# Coding Challenge (27-06-2025)

-BY GETSY JACINTH

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

**CODE:**

**movies = ["Jawan", "Interstellar", "Dune", "Barbie"]**

**movie\_prices = {"Jawan": 250, "Interstellar": 300, "Dune": 280, "Barbie": 260}**

**def show\_movies():**

**for idx, movie in enumerate(movies, 1):**

**print(f"{idx}. {movie} - Rs.{movie\_prices[movie]}")**

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

**return movie\_prices[movie] \* tickets**

**def book\_movie():**

**show\_movies()**

**choice = int(input("Enter movie number: "))**

**movie = movies[choice - 1]**

**tickets = int(input("Enter number of tickets: "))**

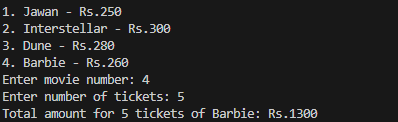
**total = calculate\_amount(movie, tickets)**

**print(f"Total amount for {tickets} tickets of {movie}: Rs.{total}")**

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

**book\_movie()**

**OUTPUT:**

****

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

**CODE:**

**class Book:**

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

**self.title = title**

**self.is\_borrowed = False**

**class Library:**

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

**self.books = [Book("Python 101"), Book("DSA "), Book("OOP")]**

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

**for book in self.books:**

**status = "Available" if not book.is\_borrowed else "Borrowed"**

**print(f"{book.title} - {status}")**

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

**for book in self.books:**

**if book.title == book\_title 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 == book\_title and book.is\_borrowed:**

**book.is\_borrowed = False**

**print(f"You returned {book.title}")**

**return**

**print("Invalid return")**

**class User(Library):**

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

**super().\_\_init\_\_()**

**self.name = name**

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

**print(f"{self.name}'s Library View:")**

**super().show\_books()**

**user = User("Alice")**

**user.show\_books()**

**user.borrow\_book("Python 101")**

**user.show\_books()**

**user.return\_book("Python 101")**

**user.show\_books()**

**OUTPUT SCREEENSHOT:**

****

## 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]

**CODE:**

**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)**

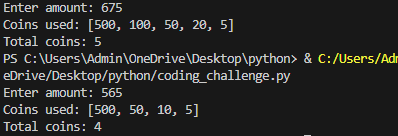
**print(f"Coins used: {result}")**

**print(f"Total coins: {len(result)}")**

**amount = int(input("Enter amount: "))**

**minimize\_coins(amount)**

**OUTPUT:**

****

Q4. Data Structure Usage (15 mins)  
- Stack: Evaluate postfix expression '231\*+9-'

**CODE:**

**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)**

**elif char == '/':**

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

**print(f"Result: {stack[0]}")**

**expression = '231\*+9-'**

**evaluate\_postfix(expression)**

**OUTPUT SCREENSHOT:**

****

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

**CODE:**

**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()**

**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**

**ll = LinkedList()**

**ll.append(10)**

**ll.append(20)**

**ll.append(30)**

**ll.display()**

**ll.reverse()**

**ll.display()**

**OUPUT SCREENSHOT:**

****

## 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

**CODE:**

**import sqlite3**

**conn = sqlite3.connect(":memory:")**

**cursor = conn.cursor()**

**cursor.execute("""**

**CREATE TABLE StudentScores(**

**name TEXT,**

**subject TEXT,**

**marks INTEGER**

**)**

**""")**

**sample\_data = [**

**("Vijay", "Math", 85),**

**("Varun", "Math", 35),**

**("Latha", "Science", 72),**

**("David", "Science", 25),**

**("Eve", "English", 90),**

**("Frank", "English", 38)**

**]**

**cursor.executemany("INSERT INTO StudentScores VALUES (?, ?, ?)", sample\_data)**

**conn.commit()**

**print("All Records:")**

**for row in cursor.execute("SELECT \* FROM StudentScores"):**

**print(row)**

**cursor.execute("SELECT AVG(marks) FROM StudentScores")**

**avg\_marks = cursor.fetchone()[0]**

**print(f"\nAverage Marks: {avg\_marks}")**

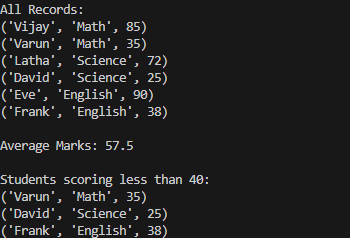
**print("\nStudents scoring less than 40:")**

**for row in cursor.execute("SELECT \* FROM StudentScores WHERE marks < 40"):**

**print(row)**

**conn.close()**

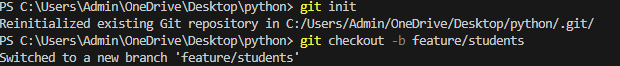
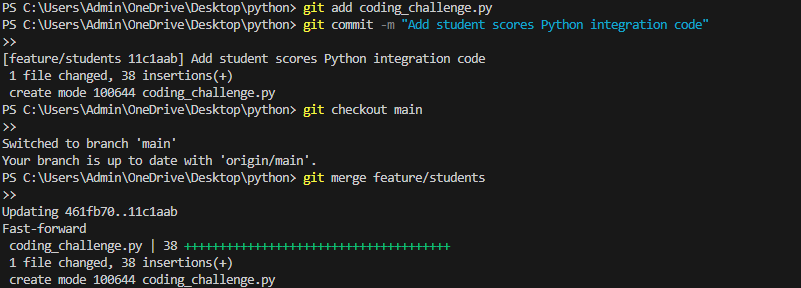
**OUTPUT SCREENSHOT:**

****

## Section 4: Version Control with Git (10 mins)

Q6. Git Challenge  
- Initialize Git repository  
- Create and switch to branch feature/students  
- Add and commit your Python code  
- Merge feature/students into main  
- Provide Git commands

**COMMANDS AND OUTPUT SCREENSHOT:**

**** ****

## 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()

**CODE:**

**import unittest**

**import coding\_challenge**

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

**def setUp(self):**

**self.movie\_prices = coding\_challenge.movie\_prices**

**def tearDown(self):**

**self.movie\_prices = None**

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

**amount = coding\_challenge.calculate\_amount("Jawan", 2)**

**self.assertEqual(amount, 500)**

**amount = coding\_challenge.calculate\_amount("Interstellar", 0)**

**self.assertEqual(amount, 0)**

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

**# Simulate the internal logic: get price and calculate**

**movie = "Dune"**

**tickets = 3**

**expected\_total = self.movie\_prices[movie] \* tickets**

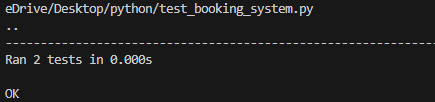
**actual\_total = coding\_challenge.calculate\_amount(movie, tickets)**

**self.assertEqual(actual\_total, expected\_total)**

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

**unittest.main()**

**OUTPUT SCREENSHOT;**

****