# Coding Challenge

-Bavatharani S

**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 = ["Thalapathi", "Amaran", "LuckyBasker", "Kubera"]

ticket\_price = {

    "Thalapathi": 280,

    "Amaran": 350,

    "LuckyBasker": 300,

    "Kubera": 200

}

def show\_movies():

    print("Available movies:")

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

        print(f"{idx}. {movie} - Rs. {ticket\_price[movie]}")

def calculate\_amount(movie, tickets):

    return ticket\_price[movie] \* tickets

def book\_movie():

    show\_movies()

    try:

        choice = int(input("\nEnter movie number (1-4): "))

        if 1 <= choice <= len(movies):

            selected\_movie = movies[choice-1]

            tickets = int(input("Number of tickets: "))

            if tickets > 0:

                total = calculate\_amount(selected\_movie, tickets)

                print("\nBooking Confirmed!")

                print(f"Movie: {selected\_movie}")

                print(f"Tickets: {tickets}")

                print(f"Total: Rs. {total}")

            else:

                print("Invalid ticket quantity")

        else:

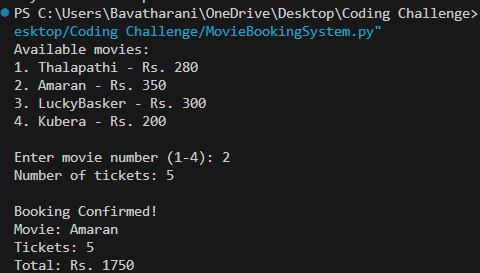
            print("Invalid movie selection")

    except ValueError:

        print("Please enter valid numbers")

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, author, year):

        self.title = title

        self.author = author

        self.year = year

        self.is\_borrowed = False

    def \_\_str\_\_(self):

        status = "(Borrowed)" if self.is\_borrowed else "(Available)"

        return f"{self.title} by {self.author} ({self.year}) {status}"

class Library:

    def \_\_init\_\_(self):

        self.books = []

    def add\_book(self, book):

        self.books.append(book)

    def view\_books(self):

        print("\nBooks in library:")

        for book in self.books:

            print(" -", book)

    def borrow\_book(self, title):

        for book in self.books:

            if book.title.lower() == title.lower() and not book.is\_borrowed:

                book.is\_borrowed = True

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

                return

        print("Book is not available.")

    def return\_book(self, title):

        for book in self.books:

            if book.title.lower() == title.lower() and book.is\_borrowed:

                book.is\_borrowed = False

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

                return

        print("Invalid return attempt.")

class User:

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

        self.name = name

        self.library = library

    def view\_books(self):

        self.library.view\_books()

    def borrow\_book(self, title):

        self.library.borrow\_book(title)

    def return\_book(self, title):

        self.library.return\_book(title)

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

    lib = Library()

    lib.add\_book(Book("To Kill a Mockingbird", "Harper Lee", 1960))

    lib.add\_book(Book("The Great Gatsby", "F. Scott Fitzgerald", 1925))

    lib.add\_book(Book("Atomic Habits", "James Clear", 2018))

    lib.add\_book(Book("Sapiens", "Yuval Noah Harari", 2011))

    user = User("John", lib)

  print("\n=== Initial Library ===")

    user.view\_books()

    print("\n=== Borrowing a Book ===")

    user.borrow\_book("The Great Gatsby")

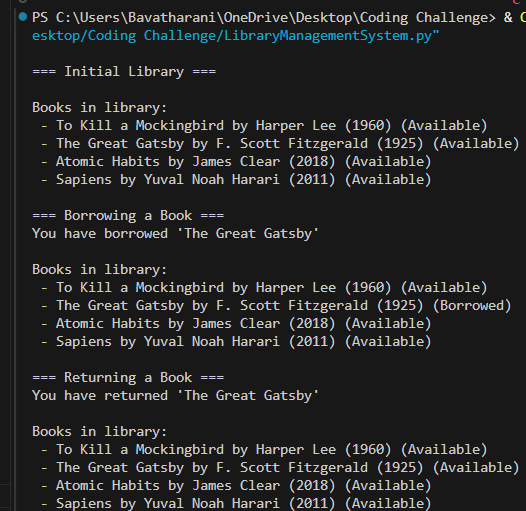
    user.view\_books()

    print("\n=== Returning a Book ===")

    user.return\_book("The Great Gatsby")

    user.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]

**Code:**

def minimize\_coins(amount):

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

    coins = {}

    for coin in denominations:

        if amount >= coin:

            count = amount // coin

            coins[coin] = count

            amount -= count \* coin

        if amount == 0:

            break

    total\_coins = sum(coins.values())

    return total\_coins, coins

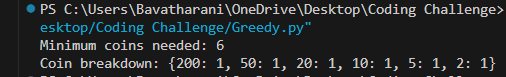
amount = 287

total, coins = minimize\_coins(amount)

print(f"Minimum coins needed: {total}")

print("Coin breakdown:", coins)

**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:

            val1 = stack.pop()

            val2 = stack.pop()

            if char == '+':

                stack.append(val2 + val1)

            elif char == '-':

                stack.append(val2 - val1)

            elif char == '\*':

                stack.append(val2 \* val1)

            elif char == '/':

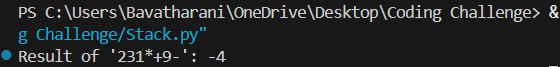
                stack.append(val2 / val1)

    return stack.pop()

result = evaluate\_postfix('231\*+9-')

print("Result of '231\*+9-':", result)

**Output:**

****

- 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

        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

ll = LinkedList()

ll.append(1)

ll.append(2)

ll.append(3)

print("Original list:")

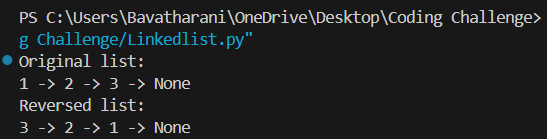
ll.display()

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

**Code:**

import mysql.connector

from mysql.connector import Error

def create\_connection():

    try:

        connection = mysql.connector.connect(

            host='localhost',

            user='root',

            password='Tharani@25'

        )

        print("MySQL connection established")

        return connection

    except Error as e:

        print(f"Error connecting to MySQL: {e}")

        return None

def setup\_database(connection):

    try:

        cursor = connection.cursor()

        cursor.execute("CREATE DATABASE IF NOT EXISTS SchoolDB")

        cursor.execute("USE SchoolDB")

        cursor.execute("""

        CREATE TABLE IF NOT EXISTS StudentScores (

            name VARCHAR(50),

            subject VARCHAR(50),

            marks INT,

            PRIMARY KEY (name, subject)

        )

        """)

        sample\_data = [

            ('Alice', 'Math', 85),

            ('Alice', 'Science', 78),

            ('Bob', 'Math', 92),

            ('Bob', 'Science', 35),

            ('Charlie', 'Math', 42),

            ('Charlie', 'Science', 55)

        ]

        cursor.executemany("""

        INSERT INTO StudentScores (name, subject, marks)

        VALUES (%s, %s, %s)

        ON DUPLICATE KEY UPDATE marks=VALUES(marks)

        """, sample\_data)

        connection.commit()

        print("Database and table created successfully with sample data")

    except Error as e:

        print(f"Error setting up database: {e}")

def display\_records(connection):

    try:

        cursor = connection.cursor()

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

        records = cursor.fetchall()

        print("\nAll Student Records:")

        print("{:<10} {:<10} {:<5}".format("Name", "Subject", "Marks"))

        print("-"\*30)

        for name, subject, marks in records:

            print("{:<10} {:<10} {:<5}".format(name, subject, marks))

    except Error as e:

        print(f"Error fetching records: {e}")

def show\_average\_marks(connection):

    try:

        cursor = connection.cursor()

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

        avg = cursor.fetchone()[0]

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

    except Error as e:

        print(f"Error calculating average: {e}")

def show\_students\_below\_40(connection):

    try:

        cursor = connection.cursor()

        cursor.execute("""

        SELECT DISTINCT name

        FROM StudentScores

        WHERE marks < 40

        """)

        students = cursor.fetchall()

        print("\nStudents scoring below 40:")

        if students:

            for (name,) in students:

                print(name)

        else:

            print("No students scored below 40")

    except Error as e:

        print(f"Error fetching students: {e}")

def main():

    connection = create\_connection()

    if connection:

        try:

            setup\_database(connection)

            while True:

                print("\n===== Student Scores Menu =====")

                print("1. Display all records")

                print("2. Show average marks")

                print("3. List students scoring below 40")

                print("4. Exit")

                choice = input("Enter your choice (1-4): ")

                if choice == "1":

                    display\_records(connection)

                elif choice == "2":

                    show\_average\_marks(connection)

                elif choice == "3":

                    show\_students\_below\_40(connection)

                elif choice == "4":

                    print("Exiting program...")

                    break

                else:

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

        finally:

            if connection.is\_connected():

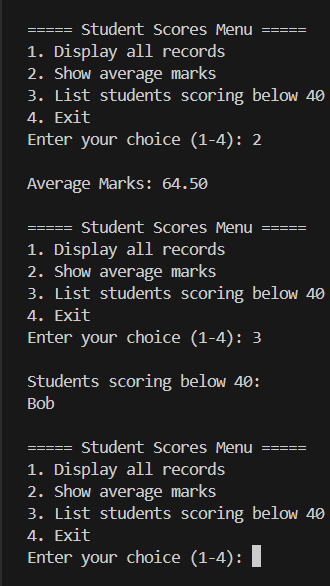
                connection.close()

                print("MySQL connection closed")

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

    main()

**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

**Commands**

Git init – initialize git repository

git branch feature/students

git checkout feature/students

git add .

git commit -m "Add student"

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()

**Code:**

import unittest

from unittest.mock import patch

from io import StringIO

import sys

from movie\_booking import calculate\_amount, book\_movie # if code is in movie\_booking.py

class TestBooking(unittest.TestCase):

def test\_amount(self):

self.assertEqual(calculate\_amount("Amaran", 2), 700)

@patch('builtins.input', side\_effect=["2", "3"])

def test\_success(self, mock\_input):

out = StringIO(); sys.stdout = out

book\_movie(); sys.stdout = sys.\_\_stdout\_\_

self.assertIn("Booking Confirmed!", out.getvalue())

@patch('builtins.input', side\_effect=["5"])

def test\_invalid\_movie(self, m):

out = StringIO(); sys.stdout = out

book\_movie(); sys.stdout = sys.\_\_stdout\_\_

self.assertIn("Invalid movie selection", out.getvalue())

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

unittest.main()

**Output:**

