

ASSIGNMENT 11.3

NAME:

HALLTICKET:

TASK1:

PROMPT

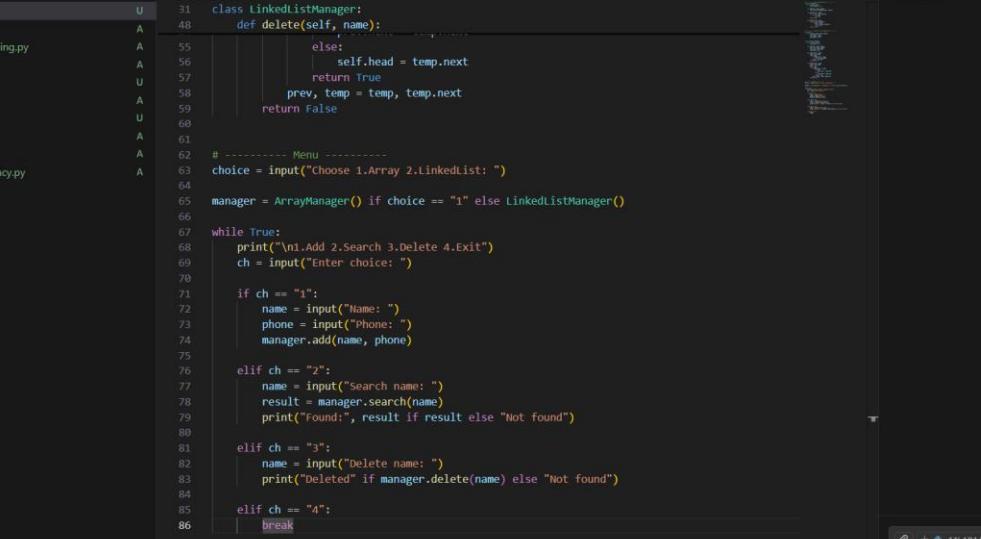
Create a Smart Contact Manager in Python using both Array (list) and Linked List. Implement operations to add, search, and delete contacts. Provide a simple menu to choose implementation and perform operations.

The screenshot shows a code editor interface with the following details:

- File Explorer (EXPLORER):** Shows a folder named "AIASSCODE" containing files: 11LAB1.PY, code.py, electricity_billing.py, lab1.py, lab2.py, multiples.py, n.py, sample.txt, student.py, and word_frequency.py.
- Code Editor:** The main pane displays Python code for a "Smart Contact Manager".

```
1 # ----- Array Implementation -----
2 class ArrayManager:
3     def __init__(self):
4         self.contacts = []
5
6     def add(self, name, phone):
7         self.contacts.append((name, phone))
8
9     def search(self, name):
10        for n, p in self.contacts:
11            if n == name:
12                return p
13        return None
14
15    def delete(self, name):
16        for c in self.contacts:
17            if c[0] == name:
18                self.contacts.remove(c)
19                return True
20        return False
21
22
23 # ----- Linked List Implementation -----
24 class Node:
25     def __init__(self, name, phone):
26         self.name = name
27         self.phone = phone
28         self.next = None
29
30
31 class LinkedListManager:
32     def __init__(self):
33         self.head = None
34
35     def add(self, name, phone):
36         new = Node(name, phone)
37         new.next = self.head
38         self.head = new
39
```
- Terminal:** Shows the command "Ln 11, Col 26" and other terminal-related information.
- Bottom Status Bar:** Shows "Spaces: 4", "UTF-8", "CRLF", "Python", "3.14.2", "Go Live", and the "CODEGPT" logo.

```
lab1.py A student.py A 11LAB1.PY U n.py U lab2.py U COMMIT_EDITMS D ... CHAT SESSIONS + 11LAB1.PY > ... 31 class LinkedListManager: 32 33     def search(self, name): 34         temp = self.head 35         while temp: 36             if temp.name == name: 37                 return temp.phone 38             temp = temp.next 39         return None 40 41     def delete(self, name): 42         temp = self.head 43         prev = None 44         while temp: 45             if temp.name == name: 46                 if prev: 47                     prev.next = temp.next 48                 else: 49                     self.head = temp.next 50             return True 51             prev, temp = temp, temp.next 52         return False 53 54 # ----- Menu ----- 55 choice = input("Choose 1.Array 2.LinkedList: ") 56 57 manager = ArrayManager() if choice == "1" else LinkedListManager() 58 59 while True: 60     print("\n1.Add 2.Search 3.Delete 4.Exit") 61     ch = input("Enter choice: ") 62 63     if ch == "1": 64         name = input("Name: ") 65         phone = input("Phone: ") 66         manager.add(name, phone) 67 68     elif ch == "2": 69         name = input("Search name: ") 70 71     elif ch == "3": 72         name = input("Delete name: ") 73 74     elif ch == "4": 75         break 76 77
```



```
File Edit Selection View Go Run Terminal Help ↺ ↻ aiaascode
EXPLORER AIASCODE 11LAB1.PY student.py 11LAB1.PY n.py lab2.py COMMIT_EDITMS ...
AIASCODE
11LAB1.PY U
code.py A
electricity_billing.py A
lab1.py A
lab2.py U
multiples.py A
n.py U
sample.txt A
student.py A
word_frequency.py A

11LAB1.PY > ...
31 class LinkedListManager:
32     def delete(self, name):
33         if self.head == None:
34             return False
35         else:
36             self.head = temp.next
37             return True
38         prev, temp = temp, temp.next
39         return False
40
41 # ----- Menu -----
42 choice = input("Choose 1.Array 2.LinkedList: ")
43
44 manager = ArrayManager() if choice == "1" else LinkedListManager()
45
46 while True:
47     print("\n1.Add 2.Search 3.Delete 4.Exit")
48     ch = input("Enter choice: ")
49
50     if ch == "1":
51         name = input("Name: ")
52         phone = input("Phone: ")
53         manager.add(name, phone)
54
55     elif ch == "2":
56         name = input("Search name: ")
57         result = manager.search(name)
58         print("Found:", result if result else "Not found")
59
60     elif ch == "3":
61         name = input("Delete name: ")
62         print("Deleted" if manager.delete(name) else "Not found")
63
64     elif ch == "4":
65         break
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
```

OUTPUT:

```

-3.14-64/python.exe c:/Users/Administrator/Desktop/aiasscode/11LAB1.PY
Choose 1.Array 2.LinkedList: 1
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 1
Name: hitresh
Phone: 234
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 1
Name: karthik
Phone: 846
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 2
Search name: karthik
Found: 846
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 2
Search name: karthik
Found: 846
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 2
Search name: karthik
Found: 846
1.Add 2.Search 3.Delete 4.Exit
Enter choice: 2
Search name: karthik
Found: 846

```

Explanation: The array-based contact manager stores contacts in a list, allowing simple insertion but requiring element shifting during deletion. Searching in the array implementation takes linear time as each contact is checked sequentially. The linked list implementation stores contacts using dynamically allocated nodes, allowing efficient insertion at the beginning. Deletion in a linked list is more efficient than arrays since no shifting of elements is required. Both implementations have linear time complexity for searching, but linked lists handle frequent updates better.

TASK2:

PROMPT:

Create a Library Book Request System in Python.

