# 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

**SOLUTION**

movies = ["Avengers", "Inception", "The Matrix", "Parasite"]

prices = {

    "Avengers": 250,

    "Inception": 200,

    "The Matrix": 180,

    "Parasite": 150

}

def show\_movies():

    print("Available Movies:")

    for i in range(len(movies)):

        name = movies[i]

        cost = prices[name]

        print(f"  {i+1}. {name} – Rs.{cost}")

def calculate\_amount(movie\_name, ticket\_count):

    price\_per\_ticket = prices.get(movie\_name, 0)

    return price\_per\_ticket \* ticket\_count

def book\_tickets()

    show\_movies()

    choice = input("Enter the movie number you want to book: ").strip()

    if not choice.isdigit():

        print("Invalid input: please enter a number.")

        return

    choice = int(choice)

    if choice < 1 or choice > len(movies):

        print("Invalid selection: that movie number does not exist.")

        return

    selected\_movie = movies[choice - 1]

    qty = input(f"How many tickets for '{selected\_movie}'? ").strip()

    if not qty.isdigit():

        print("Invalid input: please enter a number of tickets.")

        return

    qty = int(qty)

    if qty <= 0:

        print("Invalid quantity: you must book at least one ticket.")

        return

    total = calculate\_amount(selected\_movie, qty)

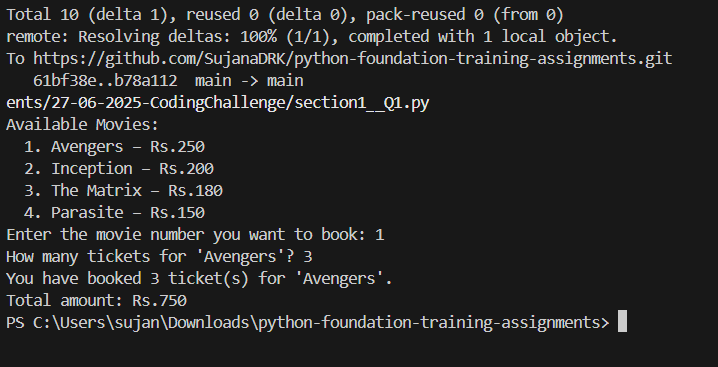
    print(f"You have booked {qty} ticket(s) for '{selected\_movie}'.")

    print(f"Total amount: Rs.{total}")

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

    book\_tickets()

**OUTPUT**

****

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

**SOLUTION**

class Person:

    def \_\_init\_\_(self, name):

        self.name = name

    def display\_info(self):

        print(f"Person: {self.name}")

class User(Person):

    def \_\_init\_\_(self, name):

        super().\_\_init\_\_(name)

        self.borrowed\_books = []

    def display\_info(self):

        # Method overriding

        print(f"Library User: {self.name}")

    def view\_books(self, library):

        print("\nCurrent Library Collection:")

        library.list\_books()

    def borrow\_book(self, title, library):

        book = library.find\_book(title)

        if not book:

            print(f" Book '{title}' not found.")

        elif not book.available:

            print(f" '{title}' is already checked out.")

        else:

            book.available = False

            self.borrowed\_books.append(book)

            print(f" You have borrowed '{title}'.")

    def return\_book(self, title, library):

        for book in self.borrowed\_books:

            if book.title.lower() == title.lower():

                book.available = True

                self.borrowed\_books.remove(book)

                print(f" You have returned '{title}'.")

                return

        print(f" You did not borrow '{title}'.")

class Book:

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

        self.title = title

        self.author = author

        self.available = True

    def \_\_str\_\_(self):

        status = "Available" if self.available else "Checked out"

