# DATE: 27/06/25

# 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

CODE:

movies = {

    "Avengers:Endgame": 550,

    "Inception": 320,

    "Interstellar": 280,

    "Dune": 200,

    "SpiderMan Homecoming":500,

    "Wakanda Forever":450,

    "Iron Man-3":500,

    "Avengers:Doomsday":1000,

    "Multiverse of Madness":400,

    "Hi Nanna":1000

}

def show\_movies():

    print("Available Movies:")

    for movie, price in movies.items():

        print(f"{movie} - ₹{price}")

def calculate\_amount(movie, tickets):

    return movies[movie] \* tickets

def book\_tickets():

    show\_movies()

    selected\_movie = input("Enter movie name to book: ").strip()

    if selected\_movie not in movies:

        print("Movie not available.🥲")

        return

    try:

        num\_tickets = int(input("Enter number of tickets: "))

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

        print(f"Booking successful!😍 Total amount: ₹{total}")

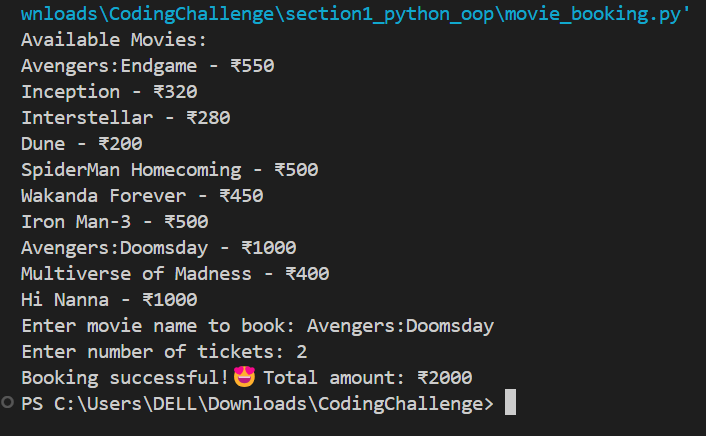
    except ValueError:

        print("Invalid ticket number.")

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

    book\_tickets()

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

    def \_\_str\_\_(self):

        return self.title

class Library:

    def \_\_init\_\_(self):

        self.books = []

    def add\_book(self, book):

        self.books.append(book)

    def display\_books(self):

        if not self.books:

            print("No books available.")

        else:

            print("Books in library:")

            for book in self.books:

                print(f"- {book}")

    def borrow\_book(self, title):

        for book in self.books:

            if book.title == title:

                self.books.remove(book)

                return book

        return None

    def return\_book(self, book):

        self.books.append(book)

class User:

    def \_\_init\_\_(self, name):

        self.name = name

        self.borrowed = []

    def borrow(self, library, title):

        book = library.borrow\_book(title)

        if book:

            self.borrowed.append(book)

            print(f"{self.name} borrowed '{title}'")

        else:

            print(f"'{title}' is not available.")

    def return\_book(self, library, title):

        for book in self.borrowed:

            if book.title == title:

                self.borrowed.remove(book)

                library.return\_book(book)

                print(f"{self.name} returned '{title}'")

                return

        print(f"{self.name} has not borrowed '{title}'.")

    def view\_borrowed(self):

        print(f"{self.name}'s borrowed books:")

        for book in self.borrowed:

            print(f"- {book}")

#  Manual Input Example

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

    lib = Library()

    lib.add\_book(Book("Python Basics"))

    lib.add\_book(Book("Machine Learning"))

    lib.add\_book(Book("Data Structures"))

    lib.add\_book(Book("AI for Beginners"))

    lib.add\_book(Book("Digital  Electronics"))

    name = input("Enter your name: ")

    user = User(name)

    lib.display\_books()

    titles\_to\_borrow = input("Enter book titles to borrow (comma-separated): ").split(',')

    for title in titles\_to\_borrow:

        user.borrow(lib, title.strip())

    user.view\_borrowed()

    lib.display\_books()

    titles\_to\_return = input("Enter book titles to return (comma-separated): ").split(',')

    for title in titles\_to\_return:

        user.return\_book(lib, title.strip())

    lib.display\_books()

    lib.display\_books()

    user.borrow(lib, "Python Basics")

