ASSIGNMENT-4

32. def calculate\_student\_users(total\_users, staff\_users):

if total\_users < 0 or staff\_users < 0 or staff\_users > total\_users:

return "Invalid input"

# Calculate non-teaching staff users (1 for every 3 staff users)

non\_teaching\_staff\_users = staff\_users // 3

# Calculate total staff users (teaching + non-teaching)

total\_staff\_users = staff\_users + non\_teaching\_staff\_users

# Calculate student users

student\_users = total\_users - total\_staff\_users

return student\_users

# Sample input

total\_users = 856

staff\_users = 126

student\_users = calculate\_student\_users(total\_users, staff\_users)

print(f"Sample Output: Student Users: {student\_users}")

# Test cases

test\_cases = [

(0, 0),

(-143, 0),

(1026, 1026),

(450, 540),

(600, 450)

]

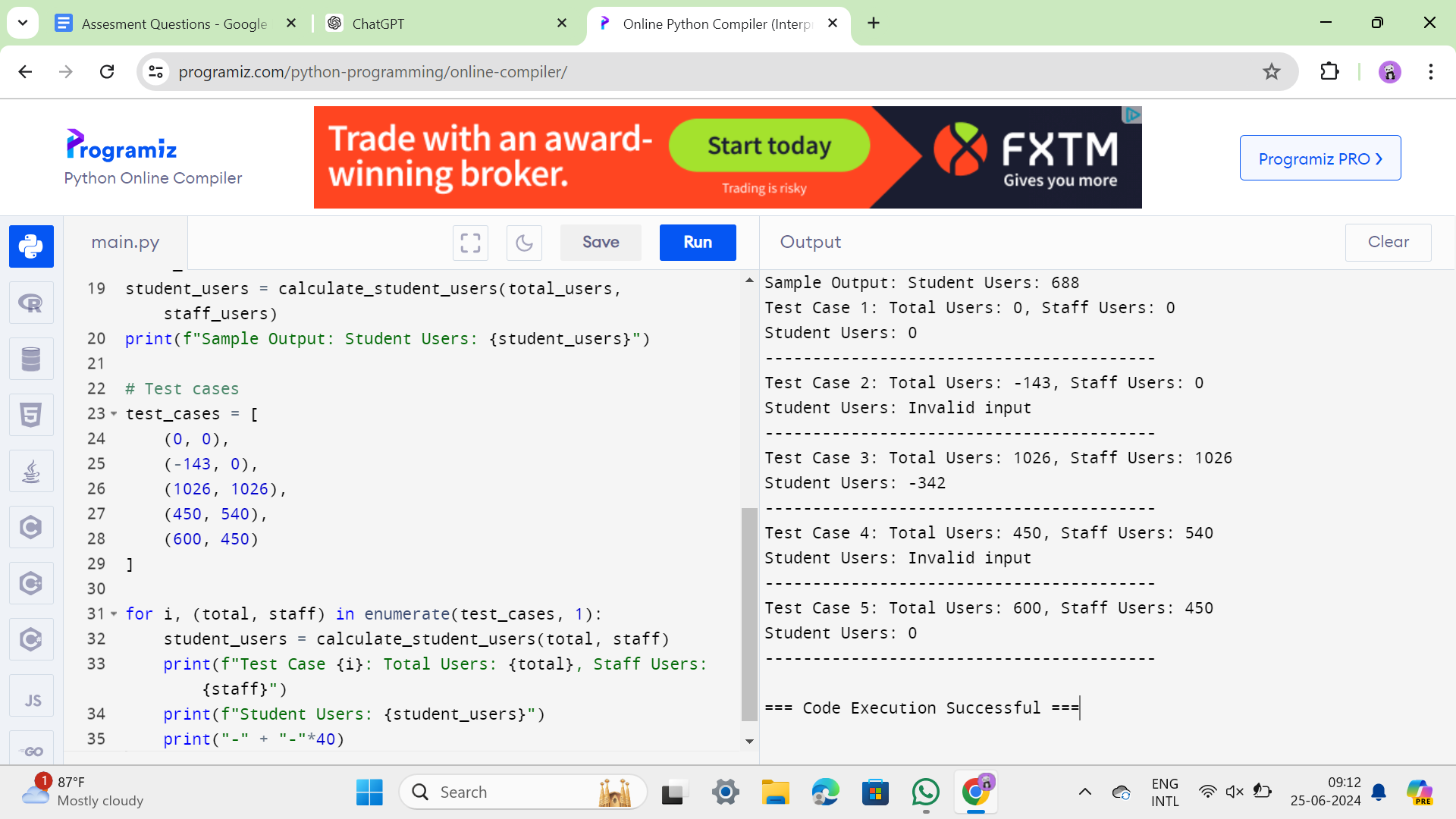
for i, (total, staff) in enumerate(test\_cases, 1):

student\_users = calculate\_student\_users(total, staff)

print(f"Test Case {i}: Total Users: {total}, Staff Users: {staff}")

print(f"Student Users: {student\_users}")

print("-" + "-"\*40)



