# 27/06/2025

# Coding Challenge

Total Duration: 2 Hours

## 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.

movies = {

"Avatar": 190,

"Mufasa": 220,

"Final Destination Bloodlines": 200,

"DNA": 180

}

def show\_movies():

print("Available Movies:")

for movie, price in movies.items():

print(f"{movie}: ₹{price}")

def calculate\_amount(movie, tickets):

return movies[movie] \* tickets

def book\_movie():

show\_movies()

movie = input("Enter movie name: ")

if movie not in movies:

print("Movie not available!")

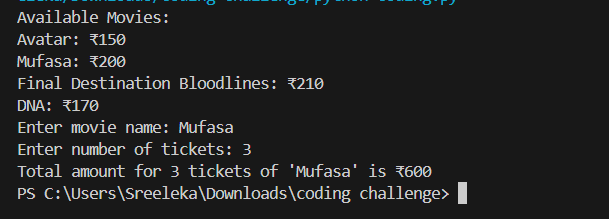
return

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

total = calculate\_amount(movie, tickets)

print(f"Total amount for {tickets} tickets of '{movie}' is ₹{total}")

book\_movie()

**OUTPUT:**

2) 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

class Library:

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

self.books = []

def add\_book(self, book):

self.books.append(book)

def view\_books(self):

print("\n----- Library Books -----")

for book in self.books:

status = "Borrowed" if book.is\_borrowed else "Available"

print(f"{book.title} by {book.author} - {status}")

def borrow\_book(self, book\_title):

for book in self.books:

if book\_title.lower() in book.title.lower():

if not book.is\_borrowed:

book.is\_borrowed = True

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

return

else:

print("Book is already borrowed.")

return

print("Book not found.")

def return\_book(self, book\_title):

for book in self.books:

if book\_title.lower() in book.title.lower():

if book.is\_borrowed:

book.is\_borrowed = False

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

return

else:

print("This book wasn't borrowed.")

return

print("Book not found.")

class User(Library):

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

super().\_\_init\_\_()

self.name = name

library = Library()

library.add\_book(Book("Learning Python", "Mark Lutz"))

library.add\_book(Book("Data Structures and Algorithms in Python", "Michael T. Goodrich"))

library.add\_book(Book("OOP in Python", "Alice Johnson"))

library.add\_book(Book("Deep Learning with Python", "François Chollet"))

library.add\_book(Book("Python Crash Course", "Eric Matthes"))

library.add\_book(Book("Automate the Boring Stuff with Python", "Al Sweigart"))

library.view\_books()

library.borrow\_book("learning python")

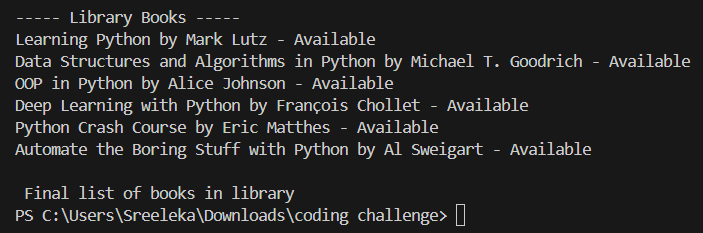
library.borrow\_book("nonexistent book")

library.return\_book("learning python")

library.return\_book("OOP in Python")

library.view\_books()

print("\n Final list of books in library")

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 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)

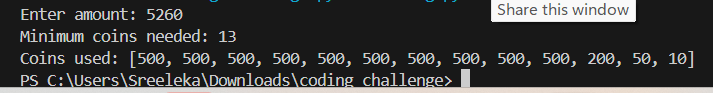
    return result

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

coins\_used = minimize\_coins(amount)

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

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



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

def eval\_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(int(a / b))

return stack[0]

expression = '231\*+9-'

result = eval\_postfix(expression)

print(f"Result of postfix expression '{expression}': {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

current = self.head

while current.next:

current = current.next

current.next = new\_node

def display(self):

current = self.head

print("Linked List:", end=" ")

while current:

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

current = current.next

print("None")

def reverse(self):

prev = None

current = self.head

while current:

next\_node = current.next

current.next = prev

prev = current

current = next\_node

self.head = prev

ll = LinkedList()

ll.append(15)

ll.append(45)

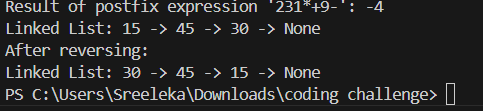
ll.append(30)

ll.display()

ll.reverse()

print("After reversing:")

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 sqlite3

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

cur = conn.cursor()

cur.execute("CREATE TABLE StudentScores(name TEXT, subject TEXT, marks INT)")

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

('Harsha', 'Math', 85),

('Neha', 'Science', 45),

('Thara', 'Math', 92),

('Durga', 'Science', 38)

])

print("All Records:")

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

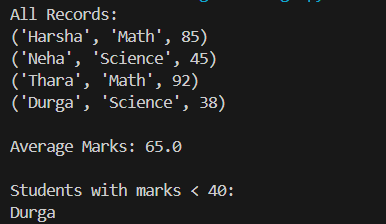
    print(row)

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

print("\nAverage Marks:", cur.fetchone()[0])

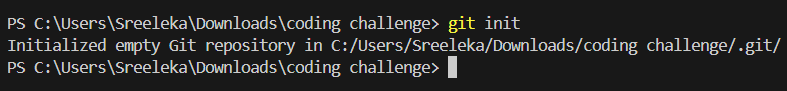
print("\nStudents with marks < 40:")

for row in cur.execute("SELECT name FROM StudentScores WHERE marks < 40"):

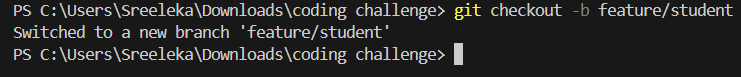
    print(row[0])

Q6. Git Challenge

- Initialize Git repository



- Create and switch to branch feature/students



- Add and commit your Python code



## 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()

movies = {

"Avengers": 150,

"Inception": 180,

"Coco": 120

}

def show\_movies():

return list(movies.keys())

def calculate\_amount(movie, tickets):

return movies[movie] \* tickets

def book\_movie(movie, tickets):

if movie in movies:

total = calculate\_amount(movie, tickets)

return f"{tickets} tickets booked for {movie}. Total: ₹{total}"

else:

return "Movie not available"

**FOR UNIT TEST:**

import unittest

from movie\_booking import calculate\_amount, book\_movie

class TestMovieBooking(unittest.TestCase):

def setUp(self):

print("Setting up test case...")

def tearDown(self):

print("Tearing down test case...")

def test\_calculate\_amount(self):

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

self.assertEqual(calculate\_amount("Coco", 3), 360)

def test\_book\_movie(self):

self.assertEqual(

book\_movie("Inception", 1),

"1 tickets booked for Inception. Total: ₹180"

)

self.assertEqual(

book\_movie("Unknown", 2),

"Movie not available"

)

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

unittest.main()

