# 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

def show\_movies(movies):

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

print(f"{idx}. {movie} - ₹{movies[movie]}")

def calculate\_amount(movie, tickets, movies):

return movies[movie] \* tickets

def book\_tickets():

movies = {

'Interstellar': 150,

'Inception': 180,

'Dune': 200

}

show\_movies(movies)

choice = int(input("Select a movie (1-3): "))

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

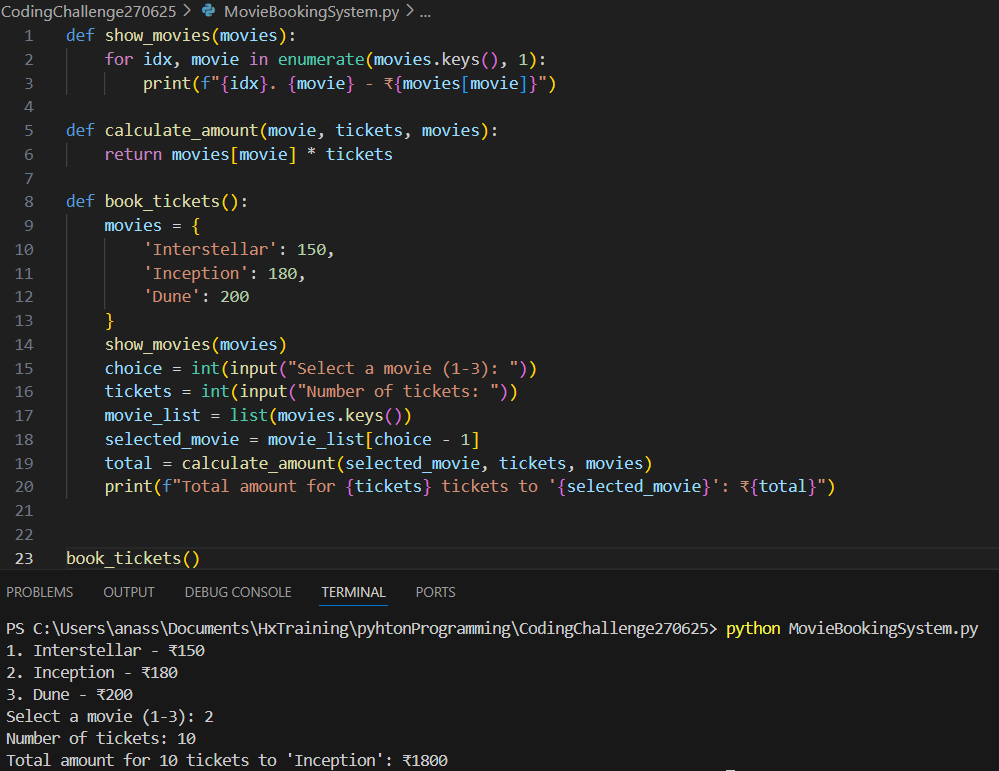
movie\_list = list(movies.keys())

selected\_movie = movie\_list[choice - 1]

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

print(f"Total amount for {tickets} tickets to '{selected\_movie}': ₹{total}")

book\_tickets()



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

class Library:

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

self.books = []

def add\_book(self, book):

self.books.append(book)

def show\_books(self):

for book in self.books:

print(book.title)

def remove\_book(self, book):

self.books.remove(book)

class User(Library):

def borrow\_book(self, title):

for book in self.books:

if book.title == title:

self.books.remove(book)

print(f"Borrowed: {title}")

return

print("Book not available.")

def return\_book(self, book):

self.books.append(book)

print(f"Returned: {book.title}")



## 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]

result = []

for coin in coins:

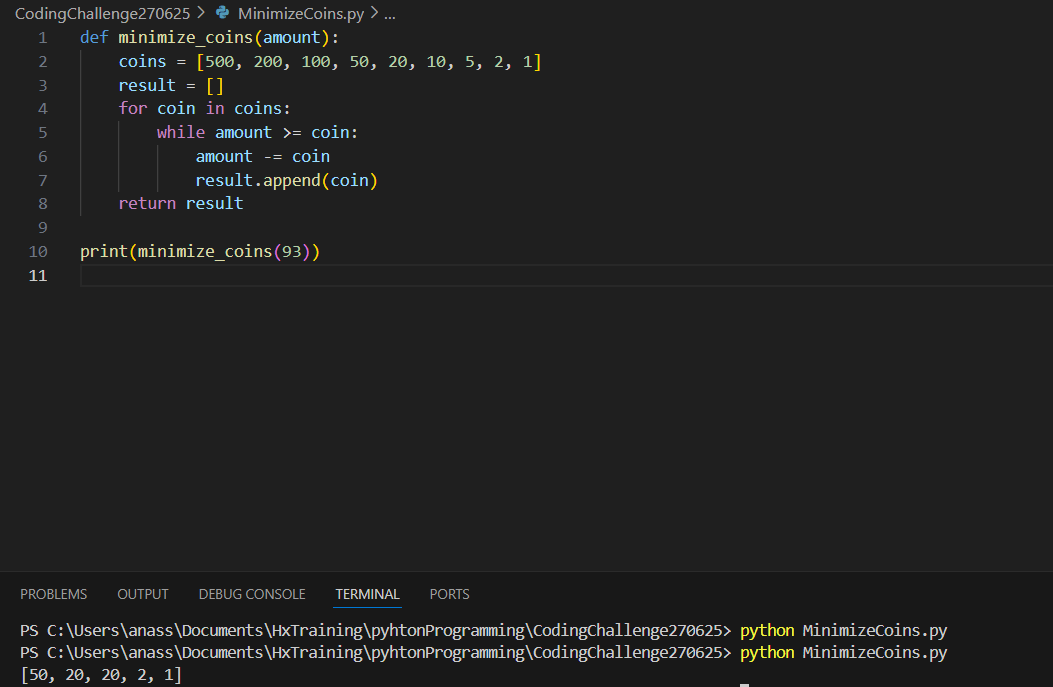
while amount >= coin:

amount -= coin

result.append(coin)

return result

print(minimize\_coins(93))