33. def is\_leap\_year(year):

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):

return True

return False

def check\_date(date):

try:

# Split the date by '/'

parts = date.split('/')

# Convert parts to integers

month = int(parts[0])

day = int(parts[1])

year = int(parts[2])

# Check if the date is valid

if not (1 <= month <= 12 and 1 <= day <= 31 and year > 0):

return "Invalid date"

# Check if the year is a leap year

if is\_leap\_year(year):

return f"Given year {year} is a Leap Year"

else:

return f"Given year {year} is Non Leap Year"

except:

return "Invalid date"

# Sample Input

date\_input = "04/11/1947"

print(f"Sample Input: Enter Date : {date\_input}")

print("Sample Output:")

print(check\_date(date\_input))

print()

# Test Cases

test\_cases = [

"04/11/19.47",

"11/15/1936",

"31/45/1996",

"64/09/1947",

"00/00/2000"

]

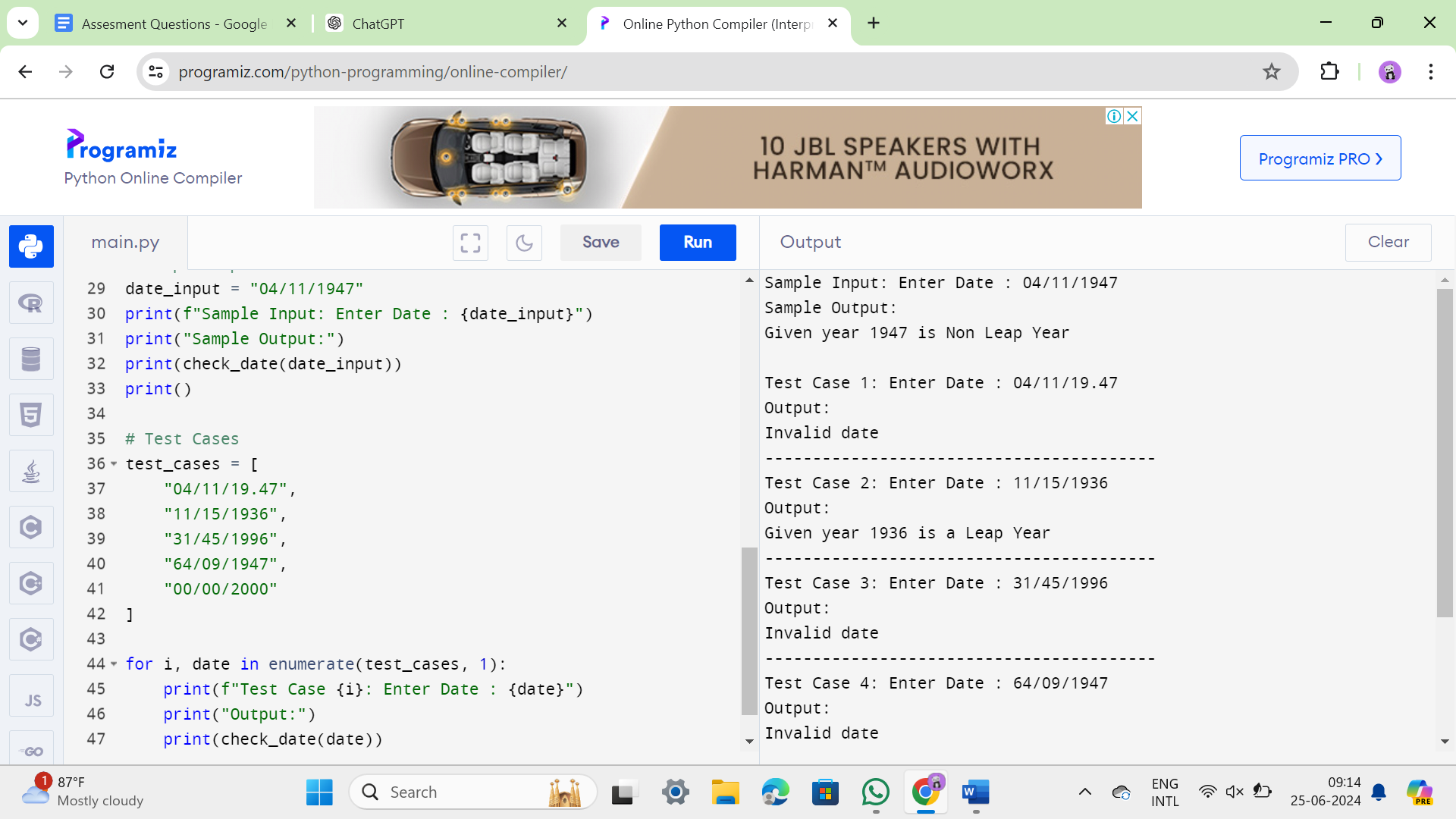
for i, date in enumerate(test\_cases, 1):

