

ASSIGNMENT-11.3

Name: E.Ramya

Ht.no: 2303A51282

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.

Code:

The screenshot shows a Windows desktop environment with a code editor window titled "AI Assistant Coding". The code editor displays a Python script for managing contacts. The sidebar contains AI-generated documentation for the ContactManager class, including details about its methods and their implementations.

```
File Edit Selection < > AI Assistant Coding D - + BROWSE API DOCUMENTATION
C:\Users\user\PycharmProjects\contact_manager> python contact_manager.py
[2023-06-01 14:45:45] INFO: Adding contact: Name: John Doe, Phone: 123-4567, Email: john.doe@example.com
[2023-06-01 14:45:46] INFO: All contacts: [Contact(id=1, name='John Doe', phone='123-4567', email='john.doe@example.com')]

Process finished with exit code 0
```

ContactManager Class Documentation

Class: ContactManager

Summary: A program Contact Manager that stores contacts in memory and provides basic operations like adding, displaying, and deleting contacts.

Attributes:

- self.contacts (list):** A list of Contact objects representing the contacts stored in memory.

Methods:

- add_contact(contact_id, contact):** Adds a contact to the list. If contact_id is None, it generates a new ID. Returns the added contact.
- display_all(self):** Prints all contacts in the list.
- display_by_name(self, name):** Prints contacts whose name matches the given name.
- delete_contact(contact_id):** Deletes a contact by ID. Returns the deleted contact if found, otherwise None.
- get_contact(contact_id):** Gets a contact by ID. Returns the contact if found, otherwise None.
- search_contacts(contact_id):** Searches for a contact by ID. Returns the contact if found, otherwise None.
- search_by_name(name):** Searches for contacts by name. Returns a list of contacts matching the name.
- update_contact(contact_id, contact):** Updates an existing contact by ID. Returns the updated contact if found, otherwise None.
- __init__(self):** Initializes the ContactManager object with an empty contacts list.

Operations:

- Add contact:** Adds a new contact to the list.
- Display all contacts:** Prints all contacts in the list.
- Display by name:** Prints contacts whose name matches the given name.
- Delete contact:** Deletes a contact by ID.
- Get contact:** Gets a contact by ID.
- Search contacts:** Searches for contacts by name.
- Update contact:** Updates an existing contact by ID.

Implementation Details:

- add_contact(contact_id, contact):** Adds a contact to the list. If contact_id is None, it generates a new ID. Returns the added contact.
- display_all(self):** Prints all contacts in the list.
- display_by_name(self, name):** Prints contacts whose name matches the given name.
- delete_contact(contact_id):** Deletes a contact by ID. Returns the deleted contact if found, otherwise None.
- get_contact(contact_id):** Gets a contact by ID. Returns the contact if found, otherwise None.
- search_contacts(contact_id):** Searches for a contact by ID. Returns the contact if found, otherwise None.
- search_by_name(name):** Searches for contacts by name. Returns a list of contacts matching the name.
- update_contact(contact_id, contact):** Updates an existing contact by ID. Returns the updated contact if found, otherwise None.