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

class Library:

     def \_\_init\_\_(self):

        self.collection = []

     def add\_book(self, book):

        self.collection.append(book)

     def list\_books(self):

         if not self.collection:

            print("  (no books in library)")

         for book in self.collection:

            print(f"  - {book}")

     def find\_book(self, title):

        for book in self.collection:

            if book.title.lower() == title.lower():

                return book

        return None

def main():

       library = Library()

       library.add\_book(Book("1984", "George Orwell"))

       library.add\_book(Book("Dune", "Frank Herbert"))

       library.add\_book(Book("Harry Potter", "J.K. Rowling"))

       user = User("Alice")

       print("Welcome to the Library Management System")

       user.display\_info()

       while True:

          print("""

Please choose an action:

  1) View all books

  2) Borrow a book

  3) Return a book

  4) Exit

""")

          choice = input("Enter choice (1–4): ").strip()

          if choice == "1":

             user.view\_books(library)

          elif choice == "2":

            title = input("Enter the title you want to borrow: ").strip()

            user.borrow\_book(title, library)

          elif choice == "3":

            title = input("Enter the title you want to return: ").strip()

            user.return\_book(title, library)

          elif choice == "4":

            print("Goodbye!")

            break

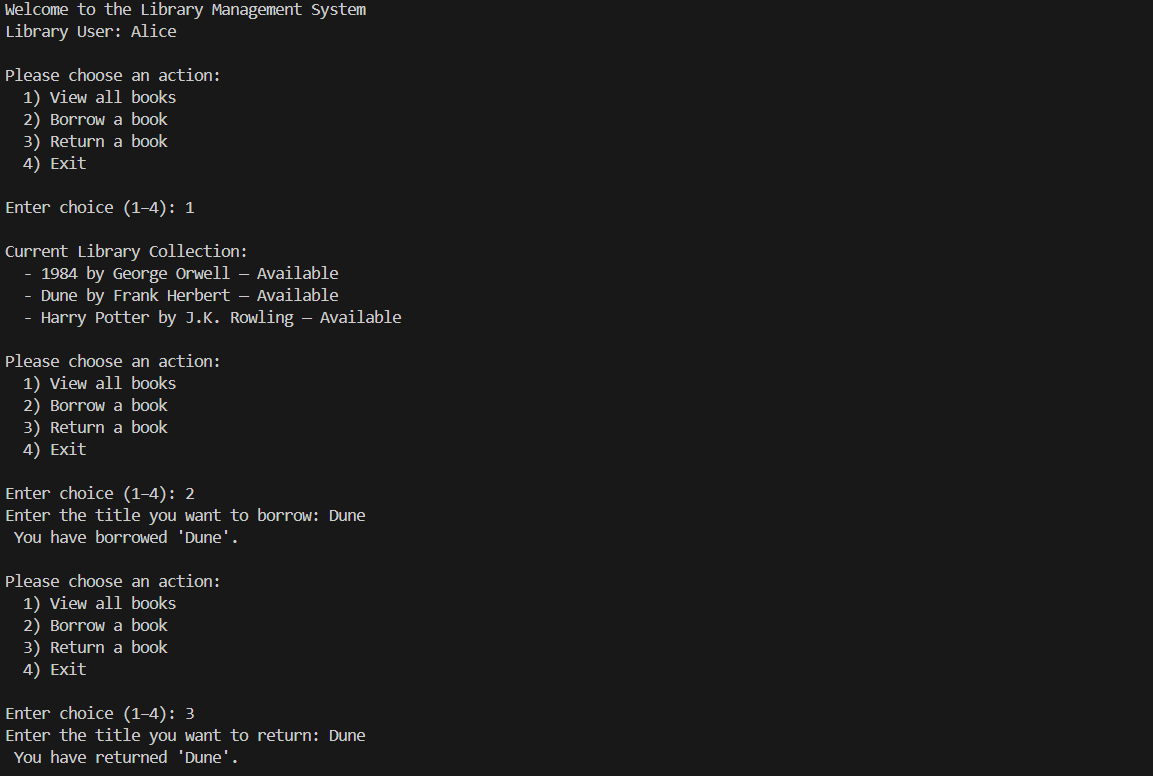
          else:

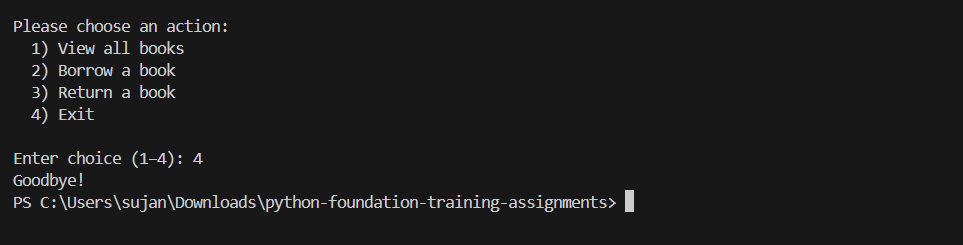
              print("Invalid selection; please choose 1–4.")

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

    main()

**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]

**SOLUTION**

def minimize\_coins(amount):

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

    result = {}

    remaining = amount

    for coin in denominations:

        count = remaining // coin

        if count > 0:

            result[coin] = count

            remaining -= coin \* count

    return result

def main():

    amt\_str = input("Enter amount (integer) to change: ").strip()

    if not amt\_str.isdigit():

        print("Invalid input; please enter a positive integer.")

        return

    amt = int(amt\_str)

    if amt <= 0:

        print("Amount must be greater than zero.")

        return

    coins\_used = minimize\_coins(amt)

    print(f"Minimum coins/notes for Rs.{amt}:")

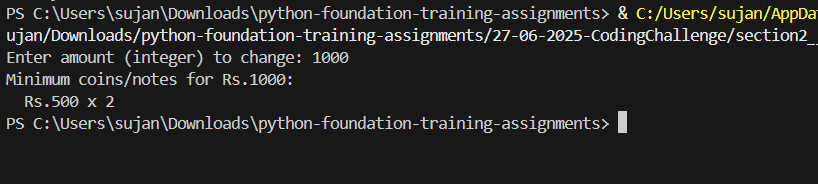
    for coin, cnt in coins\_used.items():

        print(f"  Rs.{coin} x {cnt}")

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

    main()

**OUTPUT**

****

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

**SOLUTION**

def eval\_postfix(expr):

    stack = []

    for token in expr:

        if token.isdigit():

            stack.append(int(token))

        else:

            # pop two operands (b then a)

            b = stack.pop()

            a = stack.pop()

            if token == '+':

                stack.append(a + b)

            elif token == '-':

                stack.append(a - b)

            elif token == '\*':

                stack.append(a \* b)

            elif token == '/':

                stack.append(a // b)

    return stack[0]

# Node class for linked list

class Node:

    def \_\_init\_\_(self, data):

        self.data = data

        self.next = None

# Linked list with append, display, and reverse

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:

            current = self.head

            while current.next:

                current = current.next

            current.next = new\_node

    def display(self):

        values = []

        current = self.head

        while current:

            values.append(str(current.data))

            current = current.next

        print(" -> ".join(values))

    def reverse(self):

        prev = None

        current = self.head

        while current:

            nxt = current.next

            current.next = prev

            prev = current

            current = nxt

        self.head = prev

def main():

    # Part a) Evaluate the example '231\*+9-'

    expr = "231\*+9-"

    result = eval\_postfix(expr)

    print(f"Postfix expression {expr} evaluates to: {result}")

    # Part b) Demonstrate linked list

    print("\nCreating linked list with values 1,2,3,4,5")

    ll = LinkedList()

    for i in [1,2,3,4,5]:

        ll.append(i)

    print("Original list:")

    ll.display()

    print("Reversing list...")

    ll.reverse()