Implement a normal Queue (FIFO) and a Priority Queue where faculty requests have higher priority than student requests. Include enqueue and dequeue methods and a simple menu for testing.

```
File Edit Selection View Go Run Terminal Help <- > Q aiascode
EXPLORER ... lab1.py A student.py A 11LAB1.PY U 11LAB2.PY U n.py U lab2.py U D ...
AIASSCODE
11LAB1.PY
code.py
electricity_billing.py
lab1.py
lab2.py
LLAB2.PY
multiples.py
n.py
sample.txt
student.py
word_frequency.py

l11AB2.PY > -
1 from collections import deque
2 import heapq
3
4 # ----- Normal Queue -----
5 class BookQueue:
6     def __init__(self):
7         self.q = deque()
8
9     def enqueue(self, name):
10        self.q.append(name)
11
12     def dequeue(self):
13        return self.q.popleft() if self.q else "Empty"
14
15
16 # ----- Priority Queue -----
17 class PriorityBookQueue:
18     def __init__(self):
19         self.pq = []
20
21     def enqueue(self, name, role): # role = faculty/student
22         priority = 0 if role == "faculty" else 1
23         heapq.heappush(self.pq, (priority, name))
24
25     def dequeue(self):
26        return heapq.heappop(self.pq)[1] if self.pq else "Empty"
27
28
29 # ----- Menu -----
30 choice = input("Choose 1.Queue 2.PriorityQueue: ")
31
32 manager = BookQueue() if choice == "1" else PriorityBookQueue()
33
34 while True:
35     print("\n1.Enqueue 2.Dequeue 3.Exit")
36     ch = input("Enter choice: ")
37
38     if ch == "1":
39         name = input("Enter name: ")
40         manager.enqueue(name)
41
42     elif ch == "2":
43         print("Served:", manager.dequeue())
44
45     elif ch == "3":
46         break
47
48
49
50
```

In 50, Col 14 Spaces: 4 UTF-8 CRLF () Python 3.14.2 Go Live CODEGPT

```
File Edit Selection View Go Run Terminal Help <- > Q aiascode
EXPLORER ... lab1.py A student.py A 11LAB1.PY U 11LAB2.PY U n.py U lab2.py U D ...
AIASSCODE
11LAB1.PY
code.py
electricity_billing.py
lab1.py
lab2.py
LLAB2.PY
multiples.py
n.py
sample.txt
student.py
word_frequency.py

l11AB2.PY > -
17 class PriorityBookQueue:
18     def __init__(self):
19         self.pq = []
20
21     def enqueue(self, name, role): # role = faculty/student
22         priority = 0 if role == "faculty" else 1
23         heapq.heappush(self.pq, (priority, name))
24
25     def dequeue(self):
26        return heapq.heappop(self.pq)[1] if self.pq else "Empty"
27
28
29 # ----- Menu -----
30 choice = input("Choose 1.Queue 2.PriorityQueue: ")
31
32 manager = BookQueue() if choice == "1" else PriorityBookQueue()
33
34 while True:
35     print("\n1.Enqueue 2.Dequeue 3.Exit")
36     ch = input("Enter choice: ")
37
38     if ch == "1":
39         name = input("Enter name: ")
40         if choice == "2":
41             role = input("Role (faculty/student): ")
42             manager.enqueue(name, role)
43         else:
44             manager.enqueue(name)
45
46     elif ch == "2":
47         print("Served:", manager.dequeue())
48
49     elif ch == "3":
50         break
51
52
53
54
```

In 50, Col 14 Spaces: 4 UTF-8 CRLF () Python 3.14.2 Go Live CODEGPT

OUTPUT:

```

    PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/Administrator/Desktop/aiasscode/LLLAB2.PY
    Choose 1.Queue 2.PriorityQueue: 2
    1.Enqueue 2.Dequeue 3.Exit
    Enter choice: 1
    Enter name: HITESH
    Role (faculty/student): 1291
    Served: HITESH

    1.Enqueue 2.Dequeue 3.Exit
    Enter choice: 2
    Served: HITESH

    1.Enqueue 2.Dequeue 3.Exit
    Enter choice: 2
    Served: HITESH

    1.Enqueue 2.Dequeue 3.Exit
    Enter choice: 3
    PS C:\Users\Administrator\Desktop\aiasscode>

```

The screenshot shows a VS Code interface with multiple tabs open in the Explorer panel, including files like lab1.py, student.py, 11LAB1.PY, 11LAB2.PY, n.py, and lab2.py. The terminal window is active and displays a Python script running. The script defines a `PriorityBookQueue` class with an `enqueue` method. It prompts the user to choose between enqueuing, dequeuing, or exiting. When the user chooses to enqueue, it asks for a name and role (faculty or student). It then serves the request and prints the name and role of the served user. This process repeats until the user exits.

