**CODING CHALLENGE PYTHON – 27/06/2025**

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

**Q1. Functional Coding Challenge – Movie Booking System (20 mins)**

# Movie Booking System

movies = {

"Inception": 150,

"Interstellar": 200,

"Oppenheimer": 180,

"Dunkirk": 120

}

def show\_movies():

print("\nAvailable Movies:")

for movie, price in movies.items():

print(f"{movie} - ₹{price} per ticket")

def calculate\_amount(movie, num\_tickets):

return movies[movie] \* num\_tickets

def book\_movie():

show\_movies()

movie = input("\nEnter the movie you want to book: ")

if movie not in movies:

print("Sorry, movie not available!")

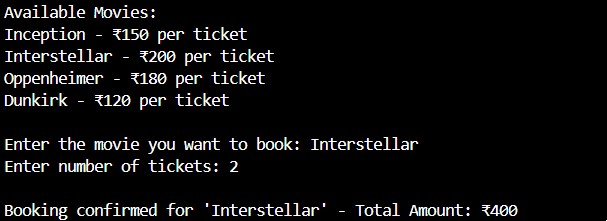
return

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

total = calculate\_amount(movie, num\_tickets)

print(f"\nBooking confirmed for '{movie}' - Total Amount: ₹{total}")

# book\_movie() # Uncomment to run interactively



**Q2. OOP Implementation – Library Management (20 mins)**

class Book:

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

self.title = title

self.available = True

class Library:

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

self.books = [Book("1984"), Book("Sapiens"), Book("Python Crash Course")]

def display\_books(self):

print("\nBooks in Library:")

for book in self.books:

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

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

def borrow\_book(self, title):

for book in self.books:

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

book.available = False

print(f"'{title}' issued successfully.")

return

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

def return\_book(self, title):

for book in self.books:

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

book.available = True

print(f"'{title}' returned successfully.")

return

print(f"Invalid return for '{title}'.")

class User(Library):

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

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

def request\_book(self):

title = input("Enter book to borrow: ")

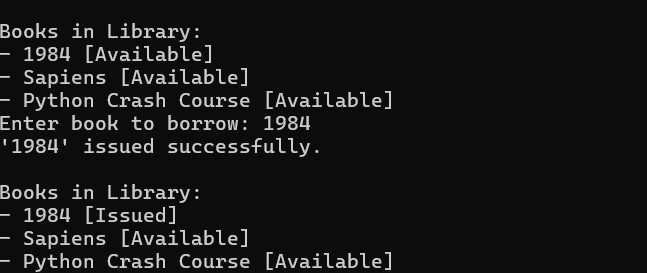
self.borrow\_book(title)

def return\_book\_user(self):

title = input("Enter book to return: ")

self.return\_book(title)

u = User(); u.display\_books(); u.request\_book(); u.display\_books()



**Section 2: Data Structures & Algorithms (30 mins)**

**Q3. Minimize Coins (Greedy Algorithm) (15 mins)**

def min\_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)

return result

print(min\_coins(93)) # Output: [50, 20, 20, 2, 1]



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

**a) *Evaluate postfix expression '231+9-' using Stack*\***

def eval\_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(a / b)

return stack[0]

print(eval\_postfix('231\*+9-'))



**b) Linked List: append(), display(), reverse()**

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:

temp = self.head

while temp.next:

temp = temp.next

temp.next = new

def display(self):

temp = self.head

while temp:

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

temp = temp.next

print("None")

def reverse(self):

prev = None

current = self.head

while current:

nxt = current.next

current.next = prev

prev = current

current = nxt

self.head = prev

l = LinkedList(); l.append(1); l.append(2); l.display(); l.reverse(); l.display()



**Section 3: SQL with Python Integration (30 mins)**

**Q5. SQL + Python – Student Scores Table**

import sqlite3

# Connect to DB

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

c = conn.cursor()

# Create Table

c.execute("CREATE TABLE IF NOT EXISTS StudentScores(name TEXT, subject TEXT, marks INTEGER)")

# Insert Sample Data

sample = [('Ava', 'Math', 78), ('John', 'Science', 35), ('Lily', 'Math', 90), ('Mark', 'History', 32)]

c.executemany("INSERT INTO StudentScores VALUES (?, ?, ?)", sample)

conn.commit()

