27.6.25 CODING CHALLENGE

Sections:

1. Python Programming & OOP (40 mins)
2. Data Structures & Algorithms (30 mins)
3. SQL with Python Integration (30 mins)
4. Version Control with Git (10 mins)
5. Bonus/Stretch Task: Unit Testing with PyUnit (10 mins)

Section 1: Python Programming & OOP (40 mins)

Q1. Functional Coding Challenge – Movie Booking System (20 mins)

- Show available movies (stored in a list)

- Allow user to select movie & number of tickets

- Calculate and show total amount (use a dictionary to store movie:price)

- Use functions for showing movies, booking logic, and calculating amount

```
movies = {"Avengers": 150, "Spiderman": 120, "Toy Story": 100}
def show_movies():
    print("Available Movies:")
    for movie in movies:
     print(f"- {movie} (Rs. {movies[movie]})")
def calculate_amount(movie, tickets):
    return movies[movie] * tickets
def book_movie():
    show_movies()
    movie = input("Enter movie name: ")
    tickets = int(input("Enter number of tickets: "))
    total = calculate_amount(movie, tickets)
    print(f"Total Amount: Rs. {total}")
    book_movie()
```

```
arci/OneDrive/Desktop/CC 2/Unt
Available Movies:
- Avengers (Rs. 150)
- Spiderman (Rs. 120)
- Toy Story (Rs. 100)
Enter movie name: Toy Story
Enter number of tickets: 2
Total Amount: Rs. 200
PS C:\Users\harci\OneDrive\Des
```

Q2. OOP Implementation – Library Management (20 mins)

- Create classes Book, Library, and User

- Library contains a collection of books

- User can borrow/return/view books

- Use class, constructor, inheritance, method overriding

```
class Book:
    def __init__(self, title):
        self.title = title
class Library:
    def __init__(self):
        self.books = [Book("Python"), Book("DSA"), Book("AI")]
    def display_books(self):
        for book in self.books:
            print(f"- {book.title}")
    def borrow_book(self, title):
        for book in self.books:
            if book.title == title:
                self.books.remove(book)
                return f"{title} borrowed."
        return "Book not available."
class User(Library):  # Inheriting Library
    def return_book(self, title):
        self.books.append(Book(title))
        return f"{title} returned."
u = User()
u.display_books()
print(u.borrow_book("DSA"))
u.display_books()
print(u.return_book("DSA"))
```

```
PS C:\Users\harci\OneDrive
arci/OneDrive/Desktop/CC 2
- Python
- DSA
- AI
DSA borrowed.
- Python
- AI
DSA returned.
PS C:\Users\harci\OneDrive
```

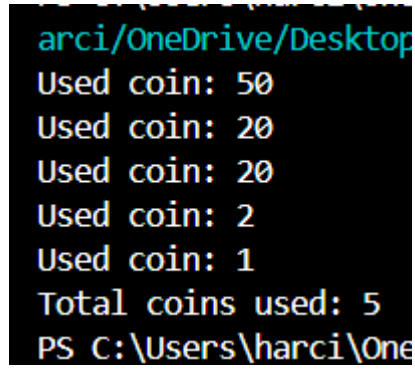
Section 2: Data Structures & Algorithms (30 mins)

Q3. Algorithm Problem – Minimize Coins (Greedy) (15 mins)

- Find minimum number of coins needed for a given amount

- Denominations: [1, 2, 5, 10, 20, 50, 100, 200, 500]

```
def min_coins(amount):
    coins = [500, 200, 100, 50, 20, 10, 5, 2, 1]
    count = 0
    for coin in coins:
        while amount >= coin:
            amount -= coin
            count += 1
            print(f"Used coin: {coin}")
    return count
print("Total coins used:", min_coins(93))
```



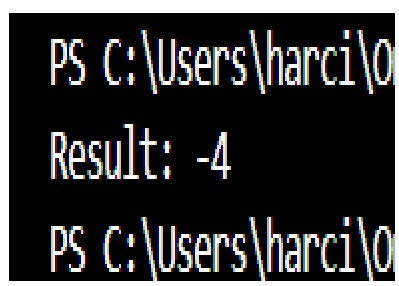
Q4. Data Structure Usage (15 mins)

- Stack: Evaluate postfix expression '231*+9-'

- Linked List class: append(), display(), reverse()

a) Stack: Evaluate postfix expression '231*+9-'

```
def eval_postfix(expr):
    stack = []
    for ch in expr:
        if ch.isdigit():
            stack.append(int(ch))
        else:
            b = stack.pop()
            a = stack.pop()
            if ch == '+': stack.append(a + b)
```



```
        elif ch == '-': stack.append(a - b)
        elif ch == '*': stack.append(a * b)
        elif ch == '/': stack.append(a // b)
    return stack[0]
print("Result:", eval_postfix("231*+9-"))
```

b) Linked List class: append(), display(), reverse()