The screenshot shows a Python code editor with a file named `contact_manager.py` open. The code implements a contact manager using a linked list. It includes functions for adding contacts, searching by ID or name, displaying all contacts, deleting contacts, and printing statistics. The code is well-structured with comments explaining the logic.

```
#!/usr/bin/python

# A using linked Contact Manager
# python contact_manager.py
# manager_ll = ContactManager()

# Add contacts
# manager_ll.add_contact(contact_id="1", name="John Doe", "123 Main St", "johndoe@email.com")
# manager_ll.add_contact(contact_id="2", name="Jane Doe", "456 Elm St", "janedoe@email.com")
# manager_ll.add_contact(contact_id="3", name="Charlie Brown", "567 Oak St", "charliebrown@email.com")
# manager_ll.add_contact(contact_id="4", name="Snoopy", "789 Pine St", "snoopy@email.com")

# Display all
manager_ll.display_all()

# Search by ID
print("Search by ID ...")
contact = manager_ll.search_contact(2)
if contact:
    print(f"Found: {contact}")

# Search by Name
print("Search by Name ...")
print(manager_ll.search_name("Doe"))

# Delete contact
print("Delete contact ...")
manager_ll.delete_contact(1)
manager_ll.display_all()

# Search using linked list ...
manager_ll + ContactManager().__str__()

# Example usage
# manager_ll.add_contact(contact_id="1", name="John Doe", "123 Main St", "johndoe@email.com")
# manager_ll.add_contact(contact_id="2", name="Jane Doe", "456 Elm St", "janedoe@email.com")
# manager_ll.add_contact(contact_id="3", name="Charlie Brown", "567 Oak St", "charliebrown@email.com")
# manager_ll.add_contact(contact_id="4", name="Snoopy", "789 Pine St", "snoopy@email.com")

# Print stats
print("Total contacts (linked list):", len(manager_ll))
print("Total contacts (linked list):", len(manager_ll[1]))
```

Output:

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows files and folders including `fb.java`, `ass.py`, `evenmodsum.py`, `student.py`, `function.py`, `temperature.py`, `bug.py`, `contact_manager.py`, and `another_file.txt`.
- Terminal:** Displays command-line output from running `python3.11.exe "c:/AI Assistant Coding/bug.py"`. The output shows search results for contacts named Diana Prince, Charlie Brown, and Alice Johnson across various documents.
- Output:** Shows the message "Indexing completed."
- Search Bar:** Contains the text "diana@email.com". The status bar indicates "No matching results".
- Status Bar:** Shows "In 226 Col 1 Spaces 4 UTF-8" and "Python 3.11.9 Microsoft Store".
- Right Panel:** A "BUG FIXING ASSISTANT..." panel is open, showing a code editor with a placeholder Python script and instructions to write a Contact Manager program.
- Bottom Status Bar:** Includes icons for file operations like Open, Save, Find, and Go To, as well as "Hidden", "Go Live", and "Prefers".

Explanation:

- 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)

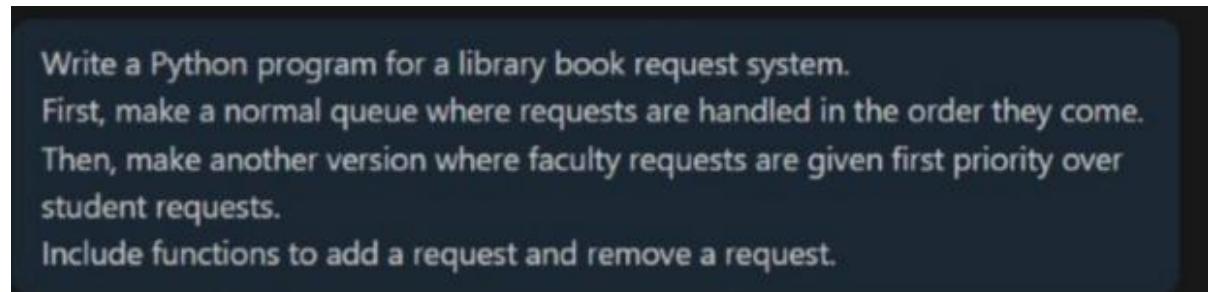
Scenario

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.



Code:

The screenshot shows a code editor window with the title "AI Assistant Coding". The main area displays the following Python code:

```
#!/usr/bin/python
# ***** NORMAL QUEUE IMPLEMENTATION *****
class BookRequest:
    """Book request class"""
    def __init__(self, request_id, requester_name, book_title):
        self.request_id = request_id
        self.requester_name = requester_name
        self.book_title = book_title

    def __str__(self):
        return f'ID: {self.request_id}, Requester: {self.requester_name}, Book: {self.book_title}'

class NormalQueue:
    """Simple FIFO queue for book requests"""
    def __init__(self):
        self.queue = deque()

    def add_request(self, request):
        """Add a request to the queue"""
        self.queue.append(request)
        print("Request added: (request)")

    def remove_request(self):
        """Remove and return the first request"""
        if not self.queue:
            print("Queue is empty")
        else:
            request = self.queue.popleft()
            print("Request processed: (request)")
        return request

    def display_queue(self):
        """Display all requests in queue"""
        if not self.queue:
            print("Queue is empty")
        else:
            for i, req in enumerate(self.queue, 1):
                print(f'{i}. {req}')

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

# ***** PRIORITY QUEUE IMPLEMENTATION *****
class PriorityBookRequest:
    """Book request with priority (Faculty > Student)"""
    def __init__(self, request_id, requester_name, book_title, user_type):
        self.request_id = request_id
        self.requester_name = requester_name
        self.book_title = book_title
        self.user_type = user_type # 'Faculty' or 'Student'

    def __str__(self):
        return f'ID: {self.request_id}, Requester: {self.requester_name} ({self.user_type}), Book: {self.book_title}'

    def get_priority(self):
        """Return priority (lower number = higher priority)"""
        if self.user_type.lower() == 'faculty':
            return 1
        else:
            return 2
```

The code editor interface includes a sidebar with file navigation, a status bar at the bottom, and two side panes on the right containing AI-generated code snippets and instructions.

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- Top Bar:** File, Edit, Selection, < >, AI Assistant Coding, & 1.
- Left Sidebar:** Explorer (Open Editors), Search, Find, AI ASSISTANT CODING, and a list of files including buggy.py, NormalQueue.py, evenoddsum.py, student.py, function.py, temperature.py, vote.py, library_book_request.py, contact_manager.py, and another_file.txt.
- Central Area:** A large code editor window displaying Python code for a priority queue system. The code includes functions for adding, removing, and displaying requests, as well as a main loop for processing library book requests. A status bar at the bottom indicates "Indexing completed".
- Right Side:** A sidebar titled "BUG FIXING..." with a list of items: "requests are given first priority over even odd sum", "include function to add a request and remove a request", "Created file", "Executed program", "To demonstrate", "Run push command!", "cd c:\AI Assistant Coding\# python library_bo_1K_request.py", "Allow Skip", and "edit in this file without creation of another file". Below this is a "REVIEWED AND UPDATED" section with "Run push command!", "cd c:\AI Assistant Coding\# python bug.py", "Allow Skip", and "Now let me test the updated code".

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.

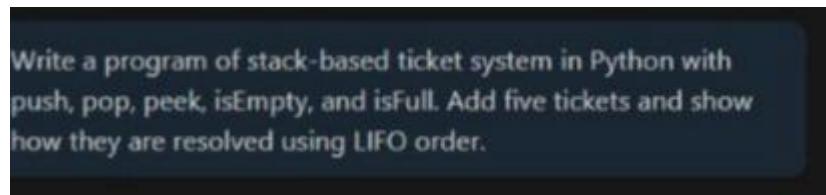
Task 3: Emergency Help Desk (Stack Implementation)

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.



Code:

The screenshot shows the Microsoft Visual Studio Code interface with a light-themed window. On the left, there is a file explorer sidebar showing various files and folders, including "bug.py" which is currently selected. The main code editor area contains the following Python code:

```
# ----- STACK-BASED TICKET SYSTEM -----
class Ticket:
    """Represents a support ticket"""
    def __init__(self, ticket_id, customer_name, issue):
        self.ticket_id = ticket_id
        self.customer_name = customer_name
        self.issue = issue

    def __str__(self):
        return f"Ticket # {self.ticket_id} | Customer: {self.customer_name} | Issue: {self.issue}"

class TicketStack:
    """Stack-based ticket management system (LIFO - Last In, First Out)"""
    def __init__(self, max_size=10):
        self.stack = []
        self.max_size = max_size

    def push(self, ticket):
        """Add a ticket to the stack (top of stack)"""
        if self.isFull():
            print("Error: Stack is full! Cannot add ticket # [ticket.ticket_id]")
            return None
        self.stack.append(ticket)
        print(f"Ticket added: {ticket}")
        return True

    def pop(self):
        """Remove and return the ticket from the top of the stack"""
        if self.isEmpty():
            print("Error: Stack is empty! No tickets to resolve.")
            return None
        ticket = self.stack.pop()
        print(f"Resolving: {ticket}")
        return ticket

    def peek(self):
        """View the top ticket without removing it"""
        if self.isEmpty():
            print("Error: Stack is empty!")
            return None
        return self.stack[-1]

    def isEmpty(self):
        """Check if the stack is empty"""
        return len(self.stack) == 0

    def isFull(self):
        """Check if the stack is full"""
        return len(self.stack) > self.max_size

    def size(self):
        """Return the number of tickets in the stack"""
        return len(self.stack)

    def display_stack(self):
        """Display all tickets in the stack (top to bottom)"""
        for ticket in reversed(self.stack):
            print(ticket)
```

At the bottom of the code editor, there is a terminal window showing the command "PS c:\AI Assistant Coding & C:/users/eduka/AppData/Local/Microsoft/WindowsApps/python3.11.exe <:/AI Assistant Coding/bug.py" and the message "(3) Resolving next ticket...". The status bar at the bottom indicates "Line 120, Col 34" and "Python 3.11.0 (Microsoft Store)".

```

File Edit Selection ...
File Explorer ...
OPEN EDITORS
J fbjava
asspy
evenoddum.py
student.py
function.py
temperaturePY
vote.py
library_book_request.py
bug.py
library_book_request...
contact_manager.py
AI ASSISTANT CODING ...
cursor
ASSIGNMENT_3.docx
ASSIGNMENT_3-3.docx
ASSIGNMENT_3-3-3.docx
ASSIGNMENT_4-4.docx
ASSIGNMENT_5-5.docx
ass
bug.py
library_book_request...
samplebat
student
sum.py
temperaturePY
vote.py
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
PS C:\VIAI Assistant Coding & C:/users/edu/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
[1] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #001 | Customer: John Smith | Issue: Login issue
✓ Resolving Ticket #001 | Customer: John Smith | Issue: Login issue
All tickets have been resolved!
Final Stack Size: 0
Is Stack Empty: True
Stack is empty! No tickets to display.
PS C:\VIAI Assistant Coding

```

Output:

```

File Edit Selection ...
File Explorer ...
OPEN EDITORS
J fbjava
asspy
evenoddum.py
student.py
function.py
temperaturePY
vote.py
library_book_request.py
bug.py
library_book_request...
contact_manager.py
AI ASSISTANT CODING ...
cursor
ASSIGNMENT_3.docx
ASSIGNMENT_3-3.docx
ASSIGNMENT_3-3-3.docx
ASSIGNMENT_4-4.docx
ASSIGNMENT_5-5.docx
ass
bug.py
library_book_request...
samplebat
student
sum.py
temperaturePY
vote.py
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE
PS C:\VIAI Assistant Coding & C:/users/edu/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/AI Assistant Coding/bug.py"
[1] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #001 | Customer: Emma Brown | Issue: Account suspended
✓ Resolving Ticket #001 | Customer: Emma Brown | Issue: Account suspended
[2] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #002 | Customer: Mike Wilson | Issue: Data export failed
✓ Resolving Ticket #002 | Customer: Mike Wilson | Issue: Data export failed
[3] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #003 | Customer: Sarah Johnson | Issue: Payment processing error
✓ Resolving Ticket #003 | Customer: Sarah Johnson | Issue: Payment processing error
[4] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #004 | Customer: John Smith | Issue: Login issue
✓ Resolving Ticket #004 | Customer: John Smith | Issue: Login issue
[5] Resolving next ticket...
Resolving next ticket...
Resolving Ticket #005 | Customer: John Smith | Issue: Login issue
✓ Resolving Ticket #005 | Customer: John Smith | Issue: Login issue
All tickets have been resolved!
Final Stack Size: 0
Is Stack Empty: True
Stack is empty! No tickets to display.
PS C:\VIAI Assistant Coding