# Show All Records

print("All Records:")

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

print(row)

# Show Average Marks

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

print("\nAverage Marks:", c.fetchone()[0])

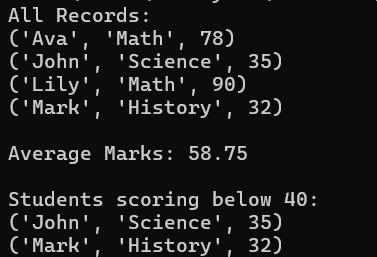
# Students scoring < 40

print("\nStudents scoring below 40:")

for row in c.execute("SELECT name, subject, marks FROM StudentScores WHERE marks < 40"):

print(row)

conn.close()



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

**Q6. Git Challenge – Commands**

# Initialize Git

git init

# Create and switch to new branch

git checkout -b feature/students

# Add and commit files

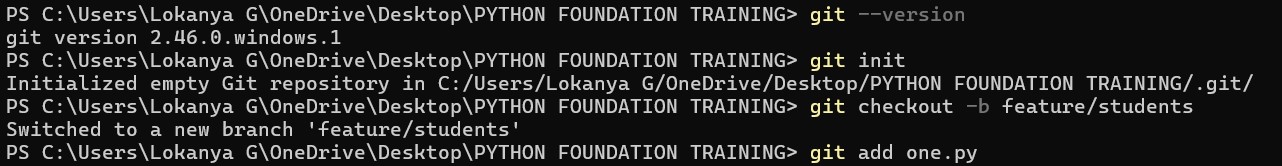
git add .

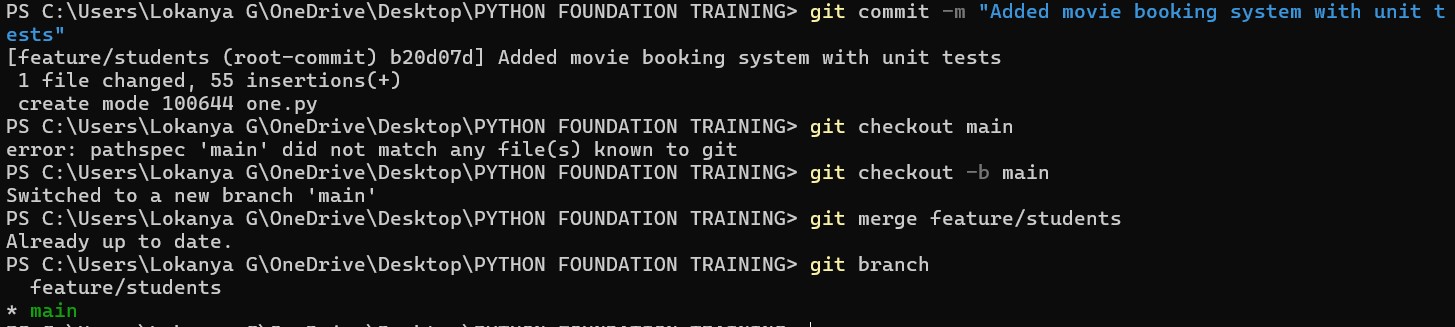
git commit -m "Added student score Python script"

# Switch to main and merge

git checkout main

git merge feature/students





**Bonus Section: PyUnit Test Case (10 mins)**

**Q7. PyUnit Test for Movie Booking System**

import unittest

from unittest.mock import patch

from io import StringIO

class TestBookingSystem(unittest.TestCase):

def setUp(self):

self.movies = {

"Inception": 150,

"Interstellar": 200

}

def test\_calculate\_amount(self):

from movie\_booking import calculate\_amount

self.assertEqual(calculate\_amount("Inception", 2), 300)

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

@patch('sys.stdout', new\_callable=StringIO)

def test\_booking(self, mock\_stdout, mock\_input):

from movie\_booking import book\_movie

book\_movie()

output = mock\_stdout.getvalue()

self.assertIn("Booking confirmed", output)

def tearDown(self):

pass

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

unittest.main()