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def append(self, val):
        new = Node(val)
        if not self.head:
            self.head = new
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new
    def display(self):
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

```
PS C:\Users\harci\OneDri
10 -> 20 -> 30 -> None
30 -> 20 -> 10 -> None
PS C:\Users\harci\OneDri
```

```
    def reverse(self):
        prev = None
        current = self.head
        while current:
            nxt = current.next
            current.next = prev
            prev = current
            current = nxt
        self.head = prev
ll = LinkedList()
ll.append(10)
ll.append(20)
ll.append(30)
ll.display()
ll.reverse()
ll.display()
```

Section 3: SQL with Python Integration (30 mins)

Q5. SQL + Python – Student Scores Table

- Create table StudentScores(name VARCHAR, subject VARCHAR, marks INT)

- Insert sample data

- Use Python to display records, show average marks, list students scoring <40

```
    import sqlite3
conn = sqlite3.connect(':memory:')
cur = conn.cursor()
cur.execute("CREATE TABLE StudentScores(name TEXT, subject TEXT, marks INT)")
cur.executemany("INSERT INTO StudentScores VALUES (?, ?, ?)", [
    ('Harci', 'Math', 85),
```

```
PS C:\Users\harci\OneDrive\De
All Records:
('Harci', 'Math', 85)
('Niha', 'Science', 35)
('Jhara', 'Math', 92)
('Dhivi', 'Science', 28)

Average Marks: 60.0

Students with marks < 40:
Niha
Dhivi
PS C:\Users\harci\OneDrive\De
```

```
    ('Niha', 'Science', 35),
    ('Jhara', 'Math', 92),
    ('Dhivi', 'Science', 28)
])
print("All Records:")
for row in cur.execute("SELECT * FROM StudentScores"):
    print(row)
cur.execute("SELECT AVG(marks) FROM StudentScores")
print("\nAverage Marks:", cur.fetchone()[0])
print("\nStudents with marks < 40:")
for row in cur.execute("SELECT name FROM StudentScores WHERE marks < 40"):
 print(row[0])
```

Section 4: Version Control with Git (10 mins)

Q6. Git Challenge

- Initialize Git repository

```
PS C:\Users\harci\OneDrive\Desktop\CC 2> git init
>>
Initialized empty Git repository in C:/Users/harci/OneDrive/Desktop/CC 2/.git/
PS C:\Users\harci\OneDrive\Desktop\CC 2>
```

- Create and switch to branch feature/students

```
PS C:\Users\harci\OneDrive\Desktop\CC 2> git checkout -b feature/students
>>
Switched to a new branch 'feature/students'
PS C:\Users\harci\OneDrive\Desktop\CC 2>
```

- Add and commit your Python code

```
PS C:\Users\harci\OneDrive\Desktop\CC 2> git add Untitled-1.py
PS C:\Users\harci\OneDrive\Desktop\CC 2> git commit -m "student score management"
[feature/students (root-commit) 31dd3c4] student score management
 1 file changed, 152 insertions(+)
 create mode 100644 Untitled-1.py
PS C:\Users\harci\OneDrive\Desktop\CC 2>
```

- Merge feature/students into main

```
PS C:\Users\harci\OneDrive\Desktop\CC 2> git merge feature/students
>>
Already up to date.
PS C:\Users\harci\OneDrive\Desktop\CC 2>
```

- Provide Git commands

Initialize a Git repository - **git init**

Check Git status (optional, to view untracked files) - **git status**

Create and switch to a new branch named feature/students -**git checkout -b feature/students**

Add your Python file (change name if needed) - **git add Untitled-1.py**

Commit the file with a meaningful message - **git commit -m "student score management"**

Switch back to the main branch - **git checkout main**

Merge feature/students into main - **git merge feature/students**

Bonus Section: PyUnit Test Case (10 mins)

Q7. PyUnit test cases for Q1 (Booking System)

- 1 test case for calculate_amount()

- 1 test case for booking() using mocks if needed

- Use unittest.TestCase, setUp(), tearDown()

```
import unittest

movies = {"Avengers": 150, "Spiderman": 120}

def calculate_amount(movie, tickets):

    return movies[movie] * tickets

def book_movie(movie, tickets):

    return f"Total: Rs. {calculate_amount(movie, tickets)}"

class TestBooking(unittest.TestCase):

    def setUp(self):
```

```
Setting up test...
Tear down complete.
.
Setting up test...
Tear down complete.
.
----------------------------------------
Ran 2 tests in 0.002s

OK
PS C:\Users\harci\OneDrive\Desktop\CC 2>
```

```
        print("\nSetting up test...")
    def test_amount(self):
        self.assertEqual(calculate_amount("Avengers", 2), 300)
    def test_booking_output(self):
        self.assertIn("Total: Rs. 240", book_movie("Spiderman", 2))
    def tearDown(self):
        print("Tear down complete.")
unittest.main(argv=["], exit=False)
```