# 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

## 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

## 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()

**ANSWERS:**

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

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

movies = ["The Batman", "RRR", "Spider-Man", "Kantara"]

ticket\_price = {

    "The Batman" : 280,

    "RRR" : 350,

    "Spider-Man" : 300,

    "Kantara" : 200

}

def show\_movies():

    print("Available movies: ")

    for idx,movie in enumerate(movies,start=1):

        print(f"{idx}.{movie} -  RS. {ticket\_price[movie]} per ticket")

def cal\_amnt(price,tickets):

    return price\*tickets

def book():

     show\_movies()

     try:

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

         if 1<=choice<=len(movies):

             selected\_movie = movies[choice-1]

             no\_of\_tickets = int(input("How many tickets?"))

             if no\_of\_tickets<=0:

                 print("Invalid number of tickets")

                 return

             total\_price = cal\_amnt(ticket\_price[selected\_movie],no\_of\_tickets)

             print("\nBooking done.")

             print("Movie : ",selected\_movie)

             print("Tickets : ",no\_of\_tickets)

             print("Total amount: ",total\_price)

         else:

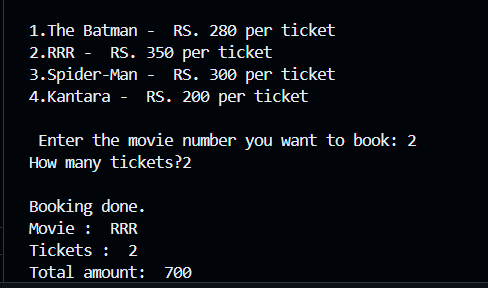
             print("Invalid")

     except:

         print("Invalid input")

book()

OUTPUT:



Q2. OOP Implementation – Library Management

class Book:

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

        self.title = title

        self.author = author

        self.is\_borrowed = False  # Fixed: consistent naming

    def \_\_str\_\_(self):

        return f"{self.title} by {self.author} {'(Borrowed)' if self.is\_borrowed else ''}"

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"You have borrowed '{book.title}'")

                return

        print("Book is not available.")

    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"You have returned '{book.title}'")

                return

        print("Invalid return attempt.")

class User:

    def \_\_init\_\_(self, name, library):

        self.name = name

        self.library = library

    def view\_books(self):

        self.library.view\_books()

    def borrow\_book(self, title):

        self.library.borrow\_book(title)

    def return\_book(self, title):

        self.library.return\_book(title)

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

    lib = Library()

    lib.add\_book(Book("The Alchemist", "Paulo Coelho"))

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

    lib.add\_book(Book("Clean Code", "Robert C. Martin"))

    user1 = User("Alice", lib)

    user1.view\_books()

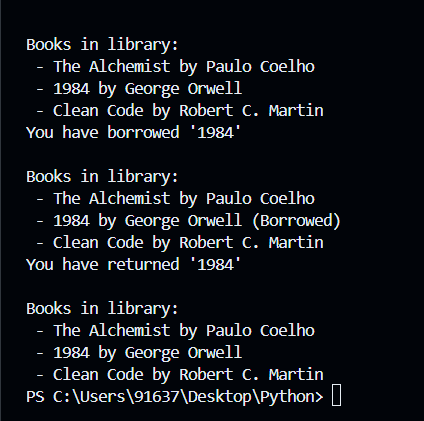
    user1.borrow\_book("1984")

    user1.view\_books()

    user1.return\_book("1984")

    user1.view\_books()

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]

def min\_coins(amount):

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

    res = []

    for coin in denominations:

        while amount>=coin:

            amount -= coin

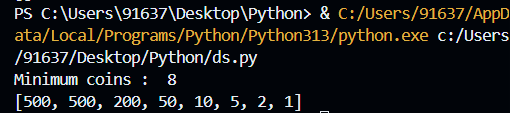
            res.append(coin)

    print("Minimum coins : ",len(res))

    print(res)

min\_coins(1268)

OUTPUT:



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

def evaluate\_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)

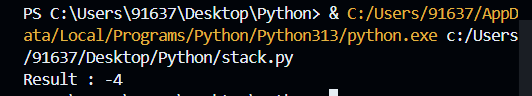
            elif char == '/':

                stack.append(int(a / b))

    return stack.pop()

result = evaluate\_postfix("231\*+9-")

print("Result :", result)



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

        curr = self.head

        while curr.next:

            curr = curr.next

        curr.next = new\_node

    def display(self):

        curr = self.head

        while curr:

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

            curr = curr.next

        print("None")

    def reverse(self):

        prev = None

        curr = self.head

        while curr:

            nxt = curr.next

            curr.next = prev

            prev = curr

            curr = nxt

        self.head = prev

ll = LinkedList()

ll.append(10)

ll.append(20)

ll.append(30)

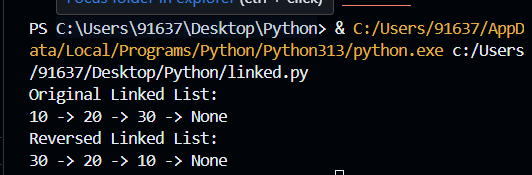
print("Original Linked List:")

ll.display()

ll.reverse()

print("Reversed Linked List:")

ll.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

import mysql.connector

conn = mysql.connector.connect(

    host="localhost",

    user="root",

    password="12345678",

    database="school"

)

cursor = conn.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS StudentScores (

    name VARCHAR(100),

    subject VARCHAR(100),

    marks INT

)

""")

sample\_data = [

    ('Alice', 'Math', 85),

    ('Bob', 'Math', 35),

    ('Charlie', 'Science', 78),

    ('David', 'Science', 39),

    ('Eva', 'English', 92),

    ('Frank', 'English', 25)

]

cursor.execute("DELETE FROM StudentScores")

insert\_query = "INSERT INTO StudentScores (name, subject, marks) VALUES (%s, %s, %s)"

cursor.executemany(insert\_query, sample\_data)

conn.commit()

print("\nAll Student Records:")

cursor.execute("SELECT \* FROM StudentScores")

for row in cursor.fetchall():

    print(row)

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

avg\_marks = cursor.fetchone()[0]

print(f"\nAverage Marks: {avg\_marks:.2f}")

print("\nStudents scoring less than 40:")

cursor.execute("SELECT name, subject, marks FROM StudentScores WHERE marks < 40")

for row in cursor.fetchall():

    print(row)

cursor.close()

conn.close()

## 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

Answers:

Git init – initialize git repository

git branch feature/students

git checkout feature/students

git add .

git commit -m "Add student"

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

class TestBookingSystem(unittest.TestCase):

def setUp(self):

print("Setting up...")

def tearDown(self):

print("Tearing down...\n")

def test\_calculate\_amount(self):

self.assertEqual(calculate\_amount(300, 2), 600)

self.assertEqual(calculate\_amount(250, 0), 0)

@patch('builtins.input', side\_effect=['2', '3'])

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

def test\_book\_ticket(self, mock\_stdout, mock\_input):

book\_ticket()

output = mock\_stdout.getvalue()

self.assertIn("Total: ₹900", output)

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

unittest.main()