Explanation: This program implements a library book request system using a normal queue and a priority queue. The normal queue follows FIFO order and processes requests in the order they arrive. The priority queue separates faculty and student requests and always serves faculty first. The enqueue method adds requests, while the dequeue method removes the correct request based on priority. This approach demonstrates how priority queues handle real-world situations where some requests are more important

TASK3:

PROMPT:

```
# Create an Emergency Help Desk Ticket System using Stack in Python.Implement push, pop, peek operations and methods to check if stack is empty or full.Simulate at least five tickets being raised and resolved to demonstrate LIFO behavior.
```

```
1 # ----- Stack Implementation -----
2 class HelpDeskStack:
3     def __init__(self, size=10):
4         self.stack = []
5         self.size = size
6
7     def push(self, ticket):
8         if len(self.stack) == self.size:
9             print("Stack Full")
10        else:
11            self.stack.append(ticket)
12            print("Ticket added:", ticket)
13
14    def pop(self):
15        if self.is_empty():
16            print("No tickets")
17        else:
18            print("Resolved:", self.stack.pop())
19
20    def peek(self):
21        if self.is_empty():
22            print("No tickets")
23        else:
24            print("Top Ticket:", self.stack[-1])
25
26    def is_empty(self):
27        return len(self.stack) == 0
28
29 desk = HelpDeskStack()
30 desk.push("Login Issue")
31 desk.push("Wifi Not Working")
32 desk.push("Software Install")
33 desk.push("Printer Error")
34 desk.push("System Crash")
35 print("\nPeek:")
36 desk.peek()
37 desk.pop()
38 desk.pop()
39 desk.pop()
```

OUTPUT:

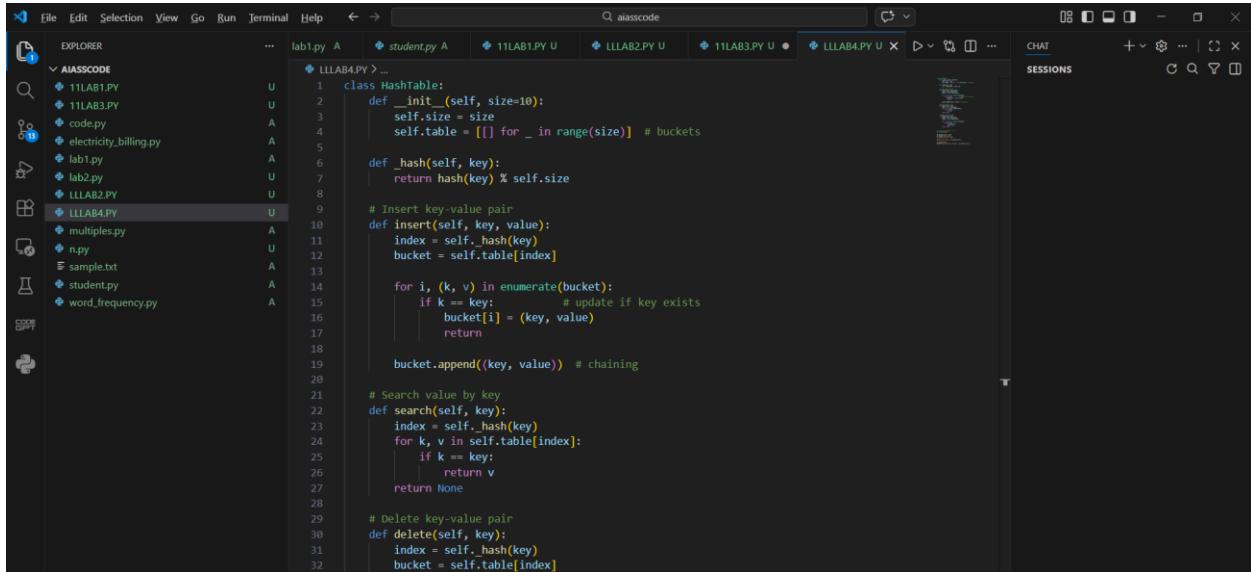
```
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiasscode/11LAB3.PY
xe c:/Users/Administrator/Desktop/aiasscode/11LAB3.PY
Ticket added: Login Issue
Ticket added: Login Issue
Ticket added: Wifi Not Working
Ticket added: Software Install
Ticket added: Printer Error
Ticket added: System Crash
Ticket added: Printer Error
Ticket added: System Crash
Ticket added: System Crash
Ticket added: System Crash
Peek:
Top Ticket: System Crash
Resolving Tickets:
Peek:
Top Ticket: System Crash
Resolving Tickets:
Resolving Tickets:
Resolved: System Crash
Resolved: System Crash
Resolved: Printer Error
Resolved: Software Install
Resolved: Wifi Not Working
Resolved: login Issue
PS C:\Users\Administrator\Desktop\aiasscode>
```

Explanation: This program models an IT help desk using a stack, where support tickets are handled in Last-In, First-Out order. Each ticket stores an ID, requester name, and issue description, and is added to the stack using the push operation. The pop operation resolves the most recently added ticket first, clearly showing LIFO behavior. Additional methods like peek, is_empty, and is_full help manage and check the stack safely.

TASK4:

PROMPT:

Create a Hash Table in Python using chaining for collision handling. Implement insert, search, and delete methods with proper comments. Test the hash table with some sample data.



```
File Edit Selection View Go Run Terminal Help <- > Q aiascode
EXPLORER ... lab1.py A student.py A 11LAB1.PY U 11LAB2.PY U 11LAB3.PY U 11LAB4.PY U ...
AIASCODE 11LAB1.PY 11LAB3.PY code.py electricity_billing.py lab1.py lab2.py LLLAB2.PY LLLAB4.PY multiples.py n.py sample.txt student.py word_frequency.py
1
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)] # buckets

    def _hash(self, key):
        return hash(key) % self.size

    # Insert key-value pair
    def insert(self, key, value):
        index = self._hash(key)
        bucket = self.table[index]

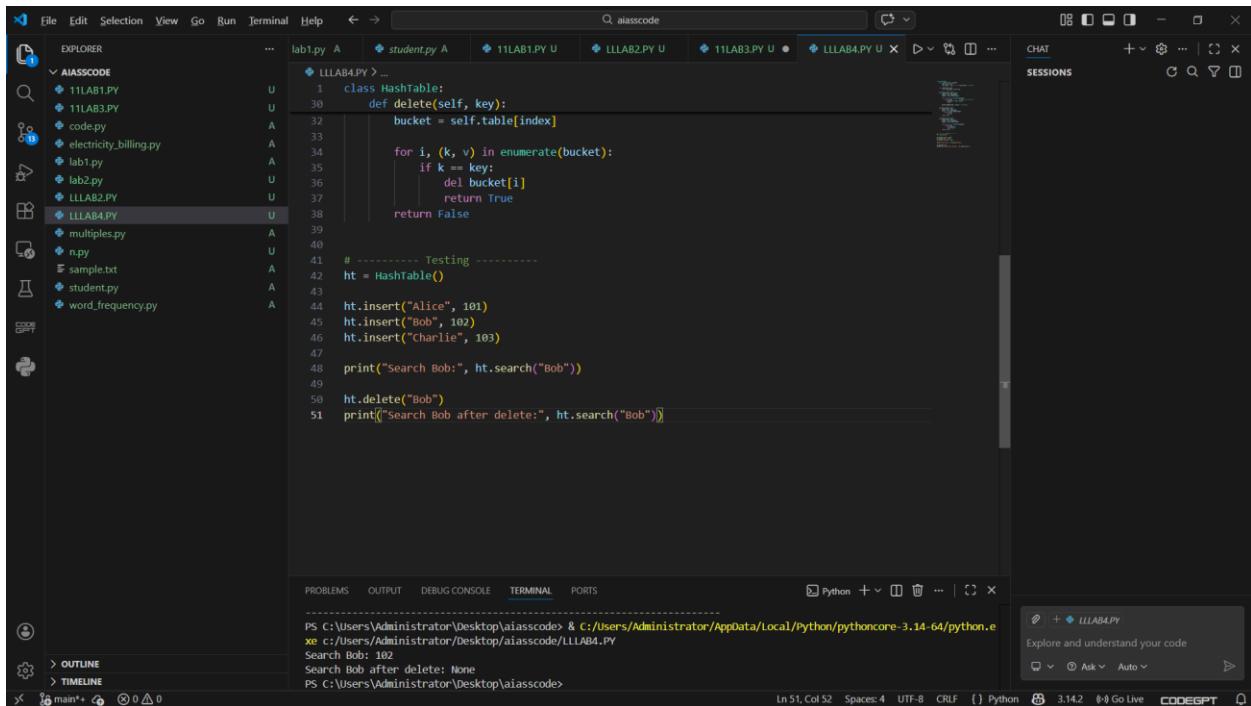
        for i, (k, v) in enumerate(bucket):
            if k == key:
                # update if key exists
                bucket[i] = (key, value)
                return

        bucket.append((key, value)) # chaining

    # Search value by key
    def search(self, key):
        index = self._hash(key)
        for k, v in self.table[index]:
            if k == key:
                return v
        return None

    # Delete key-value pair
    def delete(self, key):
        index = self._hash(key)
        bucket = self.table[index]
```

