# Coding Challenge

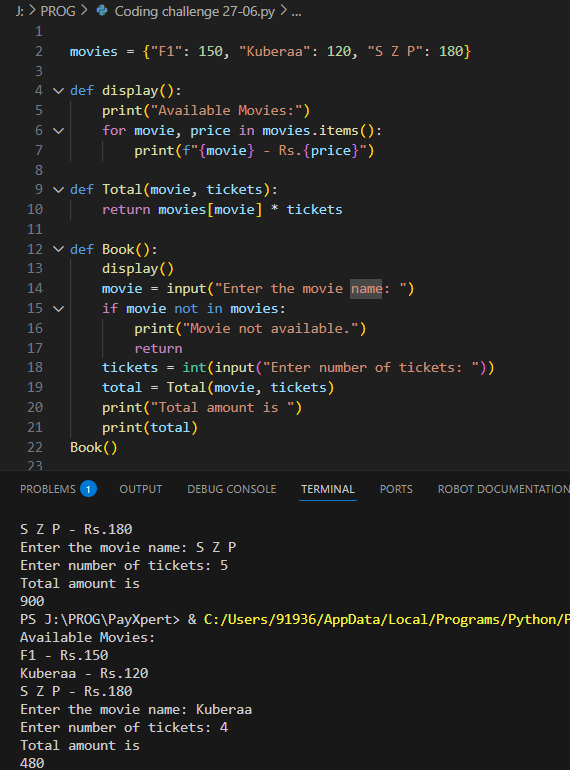
Total Duration: 2 Hours

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



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 student:

    def \_\_init\_\_(self, name):

        self.name = name

    def info(self):

        print("I am")

        print(self.name)

class User(student):

    def \_\_init\_\_(self, name):

        super().\_\_init\_\_(name)

        self.borrowed = []

    def info(self):

        print("Library user:")

        print(self.name)

class Book:

    def \_\_init\_\_(self, title):

        self.title = title

class Library:

    def \_\_init\_\_(self):

        self.books = []

    def add(self, book):

        self.books.append(book)

    def display(self):

        print("Available Books:")

        for book in self.books:

            print(book)

    def borrow(self, user, book):

        if book in self.books:

            self.books.remove(book)

            user.borrowed.append(book)

            print(user.name)

            print("borrowed:")

            print(book)

        else:

            print("Book not available.")

    def returned(self, user, book):

        if book in user.borrowed:

            user.borrowed.remove(book)

            self.books.append(book)

            print(user.name)

            print("returned:")

            print(book)

        else:

            print("This user didn't borrow that book.")

lib = Library()

lib.add("Python")

lib.add("DSA")

lib.add("OOPS")

user1 = User("Jothis")

user1.info()

print("\nInitial Library:")

lib.display()

print("\nBorrowing a book:")

lib.borrow(user1, "Python")

print("\nAfter borrowing:")

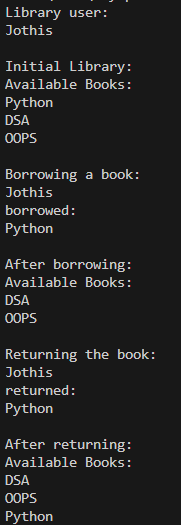
lib.display()

print("\nReturning the book:")

lib.returned(user1, "Python")

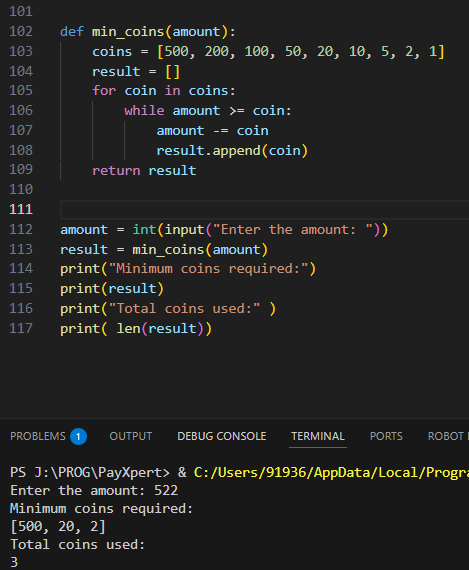
print("\nAfter returning:")

lib.display()

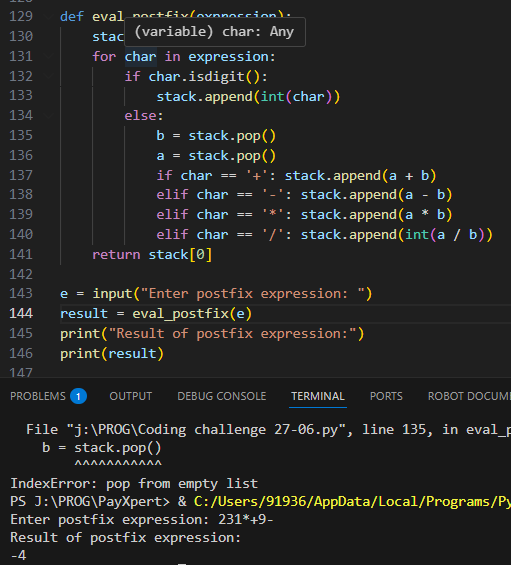


## Section 2: Data Structures & Algorithms (30 mins)

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]



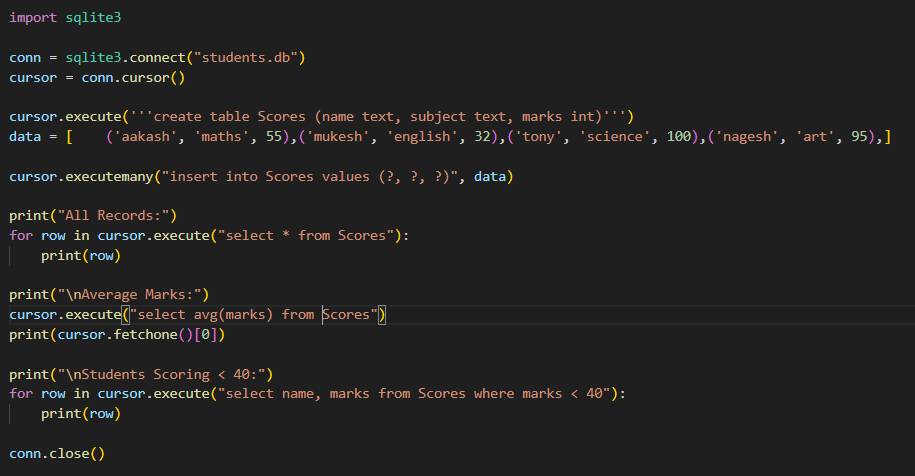
Q4. Data Structure Usage (15 mins)  
- Stack: Evaluate postfix expression '231\*+9-'

  
- Linked List class: append(), display(), reverse()



## 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



## 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 coding challenge 27-06.py

git commit -m "Add Scores table with SQLite and display logic"

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()

