# 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 ={

"Batman": 30,

"Superman": 25,

"Spiderman": 20,

"Avengers": 35,

"Fight Club": 15,

"Matrix": 40,

"Interstellar": 50,

}

def get\_movies():

    print("Available Movies: ")

    print("-----------------")

    for movie, price in movies.items():

        print(f"{movie}: Rs.{price}")

    print("-----------------")

def book\_movie():

    ticket\_count = int(input("Enter number of tickets you want to book: "))

    movie\_name = input("Enter the movie choice you want to book: ")

    if ticket\_count == 1:

        tot = calculate\_total\_price(ticket\_count, movie\_name)

        if movie\_name in movies:

            print(f"You have booked {movie\_name} for Rs.{tot}")

        else:

            print("Invalid movie choice. Please try again.")

    elif ticket\_count > 1:

        if movie\_name in movies:

            tot = calculate\_total\_price(ticket\_count, movie\_name)

            print(f"You have booked {ticket\_count} tickets for {movie\_name} for Rs.{tot}")

        else:

            print("Invalid movie choice. Please try again.")

def calculate\_total\_price(ticket\_count, movie\_name):

    for movie, price in movies.items():

        if movie == movie\_name:

            total\_price = ticket\_count \* price

            return total\_price

def main():

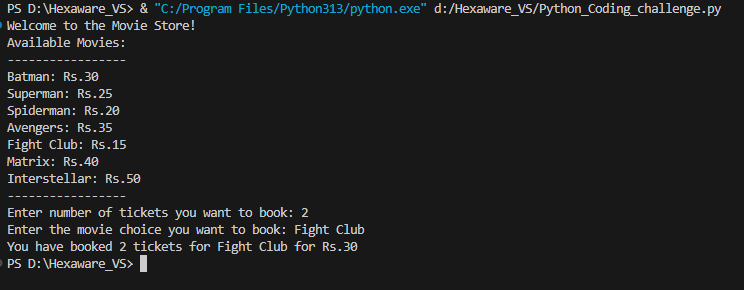
    print("Welcome to the Movie Store!")

    get\_movies()

    book\_movie()

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

    main()



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):

        self.title = title

        self.available = True

class Library:

    def \_\_init\_\_(self):

        self.books = [

            Book("Python Basics"),

            Book("OOP in Python"),

            Book("Data Structures"),

            Book("Machine Learning")

        ]

    def show\_books(self):

        print("\n Available Books:")

        for book in self.books:

            status = "Available" if book.available else "Borrowed"

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

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

        for book in self.books:

            if book.title == book\_title and book.available:

                book.available = False

                return f" You borrowed '{book.title}'"

        return " Book not available."

    def return\_book(self, book\_title):

        for book in self.books:

            if book.title == book\_title and not book.available:

                book.available = True

                return f" You returned '{book.title}'"

        return " Book was not borrowed or doesn't exist."

class User:

    def \_\_init\_\_(self, name):

        self.name = name

    def view\_books(self, library):

        library.show\_books()

    def borrow(self, library, book\_title):

        print(library.borrow\_book(book\_title))

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

        print(library.return\_book(book\_title))

# Example usage

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

    lib = Library()

    user = User("Mathew")

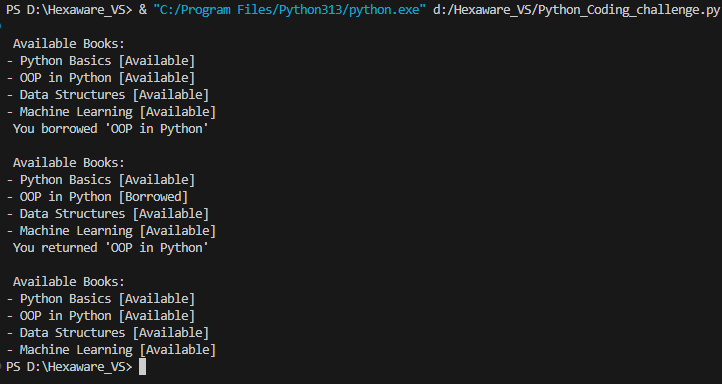
    user.view\_books(lib)

    user.borrow(lib, "OOP in Python")

    user.view\_books(lib)

    user.return\_book(lib, "OOP in Python")

    user.view\_books(lib)



## 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]  # Descending order for greedy

    result = []

    for coin in coins:

        while amount >= coin:

            amount -= coin

            result.append(coin)

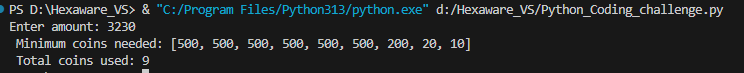
    return result

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

selected\_coins = minimize\_coins(amount)

print(" Minimum coins needed:", selected\_coins)

print(" Total coins used:", len(selected\_coins))



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

def evaluate\_postfix(expr):

    stack = []

    for char in expr:

        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(int(a / b))

    return stack[0]

expr = "231\*+9-"

print(" Postfix Result:", evaluate\_postfix(expr))



## 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 mysql.connector

def connect\_to\_database():

    try:

        conn = mysql.connector.connect(

            host="localhost",

            user="root",

            password="root",

            database="test\_db"

        )

        return conn

    except mysql.connector.Error as err:

        print(f"Error: {err}")

        return None

def create\_table(conn):

    cursor = conn.cursor()

    cursor.execute("""

        CREATE TABLE IF NOT EXISTS StudentScore (

            id INT AUTO\_INCREMENT PRIMARY KEY,

            name VARCHAR(50),

            subject VARCHAR(50),

            marks INT

        )

    """)

    conn.commit()

    cursor.close()

    print("Table created successfully.")

def insert\_data(conn, name, subject, marks):

    cursor = conn.cursor()

    cursor.execute(

        "INSERT INTO StudentScore (name, subject, marks) VALUES (%s, %s, %s)",

        (name, subject, marks)

    )

    conn.commit()

    cursor.close()

    print("Data inserted successfully.")

def fetch\_data(conn):

    cursor = conn.cursor()

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

    rows = cursor.fetchall()

    print("\nStudent Records:")

    for row in rows:

        print(f"ID: {row[0]}, Name: {row[1]}, Subject: {row[2]}, Marks: {row[3]}")

    cursor.close()

def show\_average\_marks(conn):

    cursor = conn.cursor()

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

    avg = cursor.fetchone()[0]

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

    cursor.close()

def students\_below\_40(conn):

    cursor = conn.cursor()

    cursor.execute("SELECT name, subject, marks FROM StudentScore WHERE marks < 40")

    rows = cursor.fetchall()

    print("\nStudents Scoring Below 40:")

    for row in rows:

        print(f"Name: {row[0]}, Subject: {row[1]}, Marks: {row[2]}")

    cursor.close()

def main():

    conn = connect\_to\_database()

    if conn:

        create\_table(conn)

        # You can comment this if data already inserted

        insert\_data(conn, "John Doe", "Math", 85)

        insert\_data(conn, "Jane Smith", "Science", 90)

        insert\_data(conn, "Sam Low", "Math", 38)

        insert\_data(conn, "Nina Gray", "English", 29)

        fetch\_data(conn)

        show\_average\_marks(conn)

        students\_below\_40(conn)

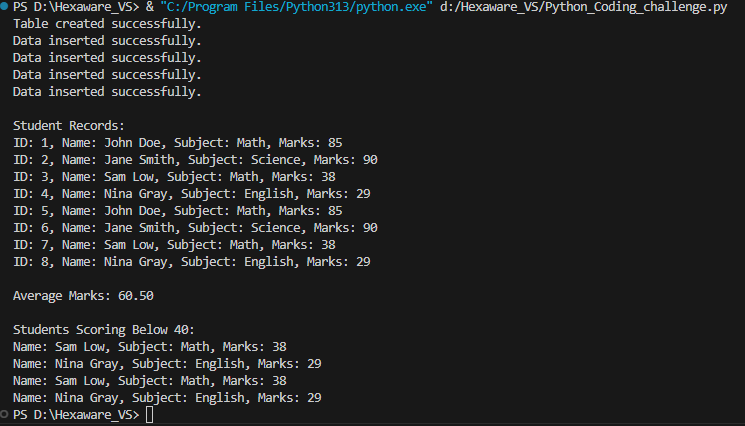
        conn.close()

    else:

        print("Failed to connect to the database.")

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

    main()

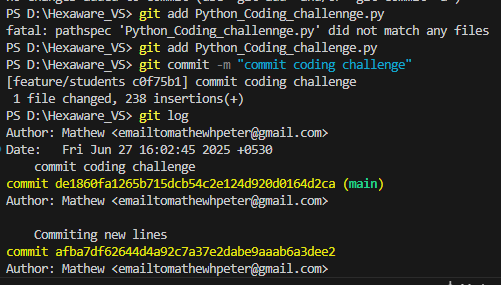


## Section 4: Version Control with Git (10 mins)

Q6. Git Challenge  
- Initialize Git repository

  
- Create and switch to branch feature/students

  
- Add and commit your Python code



- Merge feature/students into main

## 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 io import StringIO

from Python\_Coding\_challenge import calculate\_total\_price, book\_movie

class TestMovieBookingSystem(unittest.TestCase):

    def setUp(self):

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

    def tearDown(self):

        print("[Teardown] Finished test.")

    def test\_calculate\_total\_price(self):

        result = calculate\_total\_price(2, "Batman")

        self.assertEqual(result, 60)

        result = calculate\_total\_price(3, "UnknownMovie")

        self.assertIsNone(result)

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

    @patch("builtins.print")

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

        from Python\_Coding\_challenge import book\_movie

        book\_movie()

        mock\_print.assert\_any\_call("You have booked 2 tickets for Superman for Rs.50")

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

    unittest.main()

