**Day 15 - 27-06-2025**

**CODING CHALLENGE - PYTHON**

## **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

movie\_list = ["Inception", "Interstellar", "The Matrix", "Avengers: Endgame", "The Lion King"]

movie\_prices = {"Inception": 100, "Interstellar": 120, "The Matrix": 130, "Avengers: Endgame": 110, "The Lion King": 100}

def show\_movies():

print("Available Movies:")

for i in range(len(movie\_list)):

print(str(i + 1) + ". " + movie\_list[i] + " - " + str(movie\_prices[movie\_list[i]]))

def calculate\_total(price, tickets):

return price \* tickets

def book\_movie():

show\_movies()

choice = int(input("Enter the number of the movie you want to book: "))

if choice < 1 or choice > len(movie\_list):

print("Invalid movie choice.")

return

selected\_movie = movie\_list[choice - 1]

tickets = int(input(f"How many tickets do you want for {selected\_movie}? "))

if tickets < 1:

print("You must book at least 1 ticket.")

return

price = movie\_prices[selected\_movie]

total = calculate\_total(price, tickets)

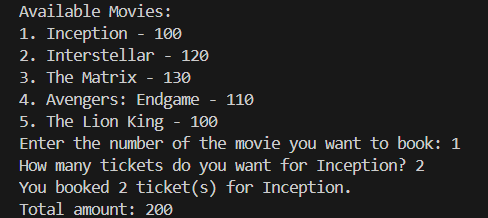
print(f"You booked {tickets} ticket(s) for {selected\_movie}.")

print(f"Total amount: {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

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("Books in Library:")

for book in self.books:

status = "Borrowed" if book.is\_borrowed else "Available"

print(f"- {book} [{status}]")

def borrow\_book(self, title):

for book in self.books:

if book.title == title and not book.is\_borrowed:

book.is\_borrowed = True

print(f"You have borrowed '{book.title}'.")

return

print("Book not available or already borrowed.")

def return\_book(self, title):

for book in self.books:

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

book.is\_borrowed = False

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

return

print("You can't return this book.")

class User(Library):

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

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

self.name = name

def borrow\_book(self, title):

print(f"{self.name} is trying to borrow a book...")

super().borrow\_book(title)

def return\_book(self, title):

print(f"{self.name} is trying to return a book...")

super().return\_book(title)

book1 = Book("Harry Potter", "J.K. Rowling")

book2 = Book("The Hobbit", "J.R.R. Tolkien")

book3 = Book("1984", "George Orwell")

my\_library = Library()

my\_library.add\_book(book1)

my\_library.add\_book(book2)

my\_library.add\_book(book3)

user1 = User("Alice")

user1.books = my\_library.books

user1.view\_books()

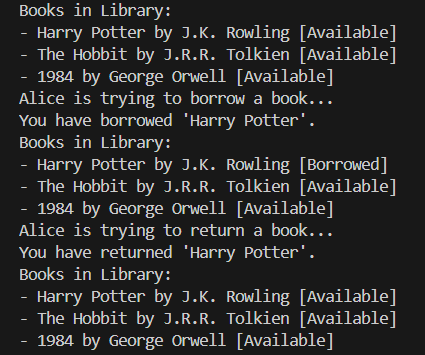
user1.borrow\_book("Harry Potter")

user1.view\_books()

user1.return\_book("Harry Potter")

user1.view\_books()

**Output:**

****

## **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):

coins = [500, 200, 100, 50, 20, 10, 5, 2, 1]

result = []

for coin in coins:

while amount >= coin:

amount -= coin

result.append(coin)

print("Coins used:")

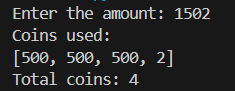
print(result)

print("Total coins:", len(result))

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

minimize\_coins(amount)

**Output:**

****

**Q4. Data Structure Usage (15 mins)**

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

def evaluate\_postfix(expression):

stack = []

for char in expression:

if char.isdigit():

stack.append(int(char))

else:

# Pop last two values

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) # Integer division

return stack.pop()

# Example usage

expr = "231\*+9-"

result = evaluate\_postfix(expr)

print("Postfix Result:", result)

**Output:**

****

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

# Node class

class Node:

def \_\_init\_\_(self, value):

self.data = value

self.next = None

# Linked List class

class LinkedList:

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

self.head = None

# Add a new node at the end

def append(self, value):

new\_node = Node(value)

if self.head is None:

self.head = new\_node

else:

temp = self.head

while temp.next is not None:

temp = temp.next

temp.next = new\_node

# Display all node values

def display(self):

temp = self.head

while temp is not None:

print(temp.data, end=" -> ")

temp = temp.next

print("None")

# Reverse the linked list

def reverse(self):

prev = None

current = self.head

while current is not None:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

self.head = prev

# Example usage

ll = LinkedList()

ll.append(1)

ll.append(2)

ll.append(3)

print("Original list:")

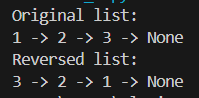
ll.display()

ll.reverse()

print("Reversed list:")

ll.display()