print(f"Test Case {i}: Enter Date : {date}")

print("Output:")

print(check\_date(date))

print("-" + "-"\*40)



34.

import re

def is\_leap\_year(year):

if (year % 4 == 0 and year % 100 != 0) or (year % 400 == 0):

return True

return False

def check\_date(date):

# Regex to check valid date format dd/mm/yyyy

date\_pattern = re.compile(r'^\d{2}/\d{2}/\d{4}$')

if not date\_pattern.match(date):

return "Invalid date format"

try:

# Split the date by '/'

parts = date.split('/')

# Convert parts to integers

day = int(parts[0])

month = int(parts[1])

year = int(parts[2])

# Check if the date is valid

if not (1 <= month <= 12 and 1 <= day <= 31 and year > 0):

return "Invalid date"

# Check if the year is a leap year

if is\_leap\_year(year):

return f"Given year {year} is a Leap Year"

else:

return f"Given year {year} is Non Leap Year"

except:

return "Invalid date"

# Sample Input

date\_input = "04/11/1947"

print(f"Sample Input: Enter Date : {date\_input}")

print("Sample Output:")

print(check\_date(date\_input))

print()

# Test Cases

test\_cases = [

"04/11/19.47",

"11/15/1936",

"31/45/1996",

"64/09/1947",

"00/00/2000",

"Welcome to &^23(&@ SSE"

