Coding Challenge

Total Duration: 2 Hours
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 = {
    "Pushpa": 150,
    "Dangal": 180,
    "Bahubali": 200,
    "RRR": 120
}

def show_movies():
    print("Available Movies:")
    for i, (movie, price) in enumerate(movies.items(), 1):
        print(f"{i}. {movie} - ₹{price}")

def book_movie():
    show_movies()
    choice = int(input("Select movie number: "))
    movie_list = list(movies.keys())

    if 1 <= choice <= len(movie_list):
        selected_movie = movie_list[choice - 1]
        num_tickets = int(input("Enter number of tickets: "))
        total = calculate_amount(selected_movie, num_tickets)
        print(f"Booking confirmed for '{selected_movie}' - Total: ₹{total}")
        return selected_movie, total
    else:
        print("Invalid selection.")
        return None, 0
```

```
def calculate_amount(movie, tickets):
    return movies[movie] * tickets

if __name__ == "__main__":
    book_movie()
```

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, author):
        self.title = title
        self.author = author
        self.is_borrowed = False

    def __str__(self):
        return f"{self.title} by {self.author}"


class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

    def view_books(self):
        print("\nLibrary Books:")
        if not self.books:
            print("No books in the library.")
            return
        for book in self.books:
            status = "Available" if not book.is_borrowed else "Borrowed"
            print(f"{book} - {status}")

    def borrow_book(self, book_title):
        for book in self.books:
            if book.title.lower() == book_title.lower() and not book.is_borrowed:
                book.is_borrowed = True
                print(f"You borrowed '{book.title}'")
                return
```

```
        print("Book not available.")

    def return_book(self, book_title):
        for book in self.books:
            if book.title.lower() == book_title.lower() and book.is_borrowed:
                book.is_borrowed = False
                print(f"You returned '{book.title}'")
                return
        print("Book not found or not borrowed.")


class User(Library):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def __str__(self):
        return f"User: {self.name}"

lib = Library()
user1 = User("Alice")

while True:
    print("\n=== Library Menu ===")
    print("1. Add Book")
    print("2. View Books")
    print("3. Borrow Book")
    print("4. Return Book")
    print("5. Exit")

    choice = input("Enter your choice: ")

    if choice == "1":
        title = input("Enter book title: ")
        author = input("Enter author name: ")
        lib.add_book(Book(title, author))
        print("Book added successfully.")

    elif choice == "2":
        lib.view_books()

    elif choice == "3":
        title = input("Enter book title to borrow: ")
        lib.borrow_book(title)
```

```
    elif choice == "4":
        title = input("Enter book title to return: ")
        lib.return_book(title)

    elif choice == "5":
        print("Exiting... Goodbye!")
        break

    else:
        print("Invalid option. Please choose from 1 to 5.")
```

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 minimize_coins(amount):
    denominations = [500, 200, 100, 50, 20, 10, 5, 2, 1]
    result = []

    for coin in denominations:
        while amount >= coin:
            amount -= coin
            result.append(coin)

    print("Minimum coins needed:")
    print(result)
    print("Total coins used:", len(result))

# Example usage
amount = int(input("Enter the amount: "))
minimize_coins(amount)
```

Q4. Data Structure Usage (15 mins)

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

```
def evaluate_postfix(expression):
    stack = []

    for ch in expression:
        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(int(a / b))

    return stack.pop()

# Example usage
expr = '231*+9-'
result = evaluate_postfix(expr)
print(f"Postfix Evaluation Result: {result}")
```

- 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, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return

        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        temp = self.head
        while temp:
```

```
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")

    def reverse(self):
        prev = None
        curr = self.head
        while curr:
            nxt = curr.next
            curr.next = prev
            prev = curr
            curr = nxt
        self.head = prev

# Main code with user input
ll = LinkedList()

n = int(input("Enter number of values to append: "))
for i in range(n):
    val = int(input(f"Enter value {i+1}: "))
    ll.append(val)

print("\nOriginal Linked List:")
ll.display()

ll.reverse()
print("Reversed Linked List:")
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

Database Creation:

```
create database schooldb;
use schooldb;
create table studentscores(student_name varchar(60),student_subject
varchar(60),student_marks int);
insert into studentscores values("Anu","math",90);
insert into studentscores values("Basha","science",67);
insert into studentscores values("Chandan","math",23);
insert into studentscores values("Dinesh","jistory",35);
```

Python Code:

```
import mysql.connector
conn = mysql.connector.connect(
    host="localhost",
    user="root",
    password="sunnybunny.123",
    database="schooldb"
)
cursor = conn.cursor()

print("\n--- All Records ---")
cursor.execute("SELECT * FROM studentscores")
for row in cursor.fetchall():
    print(row)

cursor.execute("SELECT AVG(student_marks) FROM studentscores")
avg = cursor.fetchone()[0]
print(f"\nAverage Marks: {avg:.2f}")

print("\n--- Students Scoring Less Than 40 ---")
cursor.execute("SELECT student_name, student_subject, student_marks FROM studentscores
WHERE student_marks < 40")
for row in cursor.fetchall():
    print(row)

cursor.close()
conn.close()
```

Section 4: Version Control with Git (10 mins)

Q6. Git Challenge

- Initialize Git repository

Git init

- Create and switch to branch feature/students

git checkout -b feature/students

- Add and commit your Python code

git add Student_marks.py

- Merge feature/students into main

git commit -m "Add student marks Python script"

git checkout main

- Provide Git commands

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
from unittest.mock import patch
from movie_booking import calculate_amount, book_movie

class TestBookingSystem(unittest.TestCase):

    def setUp(self):
        print("\nSetting up test case...")

    def tearDown(self):
        print("Cleaning up after test.")

    def test_calculate_amount(self):
        result = calculate_amount("Pushpa", 3)
        self.assertEqual(result, 450)

    @patch("builtins.input", side_effect=["2", "2"])
    def test_book_movie(self, mock_input):
        movie, total = book_movie()
        self.assertEqual(movie, "Dangal")
        self.assertEqual(total, 360)

if __name__ == "__main__":
    unittest.main()
```