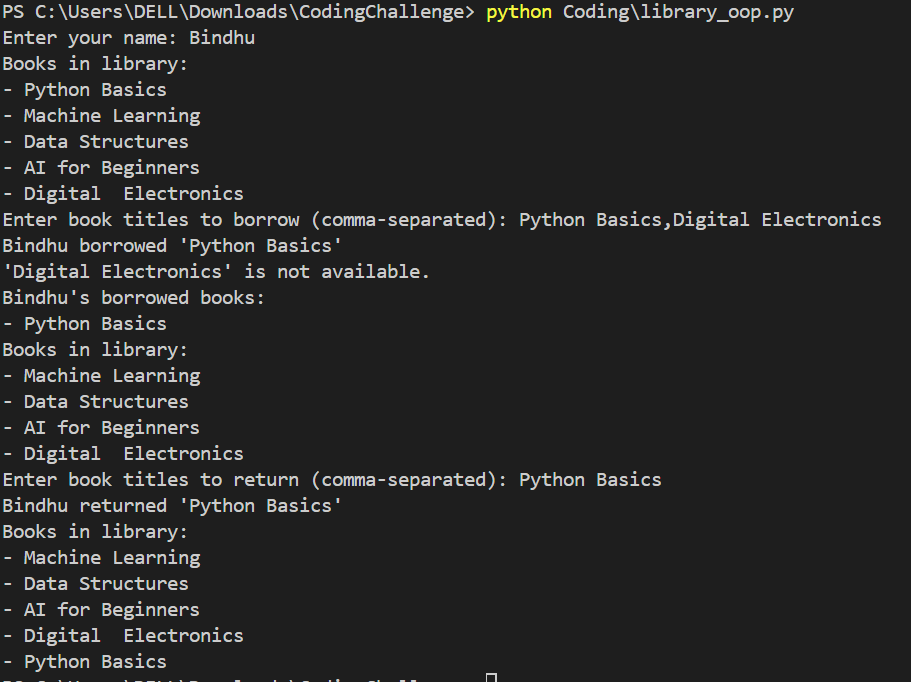
    user.view\_borrowed()

    lib.display\_books()

    user.return\_book(lib, "Python Basics")

    lib.display\_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]

CODE:

def min\_coins(amount):

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

    coins\_used = []

    for coin in denominations:

        while amount >= coin:

            amount -= coin

            coins\_used.append(coin)

    return coins\_used

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

    try:

        amt = int(input("Enter amount: ₹"))

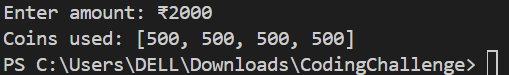
        result = min\_coins(amt)

        print("Coins used:", result)

    except ValueError:

        print("Invalid input. Please enter a number.")

OUTPUT:



Q4. Data Structure Usage (15 mins)  
- Stack: Evaluate postfix expression '231\*+9-'  
- Linked List class: append(), display(), reverse()

CODE:

# Postfix Evaluation using Stack

def eval\_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))  # integer division

    return stack[0]

# Linked List Implementation

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(data)

        if not self.head:

            self.head = new

        else:

            current = self.head

            while current.next:

                current = current.next

            current.next = new

    def display(self):

        current = self.head

        while current:

            print(current.data, end=" → ")

            current = current.next

        print("None")

    def reverse(self):

        prev = None

        curr = self.head

        while curr:

            next\_node = curr.next

            curr.next = prev

            prev = curr

            curr = next\_node

        self.head = prev

# Test Both

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

    # Postfix test

    expression = "231\*+9-"

    print(f"Postfix '{expression}' evaluates to:", eval\_postfix(expression))

    # Linked List test

    ll = LinkedList()

    for item in [1, 2, 3, 4]:

        ll.append(item)

    print("Original Linked List:")

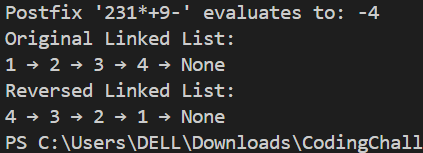
    ll.display()

    ll.reverse()

    print("Reversed Linked 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

CODE:

import mysql.connector

def connect():

    return mysql.connector.connect(

        host='localhost',

        user='root',

        password='Bindhusha@157',

        database='testdb',

        auth\_plugin='mysql\_native\_password'  # Ensures Python can authenticate with MySQL

    )

def create\_table(cursor):

    cursor.execute("DROP TABLE IF EXISTS StudentScores")

    cursor.execute("""