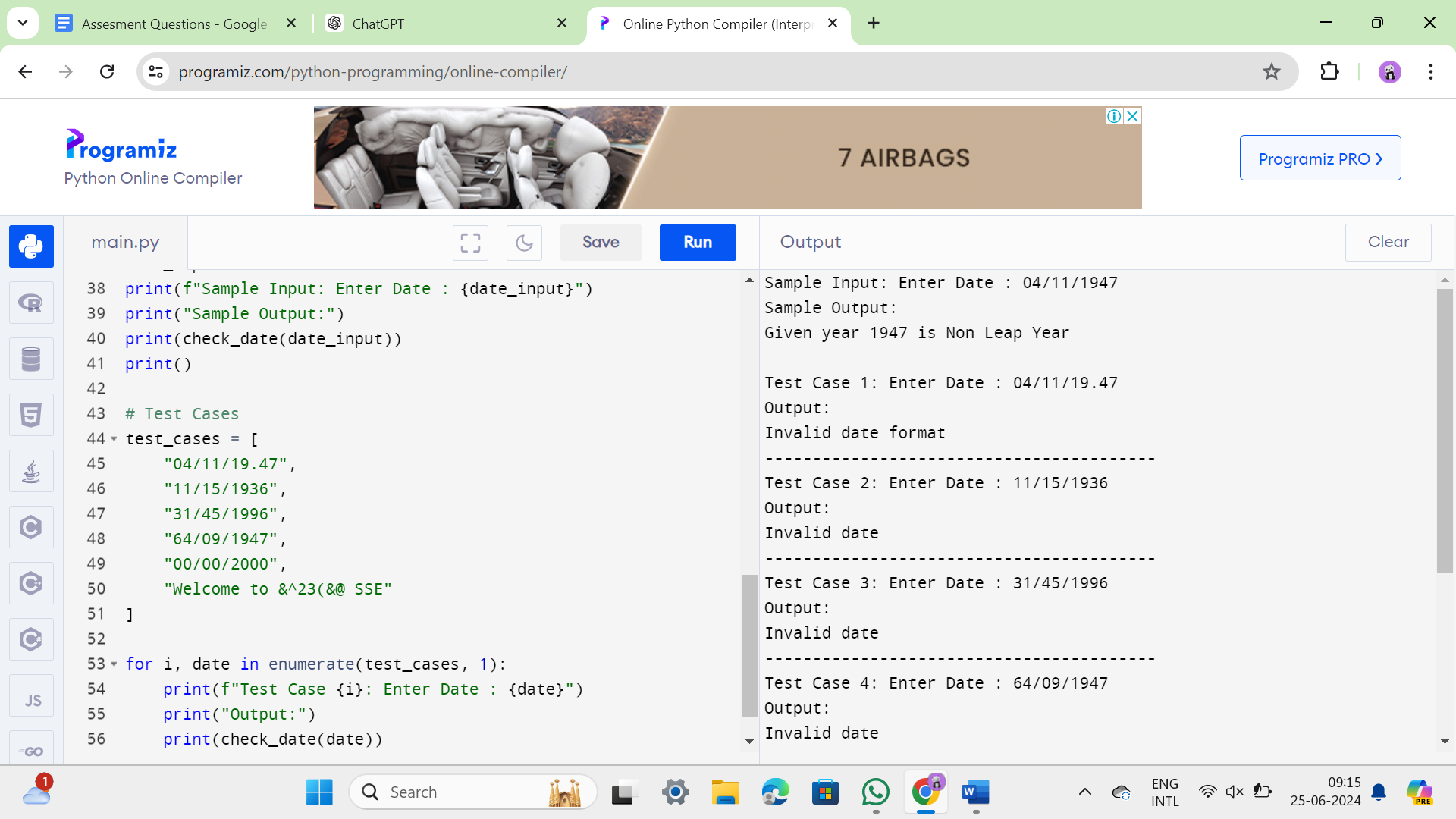
]

for i, date in enumerate(test\_cases, 1):

print(f"Test Case {i}: Enter Date : {date}")

print("Output:")

print(check\_date(date))

print("-" + "-"\*40)

35. def calculate\_grade(marks):

# Calculate total marks

total = sum(marks)

# Calculate aggregate percentage

aggregate = (total / (len(marks) \* 100)) \* 100

# Determine grade based on aggregate percentage

if aggregate > 75:

grade = "Distinction"

elif 60 <= aggregate <= 75:

grade = "First Division"

elif 50 <= aggregate < 60:

grade = "Second Division"

elif 40 <= aggregate < 50:

grade = "Third Division"

else:

grade = "Fail"

return total, aggregate, grade

# Input marks for four subjects

marks = []

for i in range(1, 5):

mark = float(input(f"Enter marks for subject {i} (out of 100): "))

marks.append(mark)

