# **Coding Challenge**

## **Section 1: Python Programming & OOP (40 mins)**

**Q1. Functional Coding Challenge – Movie Booking System  
 - 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**

**Answers**

movie\_list = ["K3G", "I", "INTERSTELLAR"]

movie\_prices = {

"K3G": 10,

"I": 12,

"INTERSTELLAR": 8

}

def show\_movies():

print("Available Movies:")

for movie in movie\_list:

print(movie)

def calculate\_total(movie\_name, tickets):

price = movie\_prices[movie\_name]

total = price \* tickets

return total

def book\_movie():

show\_movies()

selected\_movie = input("Enter the name of the movie you want to book: ")

if selected\_movie in movie\_list:

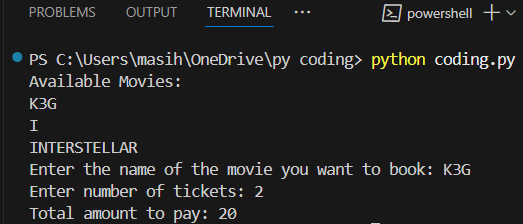
num\_tickets = int(input("Enter number of tickets: "))

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

print("Total amount to pay: " + str(total\_amount))

else:

print("Sorry, that movie is not available.")

book\_movie()

**OUTPUT :**

**Q2. OOP Implementation – Library Management  
 - Create classes Book, Library, and User  
 - Library contains a collection of books  
 - User can borrow/return/view books  
 - Use class, constructor, inheritance, method overriding**

**ANSWER**

class Book:

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

self.title = title

self.author = author

self.is\_borrowed = False

def \_\_str\_\_(self):

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

return f"{self.title} by {self.author} - {status}"

class Library:

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

self.books = []

def add\_book(self, book):

self.books.append(book)

def show\_books(self):

print("\nLibrary Books:")

for book in self.books:

print(book)

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 borrowed '{book.title}'")

return

print("Book not available.")

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 returned '{book.title}'")

return

print("Book not found or not borrowed.")

class User(Library):

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

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

self.name = name

def show\_books(self):

print(f"\n{self.name}'s View of Library:")

for book in self.books:

if not book.is\_borrowed:

print(book.title)

my\_library = Library()

my\_library.add\_book(Book("Harry Potter and the Sorcerer's Stone", "J.K. Rowling"))

my\_library.add\_book(Book("Twilight", "Stephenie Meyer"))

my\_library.add\_book(Book("The Curse of the Cheese Pyramid", "Geronimo Stilton")

user = User("Sheethal")

user.books = my\_library.books

user.show\_books()

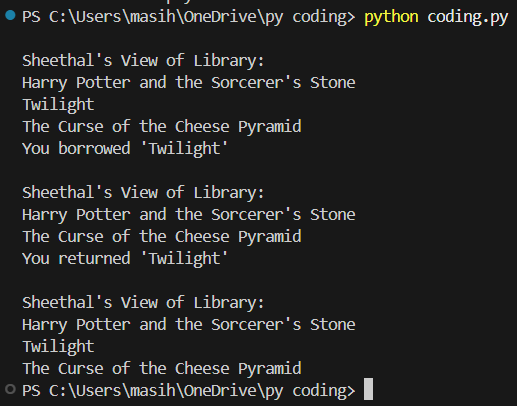
user.borrow\_book("Twilight")

user.show\_books()

user.return\_book("Twilight")

user.show\_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]**

**ANSWERS:**

def minimize\_coins(amount):

denominations = [500, 200, 100, 50, 20, 10, 5, 2, 1] # Sorted high to low

result = []

for coin in denominations:

while amount >= coin:

amount -= coin

result.append(coin)

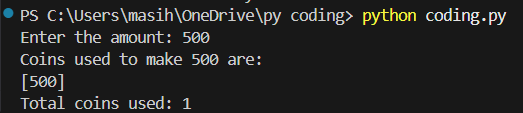
return result

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

coins\_used = minimize\_coins(amount)

print("Coins used to make", amount, "are:")

print(coins\_used)

print("Total coins used:", len(coins\_used))

**OUTPUT :**

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

**ANSWERS:**

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 == '+':

result = a + b

elif char == '-':

result = a - b

elif char == '\*':

result = a \* b

elif char == '/':

result = a // b # integer division

stack.append(result)

return stack[0]

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 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

def display(self):

temp = self.head

while temp is not None:

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

temp = temp.next

print("None")

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

expression = '231\*+9-'

print("Postfix Expression:", expression)