```

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.

```
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 ...
File Explorer View Editor Insert Run Terminal Help

AI Assistant Coding

bug.py

14 class HashTable:
15     def get_all_items(self):
16         """Return all key-value pairs in the hash table"""
17         all_items = []
18         for item in self.table:
19             for pair in item:
20                 all_items.append(pair)
21         return all_items
22
23 # ----- SEND PROGRAM -----
24 if __name__ == "__main__":
25     print("-----")
26     print("HASH TABLE WITH CHAINING COLLISION HANDLING")
27     print("-----")
28
29 # Create a hash table with 3 buckets;
30 # each bucket has a maximum of 5 slots
31 hash_table = HashTable(3)
32
33 # ----- INSERT OPERATIONS -----
34 print("----- INSERTING DATA -----")
35 hash_table.insert("name", "John")
36 hash_table.insert("age", 30)
37 hash_table.insert("city", "New York")
38 hash_table.insert("email", "john@gmail.com")
39 hash_table.insert("phone", "555-1234")
40 hash_table.insert("name", "Alice") # This may collide with other keys
41 hash_table.insert("salary", 7500)
42 hash_table.insert("department", "IT")
43
44 # ASSIGNMENT-3-3.docx
45 # Display the hash table
46 hash_table.display()
47
48 # ----- SEARCH OPERATIONS -----
49 print("----- SEARCHING FOR DATA -----")
50 hash_table.search("name")
51 hash_table.search("age")
52 hash_table.get("name") # Key that doesn't exist
53
54 # ----- UPDATE OPERATION -----
55 print("----- UPDATING DATA -----")
56 hash_table.insert("age", 31) # Update existing key
57
58 # Display the hash table after update
59 hash_table.display()
60
61 # ----- DELETE OPERATIONS -----
62 print("----- DELETING DATA -----")
63 hash_table.delete("email")
64 hash_table.delete("name") # Try to delete non-existent key
65 hash_table.delete("nonexistent")
66
67 # Display the hash table after deletions
68 hash_table.display()
69
70 # ----- GET ALL ITEMS -----
71 print("----- ALL REMAINING ITEMS -----")
72 hash_table.get_all_items()
73 for item in all_items:
74     print(item)
75
76 print(f"Total items: {hash_table.get_size()}")
77 print(f"Is empty: {hash_table.is_empty()}")
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
15
```