# Calculate total, aggregate, and grade

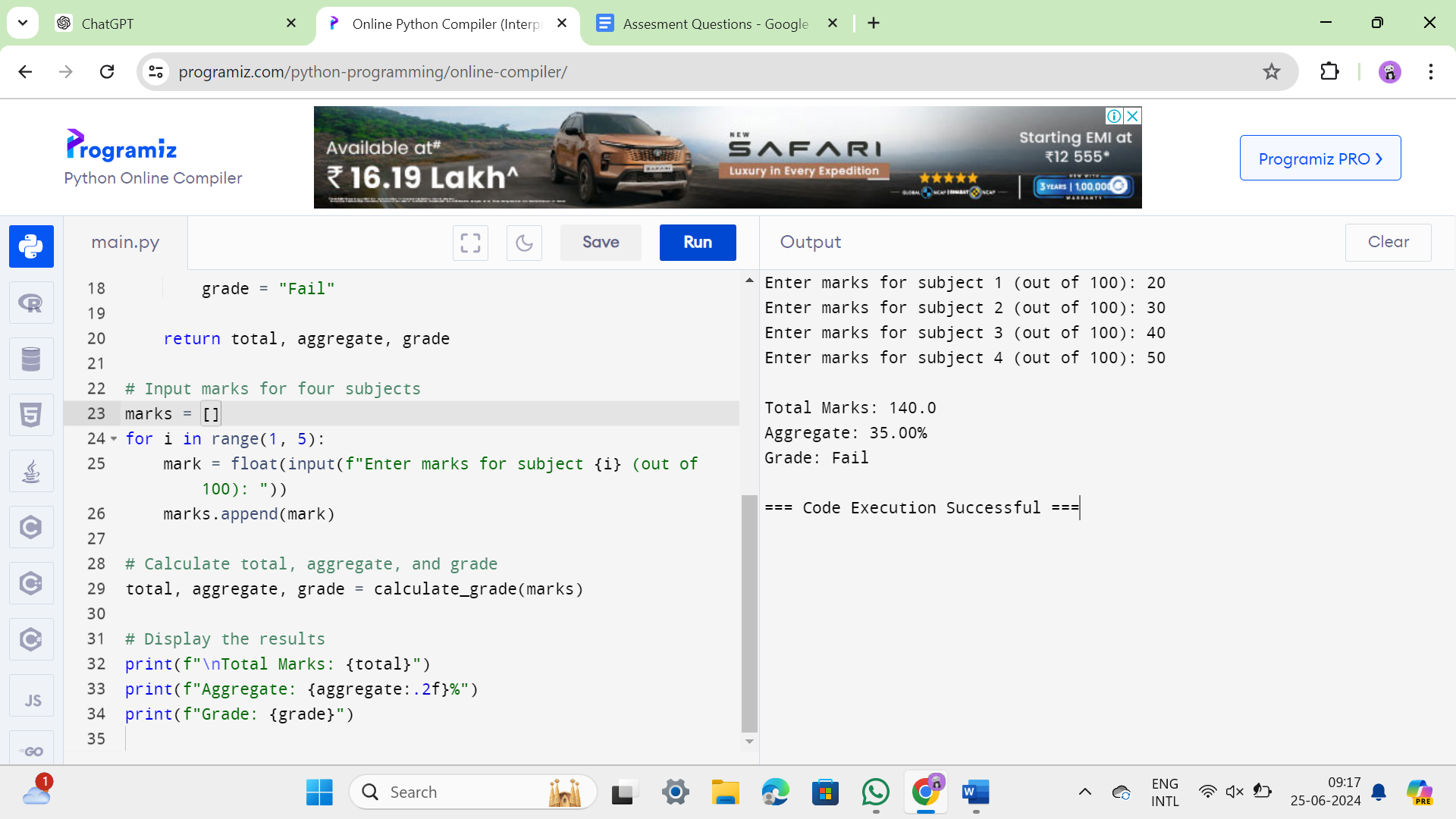
total, aggregate, grade = calculate\_grade(marks)

# Display the results

print(f"\nTotal Marks: {total}")

print(f"Aggregate: {aggregate:.2f}%")

print(f"Grade: {grade}")



36.

def factorial(n):

# Base case: factorial of 0 or 1 is 1

if n == 0 or n == 1:

return 1

# Recursive case: n \* factorial of (n-1)

elif n > 1:

return n \* factorial(n - 1)

else:

return None # To handle negative inputs

def main():

try:

# Get user input

n = input("Enter the value of n: ")

# Check if input is a valid integer

if not n.isdigit() and not (n.startswith('-') and n[1:].isdigit()):

raise ValueError("Invalid input: Please enter a valid integer")

# Convert input to integer

n = int(n)

# Handle negative inputs

if n < 0:

raise ValueError("Invalid input: Factorial is not defined for negative numbers")

# Calculate factorial

result = factorial(n)

# Output result

print(f"The factorial of {n} is: {result}")

