# 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

movie\_list = ["Amaran", "Leo", "Kaththi"]

movie\_prices = {

    "Amaran": 120,

    "Leo": 150,

    "Kaththi": 100

}

def show\_movies():

    print("\nAvailable Movies:")

    for movie in movie\_list:

        print(f"- {movie} (₹{movie\_prices[movie]})")

def calculate\_amount(movie, tickets):

    return movie\_prices[movie] \* tickets

def book\_movie():

    show\_movies()

    movie\_name = input("\nEnter the movie name: ")

    if movie\_name not in movie\_list:

        print("Invalid movie name.")

        return

    tickets = int(input("Enter number of tickets: "))

    if tickets <= 0:

        print("Invalid number of tickets.")

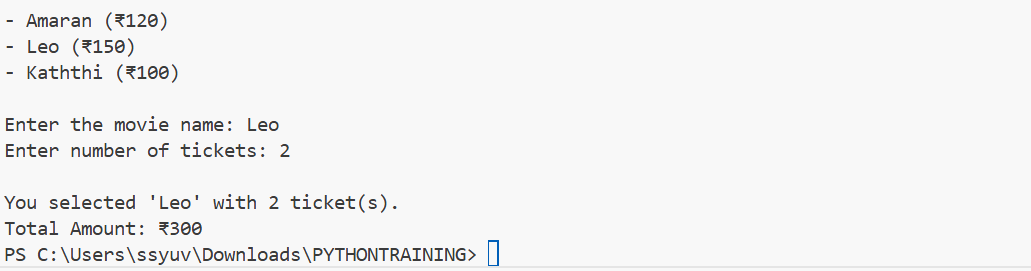
        return

    total = calculate\_amount(movie\_name, tickets)

    print(f"\nYou selected '{movie\_name}' with {tickets} ticket(s).")

    print(f"Total Amount: ₹{total}")

book\_movie()



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, author):

        self.title = title

        self.author = author

        self.is\_borrowed = False

    def \_\_str\_\_(self):

        status = "Not Available" 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 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"\nYou borrowed '{book.title}'")

                return

        print("\nBook not available or already borrowed.")

    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"\nYou returned '{book.title}'")

                return

        print("\nInvalid return or book was not borrowed.")

class User(Library):#inhertis from library

    def \_\_init\_\_(self, name):

        super().\_\_init\_\_()

        self.name = name

    def view\_books(self):#override

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

        super().view\_books()

library = Library()

library.add\_book(Book("Thirukural", "Thiruvalluvar"))

library.add\_book(Book("Gitanjali", "Rabindranath Tagore"))

library.add\_book(Book("Python Basics", "Guido van Rossum"))

user1 = User("Yuvasri")

user1.books = library.books

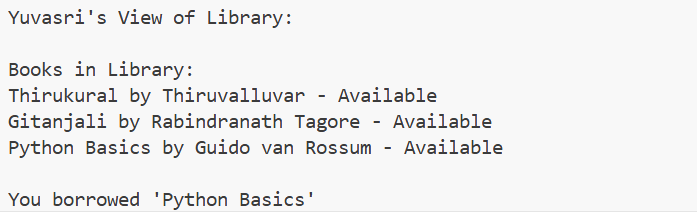
user1.view\_books()

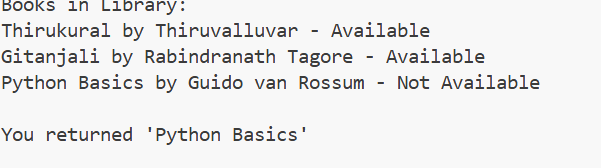
user1.borrow\_book("Python Basics")

user1.view\_books()

user1.return\_book("Python Basics")

user1.view\_books()





## 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):

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

    result = []

    for coin in denominations:

        while amount >= coin:

            amount -= coin

            result.append(coin)

    return result

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

coins\_used = minimize\_coins(amount)

print(f"Minimum coins needed: {len(coins\_used)}")

print("Coins used:", coins\_used)



Q4. Data Structure Usage (15 mins)  
- Stack: Evaluate postfix expression '231\*+9-'

def postfix(expression):

    stack = []

    for char in expression:

        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 == '/':

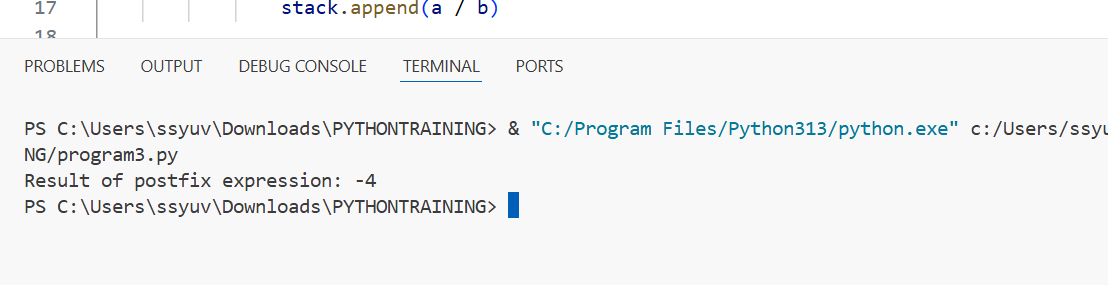
                stack.append(a / b)

    return stack[0]

expr = "231\*+9-"

result = postfix(expr)

print("Result of postfix expression:", result)