        CREATE TABLE StudentScores (

            name VARCHAR(50),

            subject VARCHAR(50),

            marks INT

        )

    """)

def insert\_data(cursor):

    data = [

        ("Anu", "Math", 90),

        ("Bindhu", "Science", 90),

        ("Rishi", "English", 95),

        ("Sindhu", "Hindi", 98),

  ("Sindhuja","Social",38)

    ]

    cursor.executemany("INSERT INTO StudentScores (name, subject, marks) VALUES (%s, %s, %s)", data)

def show\_all(cursor):

    cursor.execute("SELECT \* FROM StudentScores")

    print(" All Records:")

    for row in cursor.fetchall():

        print(row)

def show\_average(cursor):

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

    avg = cursor.fetchone()[0]

    print(f"\nAverage Marks: {avg:.2f}")

def show\_low\_scores(cursor):

    cursor.execute("SELECT name FROM StudentScores WHERE marks < 40")

    print("\nStudents Scoring < 40:")

    rows = cursor.fetchall()

    if rows:

        for row in rows:

            print(row[0])

    else:

        print("None")

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

    try:

        conn = connect()

        cur = conn.cursor()

        create\_table(cur)

        insert\_data(cur)

        conn.commit()

        show\_all(cur)

        show\_average(cur)

        show\_low\_scores(cur)

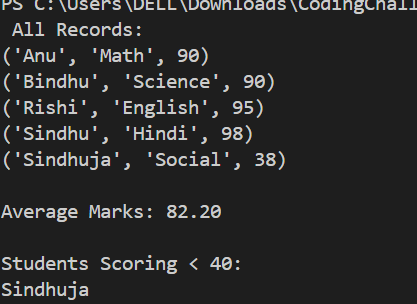
        cur.close()

        conn.close()

    except mysql.connector.Error as err:

        print(f"MySQL Error: {err}")

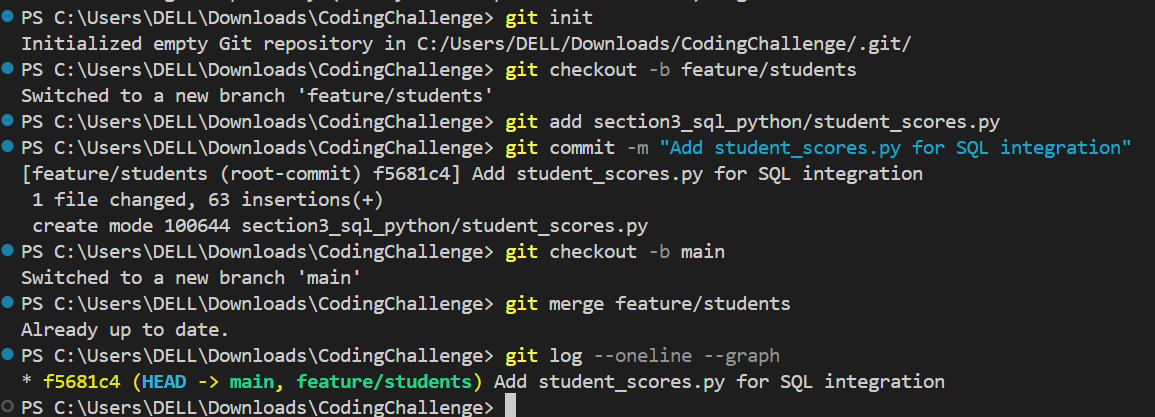
OUTPUT:



## 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

CODE AND OUTPUT:



## 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:

# test\_movie\_booking.py

import unittest

from unittest.mock import patch

from movie\_booking import calculate\_amount, book\_tickets

class TestMovieBooking(unittest.TestCase):

    def setUp(self):

        print("\n Setting up test")

    def tearDown(self):

        print(" Cleaning up test")

    def test\_calculate\_amount(self):

        amount = calculate\_amount("Inception", 2)

        self.assertEqual(amount, 640)

    @patch("builtins.input", side\_effect=["Inception", "2"])

    @patch("builtins.print")

    def test\_book\_tickets\_mock(self, mock\_print, mock\_input):

        book\_tickets()

        mock\_print.assert\_any\_call("Booking successful!😍 Total amount: ₹640")

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

    unittest.main()

OUTPUT:

