# 27-06-25-Coding Challenge

## Section 1: Python Programming & OOP

Q1. Functional Coding Challenge – Movie Booking System   
- Show available movies (stored in a list)

**movies = ["Movie A", "Movie B", "Movie C"]**

**prices = {"Movie A": 10, "Movie B": 12, "Movie C": 8}**

- Allow user to select movie & number of tickets

**def show\_movies():**

**print("Available Movies:")**

**for movie in movies:**

**print(movie)**

**def book\_movie():**

**show\_movies()**

**selected\_movie = input("Select a movie: ")**

**num\_tickets = int(input("Number of tickets: "))**

**total\_amount = calculate\_amount(selected\_movie, num\_tickets)**

**print(f"Total amount for {num\_tickets} tickets of {selected\_movie}: ${total\_amount}")**

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

**def calculate\_amount(movie, tickets):**

**return prices[movie] \* tickets**

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

**book\_movie()**

Q2. OOP Implementation – Library Management

- 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**

**class Library:**

**def \_\_init\_\_(self):**

**self.books = []**

**def add\_book(self, book):**

**self.books.append(book)**

**def view\_books(self):**

**for book in self.books:**

**print(f"{book.title} by {book.author}")**

**class User:**

**def \_\_init\_\_(self, name):**

**self.name = name**

**self.borrowed\_books = []**

**def borrow\_book(self, book, library):**

**library.books.remove(book)**

**self.borrowed\_books.append(book)**

**def return\_book(self, book, library):**

**self.borrowed\_books.remove(book)**

**library.add\_book(book)**

## Section 2: Data Structures & Algorithms

Q3. Algorithm Problem – Minimize Coins (Greedy)  
- 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]**

**coin\_count = 0**

**for coin in denominations:**

**while amount >= coin:**

**amount -= coin**

**coin\_count += 1**

**return coin\_count**

Q4. Data Structure Usage   
- Stack: Evaluate postfix expression '231\*+9-'

**def evaluate\_postfix(expression):**

**stack = []**

**for char in expression:**

**if char.isdigit():**

**stack.append(int(char))**

**else:**

**b = stack.pop()**

**a = stack.pop()**

**if char == '+':**

**stack.append(a + b)**

**elif char == '\*':**

**stack.append(a \* b)**

**elif char == '-':**

**stack.append(a - b)**

**return stack[0]**

- 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**

**last = self.head**

**while last.next:**

**last = last.next**

**last.next = new\_node**

**def display(self):**

**current = self.head**

**while current:**

**print(current.data, end=" -> ")**

**current = current.next**

**print("None")**

**def reverse(self):**

**prev = None**

**current = self.head**

**while current:**

**next\_node = current.next**

**current.next = prev**

**prev = current**

**current = next\_node**

**self.head = prev**

## Section 3: SQL with Python Integration

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('school.db')**

**cursor = conn.cursor()**

**cursor.execute('''CREATE TABLE StudentScores (name VARCHAR, subject VARCHAR, marks INT)''')**

**cursor.execute("INSERT INTO StudentScores VALUES ('Alice', 'Math', 85)")**

**cursor.execute("INSERT INTO StudentScores VALUES ('Bob', 'Math', 35)")**

**conn.commit()**

## 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 .**

**git commit -m "Add movie booking system and library management"**- Merge feature/students into main

**git checkout main**

**git merge feature/students**  
- Provide Git commands

## Bonus Section: PyUnit Test Case

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**

**class TestBookingSystem(unittest.TestCase):**

**def setUp(self):**

**self.movie = "Movie A"**

**self.tickets = 2**

**def test\_calculate\_amount(self):**

**self.assertEqual(calculate\_amount(self.movie, self.tickets), 20)**

**def test\_booking(self):**

**# Mocking user input for booking**

**with unittest.mock.patch('builtins.input', side\_effect=[self.movie, self.tickets]):**

**with unittest.mock.patch('builtins.print') as mocked\_print:**

**book\_movie()**

**mocked\_print.assert\_called\_with("Total amount for 2 tickets of Movie A: $20")**

**if \_\_name\_\_ == '\_\_main\_\_':**

**unittest.main()**