OUTPUT:



```
File Edit Selection View Go Run Terminal Help <- > Q aiascode
EXPLORER ... lab1.py A student.py A 11LAB1.PY U 11LAB2.PY U 11LAB3.PY U 11LAB4.PY U ...
AIASCODE 11LAB1.PY 11LAB3.PY code.py electricity_billing.py lab1.py lab2.py LLLAB2.PY LLLAB4.PY multiples.py n.py sample.txt student.py word_frequency.py
1
class HashTable:
    def delete(self, key):
        bucket = self.table[index]

        for i, (k, v) in enumerate(bucket):
            if k == key:
                del bucket[i]
                return True

        return False

    # ----- Testing -----
    ht = HashTable()
    ht.insert("Alice", 101)
    ht.insert("Bob", 102)
    ht.insert("Charlie", 103)

    print("Search Bob:", ht.search("Bob"))

    ht.delete("Bob")
    print("Search Bob after delete:", ht.search("Bob"))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\Administrator\Desktop\aiascode> & C:/Users/Administrator/AppData/Local/Python/pythoncore-3.14-64/python.e
xe c:/Users/Administrator/Desktop/aiascode/LLLAB4.PY
Search Bob: 102
Search Bob after delete: None
PS C:\Users\Administrator\Desktop\aiascode>

CODEGPT

Explanation: This program implements a hash table using the chaining method to handle collisions. Each table index stores a linked list of nodes, allowing multiple keys to share the same index safely. The `insert()` method places a key-value pair into the appropriate bucket or updates it if the key already exists. The `search()` method traverses the linked list at the computed index to find the required key. The `delete()` method removes the node without breaking the chain, ensuring correct hash table behavior

TASK5:

PROMPT:

Create a Student Attendance Tracking System using a dictionary (hash table). Implement functions to mark attendance, search attendance, and display all records. Use a simple menu-driven program.

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a folder named "AIASSCODE" containing several Python files: 11LAB1.PY, 11LAB3.PY, 11LAB5.PY, code.py, electricity_billing.py, lab1.py, lab2.py, LLLAB2.PY, LLLAB4.PY, multiples.py, n.py, sample.txt, student.py, and word_frequency.py.
- Code Editor:** The main pane displays the content of 11LAB5.PY. The code defines a class `AttendanceSystem` with methods for marking attendance, searching attendance, and displaying all records. It also includes a menu loop.
- Bottom Status Bar:** Shows file statistics: In 43, Col 5, Spaces: 4, UTF-8, CRLF, Python 3.14.2, and Go Live.
- CodeGPT Overlay:** A small window titled "11LAB5.PY" is open, showing the text "Explore and understand your code". It has buttons for "Ask", "Auto", and "Go".