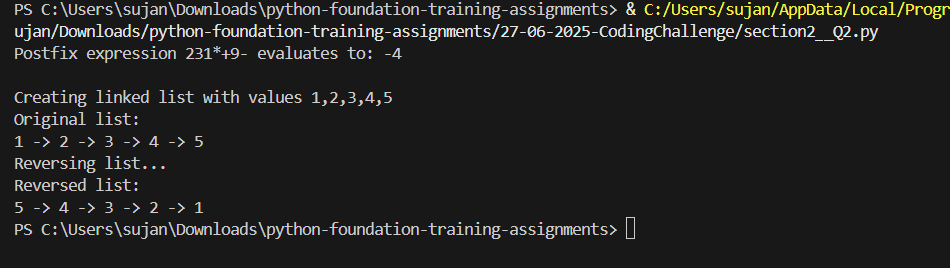
    print("Reversed list:")

    ll.display()

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

    main()

**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

**SOLUTION**

import sqlite3

conn = sqlite3.connect(':memory:')

c = conn.cursor()

c.execute("CREATE TABLE StudentScores (name TEXT, subject TEXT, marks INTEGER)")

c.execute("INSERT INTO StudentScores VALUES ('Alice', 'Math', 75)")

c.execute("INSERT INTO StudentScores VALUES ('Bob', 'Science', 55)")

c.execute("INSERT INTO StudentScores VALUES ('Charlie', 'Math', 35)")

c.execute("INSERT INTO StudentScores VALUES ('Daisy', 'English', 82)")

c.execute("INSERT INTO StudentScores VALUES ('Ethan', 'Science', 29)")

c.execute("INSERT INTO StudentScores VALUES ('Fiona', 'Math', 91)")

print("All Students:")

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

    print(row)

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

avg = c.fetchone()[0]

print("\nAverage Marks:", round(avg, 2))

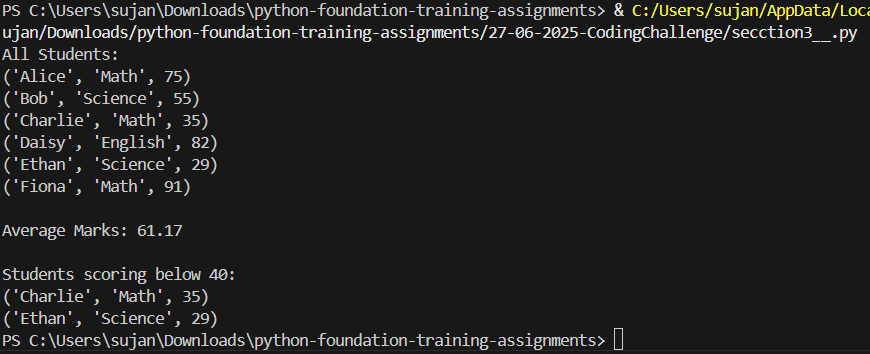
print("\nStudents scoring below 40:")

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

    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

# 1. (If you haven’t already) initialize the repo

git init

# 2. Create & switch to the feature branch

git checkout -b feature/students

# 3. Stage and commit your Python code

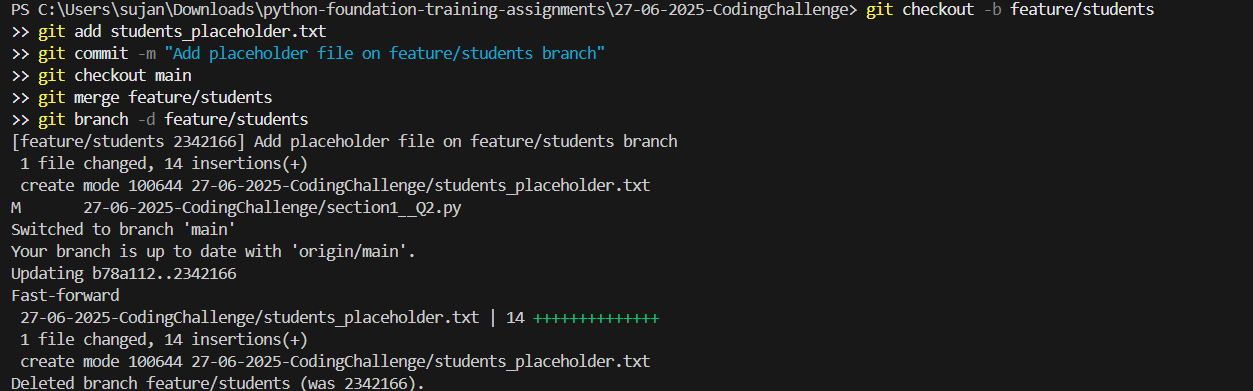
git add .