- 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

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 # Features and chosen data structures:
2 # Attendance: set O(1) add/remove to track present student ID
3 # Event Registration: list (queue) to store attendees in arrival order
4 # Library: dict (hash table) mapping ISBN -> book record for fast lookup
5 # Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
6 # Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
7
8 Run this file to see a small demo of each feature.
9
10 from collections import deque
11 import heapq
12 import timeit
13 import itertools
14 from bisect import insort
15 from datetime import datetime, timedelta
16
17 # ----- Attendance (set) -----
18
19 class Attendance:
20     """Track attendance using a set for O(1) add/remove/check."""
21     def __init__(self):
22         self.present = set()
23
24     def mark_present(self, student_id):
25         self.present.add(student_id)
26         print("Marked present: (%s)" % student_id)
27
28     def mark_absent(self, student_id):
29         self.present.discard(student_id)
30         print("Marked absent: (%s)" % student_id)
31
32     def is_present(self, student_id):
33         return student_id in self.present
34
35     def present_count(self):
36         return len(self.present)
37
38     def list_present(self):
39         return sorted(self.present)
40
41 # ----- Event Registration (FIFO queue) -----
42
43 class EventRegistration:
44     """Register attendees in arrival order using a 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("Registered: (%s) - (%s)" % (attendee_id, name))
51
52     def process_registration(self):
53         if not self.queue:
54             print("No registrations to process.")
55             return
56
57         attendee = self.queue.popleft()
58         print("Processed registration: (%s) - (%s)" % (attendee[0], attendee[1]))
59
60     def pending_count(self):
61         return len(self.queue)
62
63
64 # ----- Library (dict/hash table) -----
65
66 class Library:
67     """Simple library using a dict for O(1) lookups by ISBN."""
68     def __init__(self):
69         self.catalog = {}
70
71     def add_book(self, isbn, title, author, copies=1):
72         if isbn in self.catalog:
73             self.catalog[isbn][copies] += copies
74             print("%d more copies added for (%s) (%s). Total: (%s) (%s)." % (copies, title, author, isbn, str(self.catalog[isbn][copies])))
75         else:
76             self.catalog[isbn] = {
77                 'title': title,
78                 'author': author,
79                 'copies': copies,
80                 'borrowers': []
81             }
82             print("Added book: (%s) (%s)." % (title, isbn))
83
84     def search(self, isbn):
85         return self.catalog.get(isbn)
86
87     def borrow_book(self, isbn, user_id):
88         book = self.catalog.get(isbn)
89         if not book:
90             print("Book not found.")
91             return False
92         if book['copies'] <= 0:
93             print("No copies available.")
94             return False
95         book['copies'] -= 1
96         book['borrowers'].append(user_id)
97         print("User (%s) borrowed (%s)." % (user_id, book['title']))
98
99     def return_book(self, isbn, user_id):
100        book = self.catalog.get(isbn)
101        if not book:
102            print("Book not found.")
103            return False
104        try:
105            book['borrowers'].remove(user_id)
106        except ValueError:
107            print("User did not borrow the book (%s)." % user_id)
108        return True
109
110    def list_available(self):
111        return [(isbn, info['title'], info['copies']) for isbn, info in self.catalog.items()]
112
113
114 # ----- Bus Schedule (route -> sorted list of times) -----
115
116
117
118
119

```