  
- Linked List class: 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 = Node(data)

        if not self.head:

            self.head = new\_node

            return

        temp = self.head

        while temp.next:

            temp = temp.next

        temp.next = new\_node

    def display(self):

        temp = self.head

        while temp:

            print(temp.data, end=' ')

            temp = temp.next

        print()

    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(71)

l.append(72)

l.append(56)

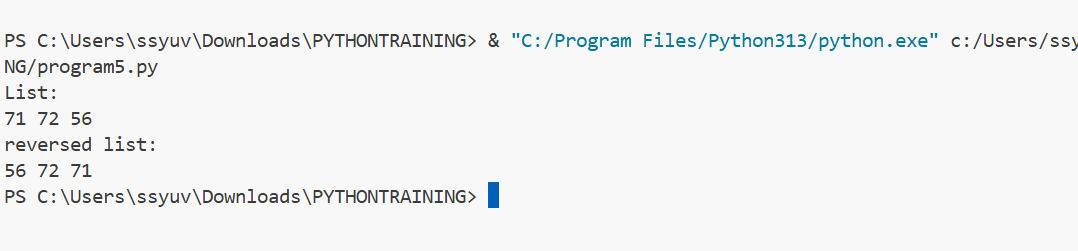
print("List:")

l.display()

l.reverse()

print("reversed list:")

l.display()



## 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

use hexaware;

create table studentscores (

name varchar(50),

subject varchar(50),

marks int

);

insert into studentscores values ('yuvasri', 'math', 85);

insert into studentscores values ('arun', 'math', 35);

insert into studentscores values ('divya', 'science', 42);

insert into studentscores values ('karan', 'science', 25);

insert into studentscores values ('meena', 'english', 75);

import mysql.connector

conn = mysql.connector.connect(

    host='localhost',

    user='root',

    password='Yuvasri@310\*',

    database='hexaware'

)

cursor = conn.cursor()

print("All records:")

cursor.execute("select \* from studentscores")

for row in cursor.fetchall():

    print(row)

cursor.execute("select avg(marks) from studentscores")

avg = cursor.fetchone()[0]

print("average marks:", avg)

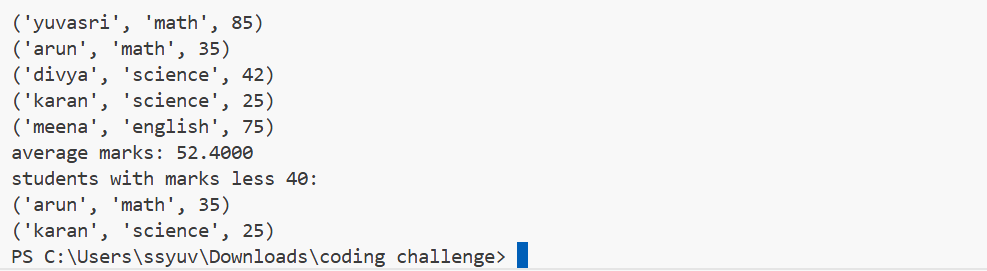
print("students with marks less 40:")

cursor.execute("select name, subject, marks from studentscores where marks < 40")

for row in cursor.fetchall():

    print(row)

conn.close()



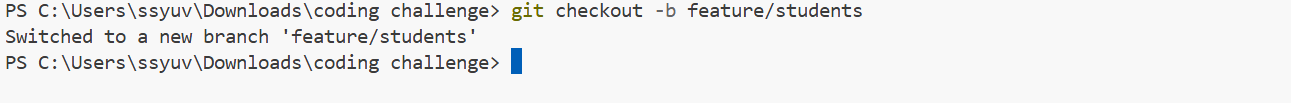
## Section 4: Version Control with Git (10 mins)

Q6. Git Challenge  
- Initialize Git repository

git init

  
- Create and switch to branch feature/students

git checkout -b feature/students

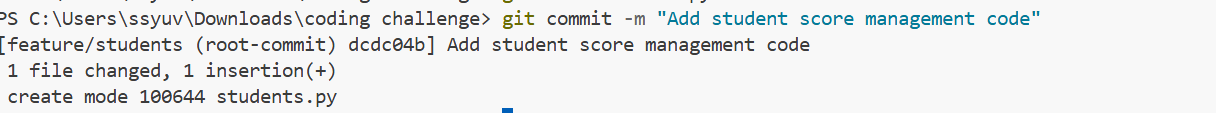


- Add and commit your Python code

git add students.py

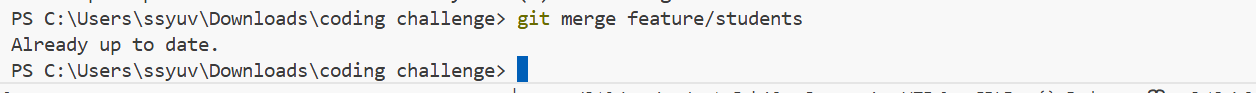


git commit -m "Add student score management code"



- Merge feature/students into main

git merge feature/students

  
- Provide Git commands

## 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 booksample import calculate\_amount, booking

class TestBookingSystem(unittest.TestCase):

    def setUp(self):

        print("\nStarting a test...")

    def tearDown(self):

        print("Test finished.")

    def test\_calculate\_amount(self):

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

    def test\_booking\_valid(self):

        self.assertEqual(booking("Amaran", 3), "Booked 3 for Amaran. Total: ₹360")

    def test\_booking\_invalid(self):

        self.assertEqual(booking("Unknown", 2), "Invalid")

        self.assertEqual(booking("Leo", 0), "Invalid")

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

    unittest.main()