**Output:**

****

## **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:')

c = conn.cursor()

c.execute("CREATE TABLE StudentScores (name TEXT, subject TEXT, marks INTEGER)")

c.execute("INSERT INTO StudentScores VALUES ('Alice', 'Math', 75)")

c.execute("INSERT INTO StudentScores VALUES ('Bob', 'Science', 55)")

c.execute("INSERT INTO StudentScores VALUES ('Charlie', 'Math', 35)")

c.execute("INSERT INTO StudentScores VALUES ('Daisy', 'English', 82)")

c.execute("INSERT INTO StudentScores VALUES ('Ethan', 'Science', 29)")

c.execute("INSERT INTO StudentScores VALUES ('Fiona', 'Math', 91)")

print("All Students:")

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

print(row)

c.execute("SELECT AVG(marks) FROM StudentScores")

avg = c.fetchone()[0]

print("\nAverage Marks:", round(avg, 2))

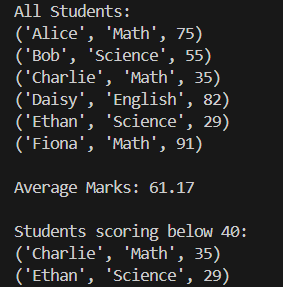
print("\nStudents scoring below 40:")

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

print(row)

conn.close()

**Output:**

****

## **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

git init

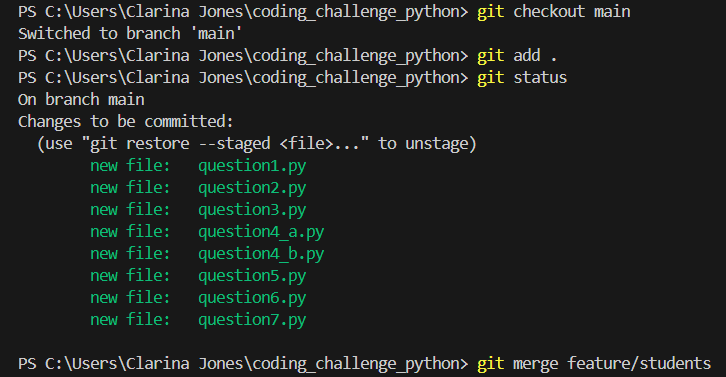
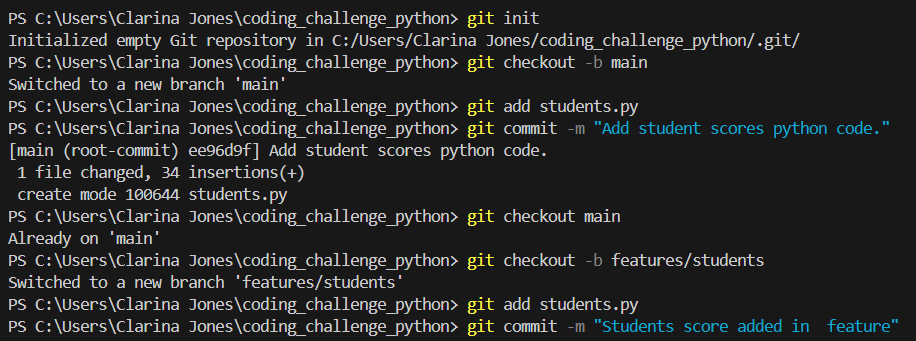
git checkout -b feature/students

git add students.py

git commit -m “Add student scores Python code”

git checkout 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

from unittest.mock import patch

from question1 import calculate\_total, book\_movie

class TestMovieBooking(unittest.TestCase):

def setUp(self):

print("\n[SETUP] Starting test...")

def tearDown(self):

print("[TEARDOWN] Test finished.")

def test\_calculate\_total(self):

self.assertEqual(calculate\_total(100, 2), 200)

@patch('builtins.input', side\_effect=['1', '2']) # Choose "Inception", 2 tickets

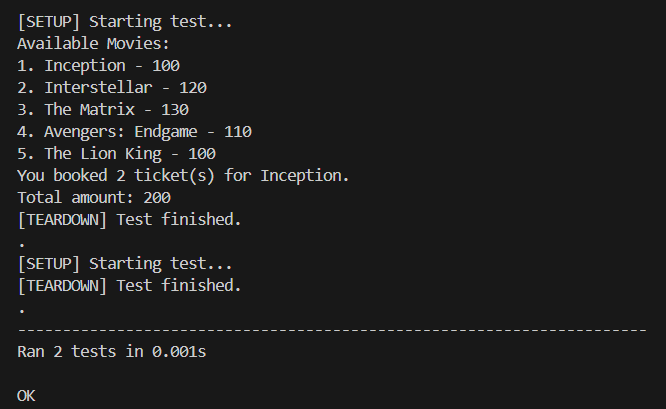
def test\_book\_movie\_with\_mock(self, mock\_inputs):

book\_movie()

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

unittest.main()

**Output:**

****