```

1 # Features and chosen data structures:
2 # Attendance: set O(1) add/remove to track present student ID
3 # Event Registration: list (queue) to store attendees in arrival order
4 # Library: dict (hash table) mapping ISBN -> book record for fast lookup
5 # Bus Schedule: dict of route -> sorted list of departure times (list kept sorted)
6 # Cafeteria Orders: heapq (priority queue) to prioritize faculty over students while preserving arrival order
7
8 Run this file to see a small demo of each feature.
9
10 from collections import deque
11 import heapq
12 import timeit
13 import itertools
14 from bisect import insort
15 from datetime import datetime, timedelta
16
17 # ----- Attendance (set) -----
18
19 class Attendance:
20     """Track attendance using a set for O(1) add/remove/check."""
21     def __init__(self):
22         self.present = set()
23
24     def mark_present(self, student_id):
25         self.present.add(student_id)
26         print("Marked present: (%s)" % student_id)
27
28     def mark_absent(self, student_id):
29         self.present.discard(student_id)
30         print("Marked absent: (%s)" % student_id)
31
32     def is_present(self, student_id):
33         return student_id in self.present
34
35     def present_count(self):
36         return len(self.present)
37
38     def list_present(self):
39         return sorted(self.present)
40
41 # ----- Event Registration (FIFO queue) -----
42
43 class EventRegistration:
44     """Register attendees in arrival order using a 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("Registered: (%s) - (%s)" % (attendee_id, name))
51
52     def process_registration(self):
53         if not self.queue:
54             print("No registrations to process.")
55             return
56
57         attendee = self.queue.popleft()
58         print("Processed registration: (%s) - (%s)" % (attendee[0], attendee[1]))
59
60     def pending_count(self):
61         return len(self.queue)
62
63
64 # ----- Library (dict/hash table) -----
65
66 class Library:
67     """Simple library using a dict for O(1) lookups by ISBN."""
68     def __init__(self):
69         self.catalog = {}
70
71     def add_book(self, isbn, title, author, copies=1):
72         if isbn in self.catalog:
73             self.catalog[isbn][copies] += copies
74             print("%d more copies added for (%s) (%s). Total: (%s) (%s)." % (copies, title, author, isbn, str(self.catalog[isbn][copies])))
75         else:
76             self.catalog[isbn] = {
77                 'title': title,
78                 'author': author,
79                 'copies': copies,
80                 'borrowers': []
81             }
82             print("Added book: (%s) (%s)." % (title, isbn))
83
84     def search(self, isbn):
85         return self.catalog.get(isbn)
86
87     def borrow_book(self, isbn, user_id):
88         book = self.catalog.get(isbn)
89         if not book:
90             print("Book not found.")
91             return False
92         if book['copies'] <= 0:
93             print("No copies available.")
94             return False
95         book['copies'] -= 1
96         book['borrowers'].append(user_id)
97         print("User (%s) borrowed (%s)." % (user_id, book['title']))
98
99     def return_book(self, isbn, user_id):
100        book = self.catalog.get(isbn)
101        if not book:
102            print("Book not found.")
103            return False
104        try:
105            book['borrowers'].remove(user_id)
106        except ValueError:
107            print("User did not borrow the book (%s)." % user_id)
108        return True
109
110    def list_available(self):
111        return [(isbn, info['title'], info['copies']) for isbn, info in self.catalog.items()]
112
113
114 # ----- Bus Schedule (route -> sorted list of times) -----
115
116
117
118
119

```

The screenshot shows a developer's workspace with the following tabs open:

- ass.py
- evenodumod.py
- student.py
- function.py
- temperature.py
- vote.py
- campus_resource_management.py
- bug.py
- library_book_request.py
- contact_m
- chat

The code editor displays a file named `campus_resource_management.py` which includes:

- A class `CampusResourceManagement` with methods for adding resources, marking presence, and printing resource details.
- A class `Attendance` with methods for marking present and counting attendance.
- A class `Event` with methods for adding registrations and processing registrations.
- A library `lib` containing functions for adding books, borrowing books, returning books, and listing available books.
- A class `CafeteriaOrders` with methods for adding orders and serving them.

On the right side of the interface, there is a "BUG FIXING ASSISTANCE" panel with the message "It is not giving any output". Below it, a "REVIEWED" section lists:

- bug.py: I'll open > bug.py to verify its current contents and the `__main__` section.
- campus_resource_management.py: I'll fix > bug.py by replacing it with a standalone Campus Resource Management System so the module runs at module level.

At the bottom, there are buttons for "Delete bug.py", "Create", "Run [local command]", and "Allow Skip".

Output:

PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL POSTMAN CONSOLE

PS C:\AI Assistant Coding & C:/Users/edula/AppData/Local/Microsoft/Windows/Apps/python3.11.exe "c:/AI Assistant Coding/bug.py"
Is empty: False
PS C:\AI Assistant Coding & C:/Users/edula/AppData/Local/Microsoft/Windows/Apps/python3.11.exe "c:/AI Assistant Coding/bug.py"
PS C:\AI Assistant Coding & C:/Users/edula/AppData/Local/Microsoft/Windows/Apps/python3.11.exe "c:/AI Assistant Coding/campus_resource_management.py"

Campus Resource Management Demo
=====

Marked present: S001
Marked present: S002
Marked present: S010
Present list: ['S001', 'S002', 'S010']
Is S002 present? True
Marked absent: S002
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: 978-0135166387).
Added book: Fluent Python (ISBN: 978-1491958296).
S001 borrowed Clean Code
S003 borrowed Clean Code
No copies available.
Available books: [('978-0135166387', 'Clean Code', 0), ('978-1491958296', 'Fluent Python', 1)]
S001 returned Clean Code
Available books after return: [('978-0135166387', '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: 0081 (Student)
Order added: 0082 (Faculty)
Order added: 0083 (Student)
Order added: 0084 (Faculty)
Pending cafeteria orders: 4
Serving order: 0082 (Faculty)
Serving order: 0084 (Faculty)
Pending orders after serving: 2

Demo complete.

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.