```
1 # Attendance System using Hash Table (Dictionary)
2
3 class AttendanceSystem:
4     def __init__(self):
5         self.records = {} # student_id : status
6
7     def mark_attendance(self, student_id, status):
8         self.records[student_id] = status
9         print("Attendance marked.")
10
11     def search_attendance(self, student_id):
12         if student_id in self.records:
13             print("Status:", self.records[student_id])
14         else:
15             print("Student not found.")
16
17     def display_all(self):
18         for sid, status in self.records.items():
19             print(sid, ":", status)
20
21
22     # Menu
23     system = AttendanceSystem()
24
25     while True:
26         print("\n1.Mark 2.Search 3.Display 4.Exit")
27         ch = input("Enter choice: ")
28
29         if ch == "1":
30             sid = input("Student ID: ")
31             status = input("Present/Absent: ")
32             system.mark_attendance(sid, status)
33
34         elif ch == "2":
35             sid = input("Student ID: ")
36             system.search_attendance(sid)
37
38         elif ch == "3":
39             system.display_all()
```

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files in the 'AISSCODE' folder, including 11LAB1.PY, 11LAB3.PY, 11LAB5.PY, code.py, electricity_billing.py, lab1.py, lab2.py, LLLAB2.PY, LLLAB4.PY, multiples.py, n.npy, sample.txt, student.py, and word_frequency.py.
- Code Editor:** The active file is 11LAB5.PY, containing Python code for an Attendance System. The code defines a class 'AttendanceSystem' with methods for marking attendance, searching for a student, displaying all students, and exiting the program.
- Terminal:** The terminal shows the command 'code .'. The code editor status bar indicates 'Python 3.14.2'.

```
class AttendanceSystem:
    def __init__(self):
        self.attendance = {}
    def mark_attendance(self, sid, status):
        self.attendance[sid] = status
    def search_attendance(self, sid):
        if sid in self.attendance:
            print(f"Student ID: {sid} Present/Absent: {self.attendance[sid]}")
        else:
            print("Student ID not found")
    def display_all(self):
        for sid, status in self.attendance.items():
            print(f"Student ID: {sid} Present/Absent: {status}")
    def exit(self):
        print("Exiting program")
    def main(self):
        system = AttendanceSystem()
        while True:
            print("\n1.MARK 2.SEARCH 3.DISPLAY 4.EXIT")
            ch = input("Enter choice: ")
            if ch == "1":
                sid = input("Student ID: ")
                status = input("Present/Absent: ")
                system.mark_attendance(sid, status)
            elif ch == "2":
                sid = input("Student ID: ")
                system.search_attendance(sid)
            elif ch == "3":
                system.display_all()
            elif ch == "4":
                break
```

OUTPUT:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Same as the previous screenshot, showing files in the 'AISSCODE' folder.
- Terminal:** The terminal shows the execution of 11LAB5.PY and its output. The user enters choices 1, 2, 3, and 4, demonstrating the marking, searching, displaying, and exiting functionality of the attendance system.
- Status Bar:** Shows 'Python 3.14.2' and other VS Code status indicators.

```
xe c:/Users/Administrator/Desktop/aiasscode/11LAB5.PY
1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 1
Student ID: HITHESH
Present/Absent: 23
Attendance marked.

1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 1
Student ID: KARTHIK
Present/Absent: 20
Attendance marked.

1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 3
HITHESH : 23
KARTHIK : 20

1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 2HITHESH
1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 3
HITHESH : 23
KARTHIK : 20

1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 2HITHESH
1.MARK 2.SEARCH 3.DISPLAY 4.EXIT
Enter choice: 4
PS C:\Users\Administrator\Desktop\aiasscode>
```

Explanation: I selected the Library Book Borrowing System because it requires fast access, insertion, and deletion of book records. A hash table is used where the key is book_id and the value is book_name. Adding a book stores the key-value pair directly, giving O(1) average time complexity. Removing a book is efficient because the book ID allows direct lookup without traversal.