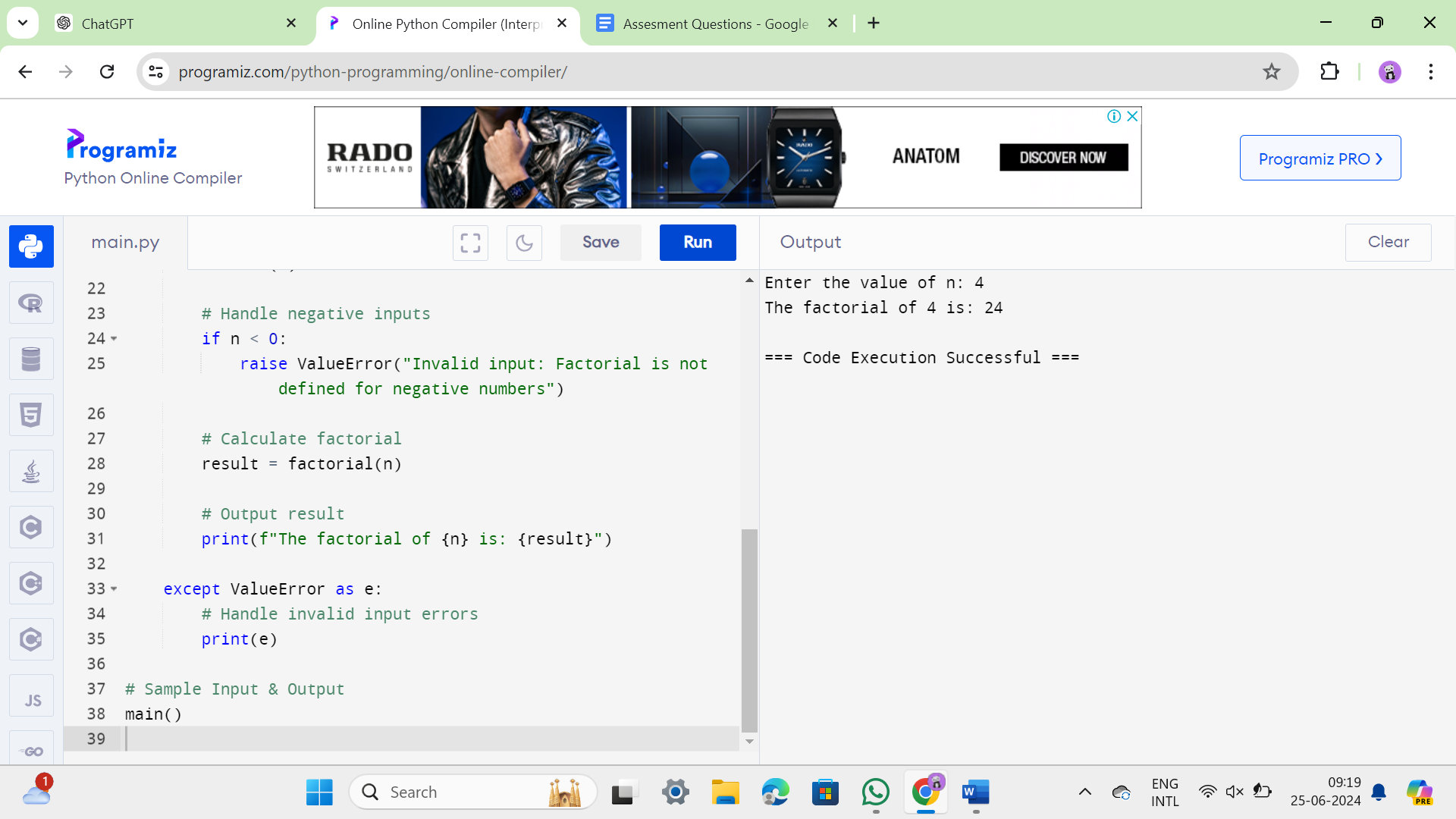
except ValueError as e:

# Handle invalid input errors

print(e)

# Sample Input & Output

main()



37

def find\_nth\_largest\_number(nums, n):

# Sort the list in descending order

sorted\_nums = sorted(nums, reverse=True)

# Check if n is within valid range

if n <= 0 or n > len(sorted\_nums):

return None

# Return the nth largest number

return sorted\_nums[n - 1]

def main():

try:

# Sample Input

nums = [14, 67, 48, 23, 5, 62]

print(f"List : {nums}")

# Prompt user to enter value of N

n = input("Enter the value of N: ")

# Check if input is a valid integer

if not n.isdigit() and not (n.startswith('-') and n[1:].isdigit()):

raise ValueError("Invalid input: Please enter a valid integer")

# Convert input to integer

n = int(n)

# Find the Nth largest number

nth\_largest = find\_nth\_largest\_number(nums, n)

if nth\_largest is None:

raise ValueError(f"N = {n} is out of range or invalid")

# Output the result

print(f"{n}th Largest number: {nth\_largest}")

