.route = []
        self.times = []

    def add_route(self, route, times):
        self.route.append(route)
        self.times.append(times)

    def get_route(self, route):
        return self.route[self.route.index(route)]

    def get_times(self, route):
        return self.times[self.route.index(route)]

    def is_route(self, route):
        return route in self.route

    def is_time(self, time):
        return time in self.times

    def get_route_index(self, route):
        return self.route.index(route)

    def get_time_index(self, time):
        return self.times.index(time)

    def get_route_and_time(self, route, time):
        return self.get_route_index(route), self.get_time_index(time)

    def get_route_and_time_index(self, route, time):
        return self.get_route_index(route), self.get_time_index(time)
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
Is empty: False
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
PS C:\AI Assistant Coding> & C:/Users/edula/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/campus_resource_management.py"
=====
Campus Resource Management Demo
=====
Marked present: 5001
Marked present: 5002
Marked present: 5010
Present list: ['5001', '5002', '5010']
Is 5002 present? True
Marked absent: 5002
Present count: 2
Registered: A001 - Alice
Registered: A002 - Bob
Registered: A003 - Charlie
Pending registrations: [('A001', 'Alice'), ('A002', 'Bob'), ('A003', 'Charlie')]
Processed registration: A001 - Alice
Pending count: 2
Added book: Clean Code (ISBN: 978-0135166307).
Added book: Fluent Python (ISBN: 978-1491958296).
5001 borrowed Clean Code
5003 borrowed Clean Code
No copies available.
Available books: [('978-0135166307', 'Clean Code', 0), ('978-1491958296', 'Fluent Python', 1)]
5001 returned Clean Code
Available books after return: [('978-0135166307', 'Clean Code', 1), ('978-1491958296', 'Fluent Python', 1)]
Added bus time for Route A: 2026-02-18 10:42:24.367227
Added bus time for Route A: 2026-02-18 10:57:24.367227
Added bus time for Route B: 2026-02-18 10:39:24.367227
Next Route A bus: 2026-02-18 10:42:24.367227
Route A schedule: [datetime.datetime(2026, 2, 18, 10, 42, 24, 367227), datetime.datetime(2026, 2, 18, 10, 57, 24, 367227)]
Order added: 0001 (Student)
Order added: 0002 (Faculty)
Order added: 0003 (Student)
Order added: 0004 (Faculty)
Pending cafeteria orders: 4
Serving order: 0002 (Faculty)
Serving order: 0004 (Faculty)
Pending orders after serving: 2

Demo complete.
PS C:\AI Assistant Coding>
```

Explanation:

Library Book Borrowing using a queue:

- The queue stores student names who request a book.
- When a student requests a book, we use `enqueue()` to add them to the queue.
- When a book becomes available, we use `dequeue()` to give it to the first student in line.
- This ensures fairness because the first requester gets the book first.