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

ll = LinkedList()

ll.append(10)

ll.append(20)

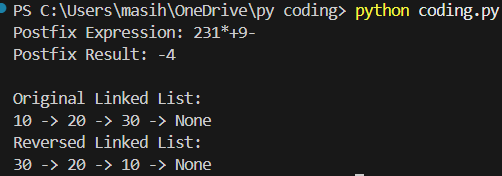
ll.append(30)

print("\nOriginal 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**

**ANSWER :**

import sqlite3

conn = sqlite3.connect("students.db")

cursor = conn.cursor()

cursor.execute('''

CREATE TABLE IF NOT EXISTS StudentScores (

name TEXT,

subject TEXT,

marks INTEGER

)

''')

cursor.execute("DELETE FROM StudentScores") # Clear table if already exists

sample\_data = [

('Aarav', 'Maths', 88),

('Riya', 'Science', 76),

('Kabir', 'English', 35),

('Ananya', 'Maths', 91),

('Vivaan', 'Science', 29),

('Meera', 'English', 47),

('Rohan', 'Maths', 38)

]

cursor.executemany("INSERT INTO StudentScores (name, subject, marks) VALUES (?, ?, ?)", sample\_data)

conn.commit()

print("All Student Records:")

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

for row in cursor.fetchall():

print(row)

print("\nAverage Marks:")

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

avg = cursor.fetchone()[0]

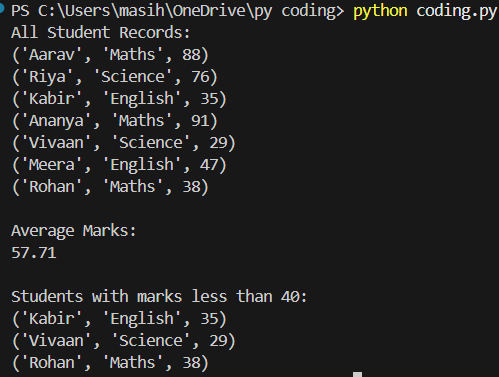
print(f"{avg:.2f}")

print("\nStudents with marks less than 40:")

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

for row in cursor.fetchall():

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**

**ANSWERS :**

git init

git checkout -b feature/students

git add .

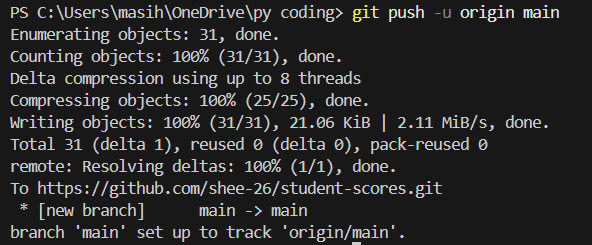
git commit -m "Add student scores program"

git checkout -b main

git merge feature/students

git branch -d feature/students

git remote add origin https://github.com/shee-26/student-scores.git

git push -u origin main

**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()**

**ANSWERS :**

movies = {"Harry Potter": 150, "Twilight": 120, "Geronimo Stilton": 100}

def calculate\_amount(movie, tickets):

return movies[movie] \* tickets

def booking(movie, tickets):

if movie in movies and tickets > 0:

return f"Booked {tickets} tickets for {movie}"

else:

return "Booking failed"

import unittest

class TestMovieBookingSystem(unittest.TestCase):

def setUp(self):

print("Setting up test environment...")

def tearDown(self):

print("Cleaning up after test...")

def test\_calculate\_amount(self):

result = calculate\_amount("Harry Potter", 2)

self.assertEqual(result, 300)

def test\_booking\_valid(self):

result = booking("Twilight", 3)

self.assertEqual(result, "Booked 3 tickets for Twilight")

def test\_booking\_invalid(self):

result = booking("Unknown", 2)

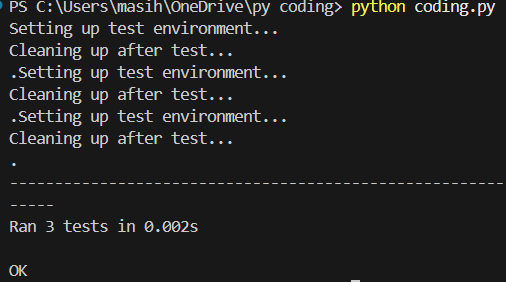
self.assertEqual(result, "Booking failed")

result = booking("Harry Potter", 0)

self.assertEqual(result, "Booking failed")

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

unittest.main()



**OUTPUT ;**