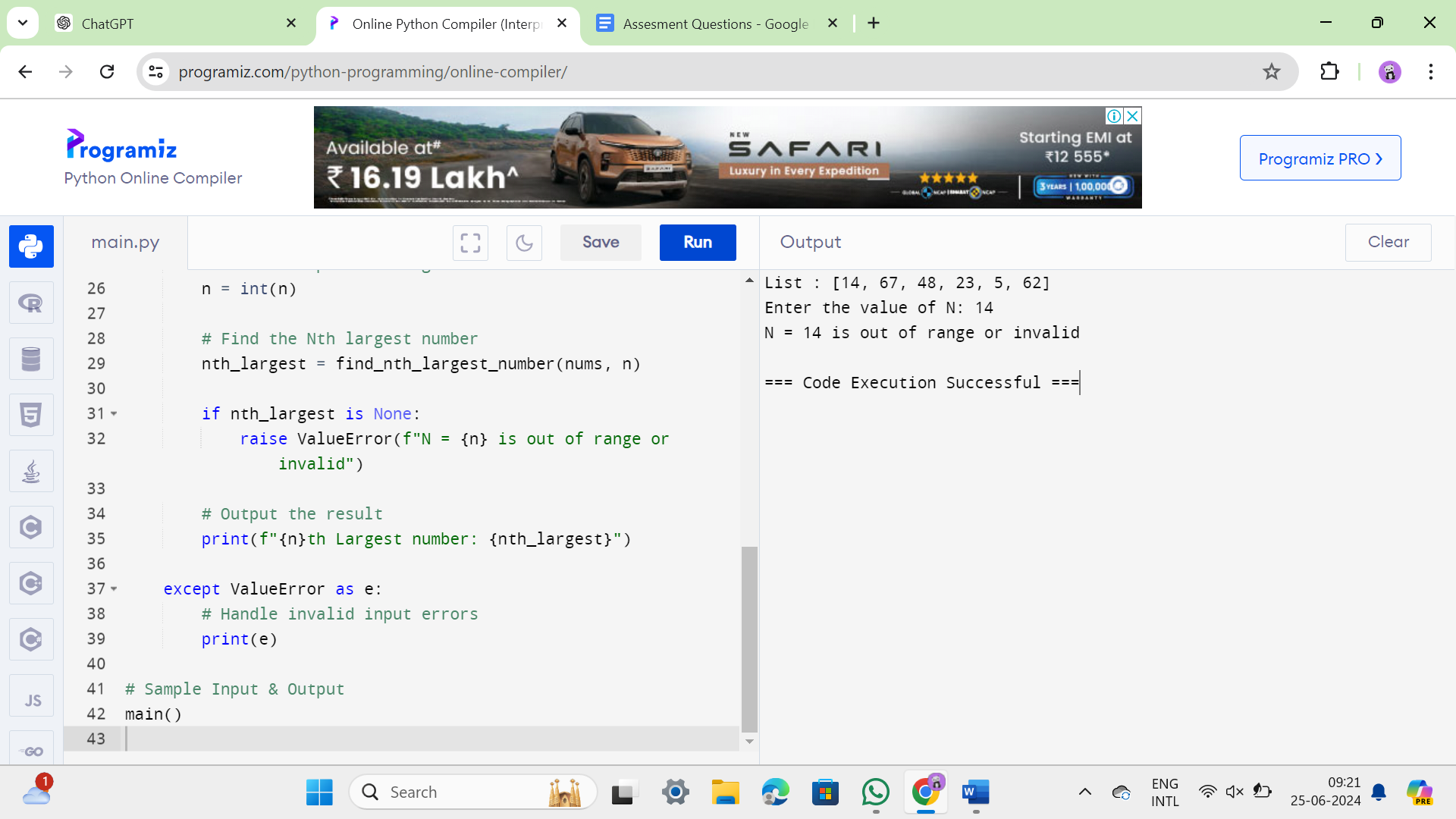
except ValueError as e:

# Handle invalid input errors

print(e)

# Sample Input & Output

main()



38. def remove\_duplicates(nums):

if not nums:

return []

# Initialize variables

unique\_nums = []

current\_num = nums[0]