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

def evaluate\_postfix(expr):

stack = []

for ch in expr:

if ch.isdigit():

stack.append(int(ch))

else:

b = stack.pop()

a = stack.pop()

if ch == '+': stack.append(a + b)

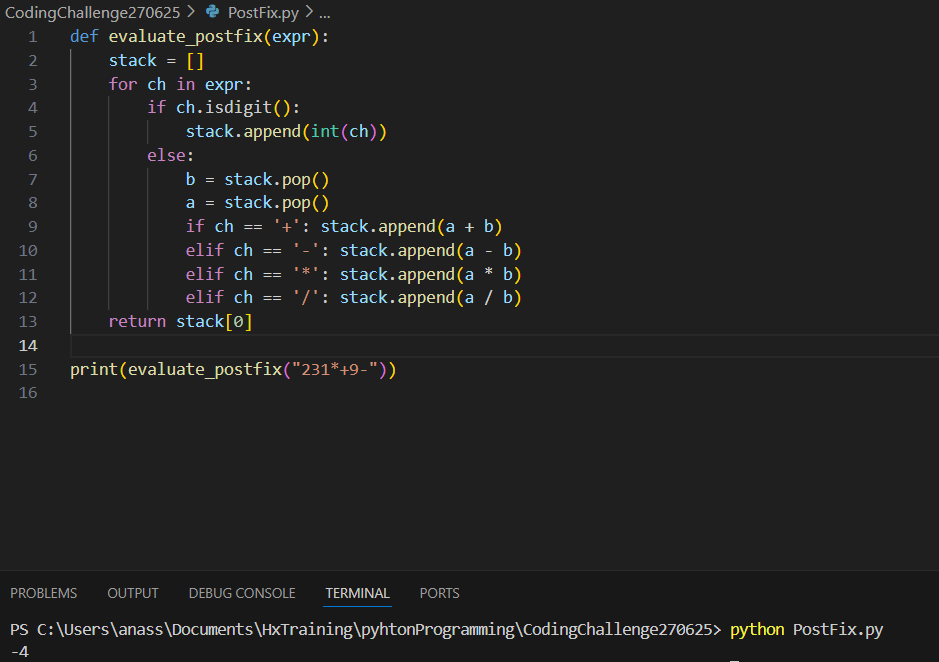
elif ch == '-': stack.append(a - b)

elif ch == '\*': stack.append(a \* b)

elif ch == '/': stack.append(a / b)

return stack[0]

print(evaluate\_postfix("231\*+9-"))



LINKED LIST

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def append(self, data):

if not self.head:

self.head = Node(data)

else:

temp = self.head

while temp.next:

temp = temp.next

temp.next = Node(data)

def display(self):

temp = self.head

while temp:

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

temp = temp.next

print("None")

def reverse(self):

prev, curr = None, self.head

while curr:

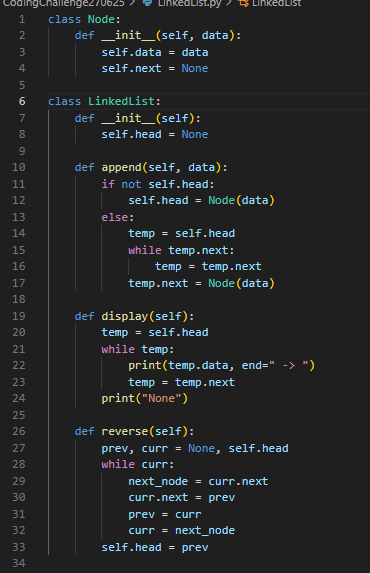
next\_node = curr.next

curr.next = prev

prev = curr

curr = next\_node

self.head = prev



## 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 sqlite3

def setup\_db():

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

c = conn.cursor()

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

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

('Alice', 'Math', 85),

('Bob', 'Science', 35),

('Carol', 'Math', 65),

('Dave', 'Science', 95),

])

conn.commit()

conn.close()

def show\_records():

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

c = conn.cursor()

print("All Records:")

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

print(row)

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

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

print("Students scoring < 40:")

for row in c.execute("SELECT \* FROM StudentScores WHERE marks < 40"):

print(row)

conn.close()

setup\_db()

show\_records()

## 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

git init

git checkout -b feature/students

git add .

git commit -m "Add movie booking and student scoring features"

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()

import unittest

from movie\_booking import calculate\_amount

class TestBookingSystem(unittest.TestCase):

def test\_calculate\_amount(self):

movies = {'Interstellar': 150, 'Dune': 200}

self.assertEqual(calculate\_amount('Interstellar', 2, movies), 300)

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

unittest.main()