git commit -m "Add all Python solutions and tests for student challenge"

# 4. Switch back to main and merge the feature branch

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()

**OUTPUT**

import unittest

from io import StringIO

import sys

# Import the functions from your section1q1 script

import section1\_\_Q1

class TestMovieBooking(unittest.TestCase):

    def test\_calculate\_amount(self):

        # Test known prices

        self.assertEqual(section1\_\_Q1.calculate\_amount("Avengers", 3), 250 \* 3)

        self.assertEqual(section1\_\_Q1.calculate\_amount("Parasite", 2), 150 \* 2)

        # Unknown movie should cost 0

        self.assertEqual(section1\_\_Q1.calculate\_amount("Unknown", 5), 0)

    def test\_booking\_flow\_valid(self):

        # Simulate user input: pick movie 1 (Avengers), then 2 tickets

        inputs = ["1", "2"]

        # Capture stdout

        backup\_stdout = sys.stdout

        sys.stdout = StringIO()

        # Monkey-patch input()

        backup\_input = \_\_builtins\_\_.input

        \_\_builtins\_\_.input = lambda prompt="": inputs.pop(0)

        try:

            section1\_\_Q1.book\_tickets()

            output = sys.stdout.getvalue()

        finally:

            # Restore

            sys.stdout = backup\_stdout

            \_\_builtins\_\_.input = backup\_input

        # Check that the output mentions the correct booking and total

        self.assertIn("booked 2 ticket", output)

        self.assertIn("Total amount: Rs.500", output)

    def test\_booking\_flow\_invalid\_movie(self):

        # Non-numeric movie choice should abort early

        inputs = ["x"]

        backup\_stdout = sys.stdout

        sys.stdout = StringIO()

        backup\_input = \_\_builtins\_\_.input

        \_\_builtins\_\_.input = lambda prompt="": inputs.pop(0)

        try:

            section1\_\_Q1.book\_tickets()

            output = sys.stdout.getvalue()

        finally:

            sys.stdout = backup\_stdout

            \_\_builtins\_\_.input = backup\_input

        self.assertIn("Invalid input", output)

    def test\_booking\_flow\_invalid\_qty(self):

        # Valid movie choice, then invalid ticket count

        inputs = ["2", "-3"]

        backup\_stdout = sys.stdout

        sys.stdout = StringIO()

        backup\_input = \_\_builtins\_\_.input

        \_\_builtins\_\_.input = lambda prompt="": inputs.pop(0)

        try:

            section1\_\_Q1.book\_tickets()

            output = sys.stdout.getvalue()

        finally:

            sys.stdout = backup\_stdout

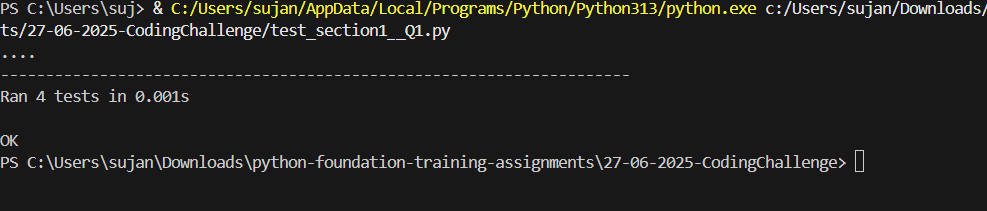
            \_\_builtins\_\_.input = backup\_input

        self.assertIn("Invalid input", output)

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

    unittest.main()

**OUTPUT**

****