# Iterate through the array

for num in nums[1:]:

if num != current\_num:

unique\_nums.append(current\_num)

current\_num = num

# Add the last unique number

unique\_nums.append(current\_num)

return unique\_nums

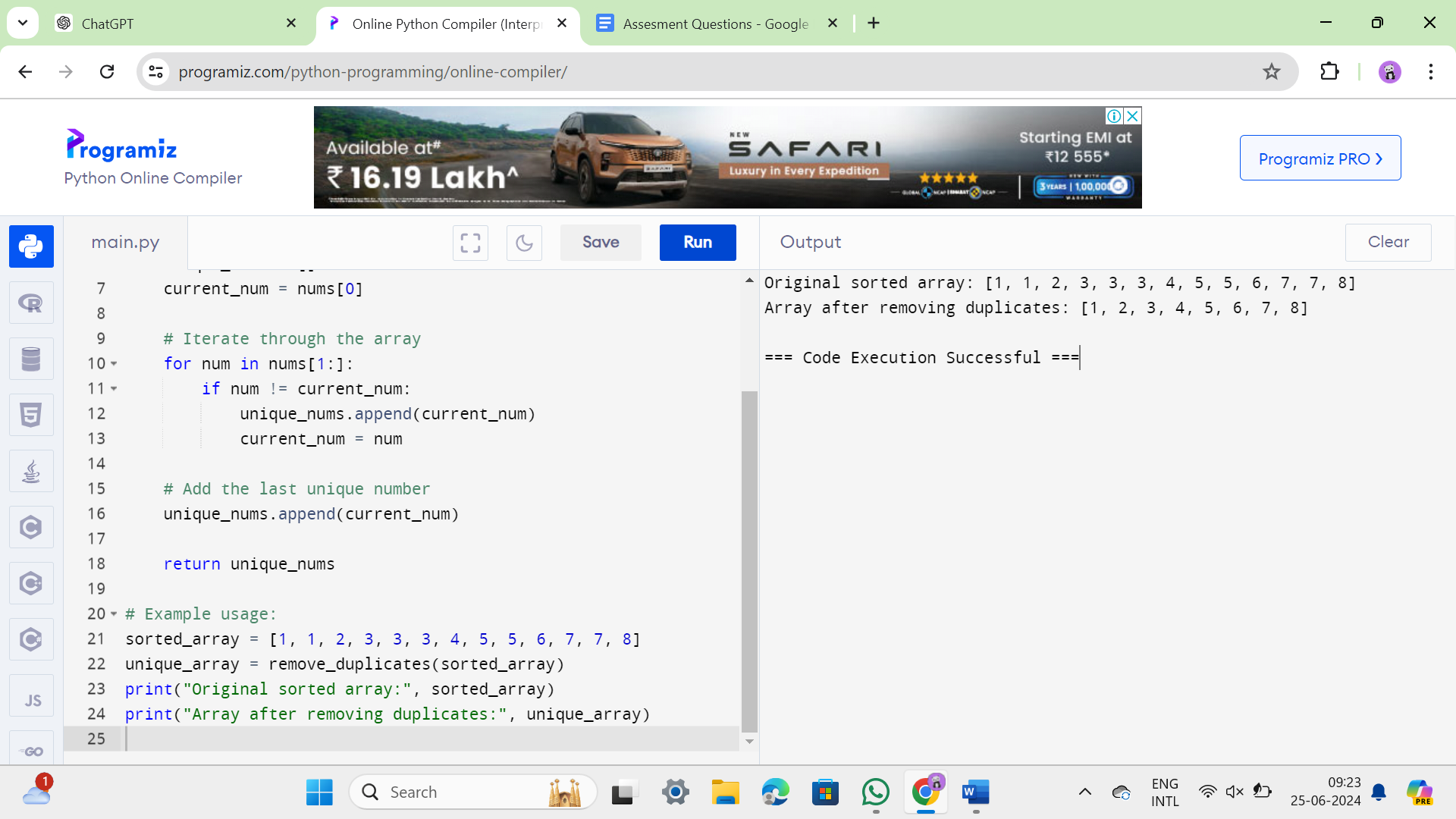
# Example usage:

sorted\_array = [1, 1, 2, 3, 3, 3, 4, 5, 5, 6, 7, 7, 8]

unique\_array = remove\_duplicates(sorted\_array)

print("Original sorted array:", sorted\_array)

print("Array after removing duplicates:", unique\_array)



39. def merge\_sorted\_lists(list1, list2):

# Initialize empty list for merged result

merged\_list = []

# Initialize indices for both lists

i, j = 0, 0

# Iterate through both lists

while i < len(list1) and j < len(list2):

if list1[i] <= list2[j]:

merged\_list.append(list1[i])

i += 1

else:

merged\_list.append(list2[j])

j += 1

# Add remaining elements from list1, if any

while i < len(list1):

merged\_list.append(list1[i])

i += 1

# Add remaining elements from list2, if any

while j < len(list2):

merged\_list.append(list2[j])

j += 1

return merged\_list

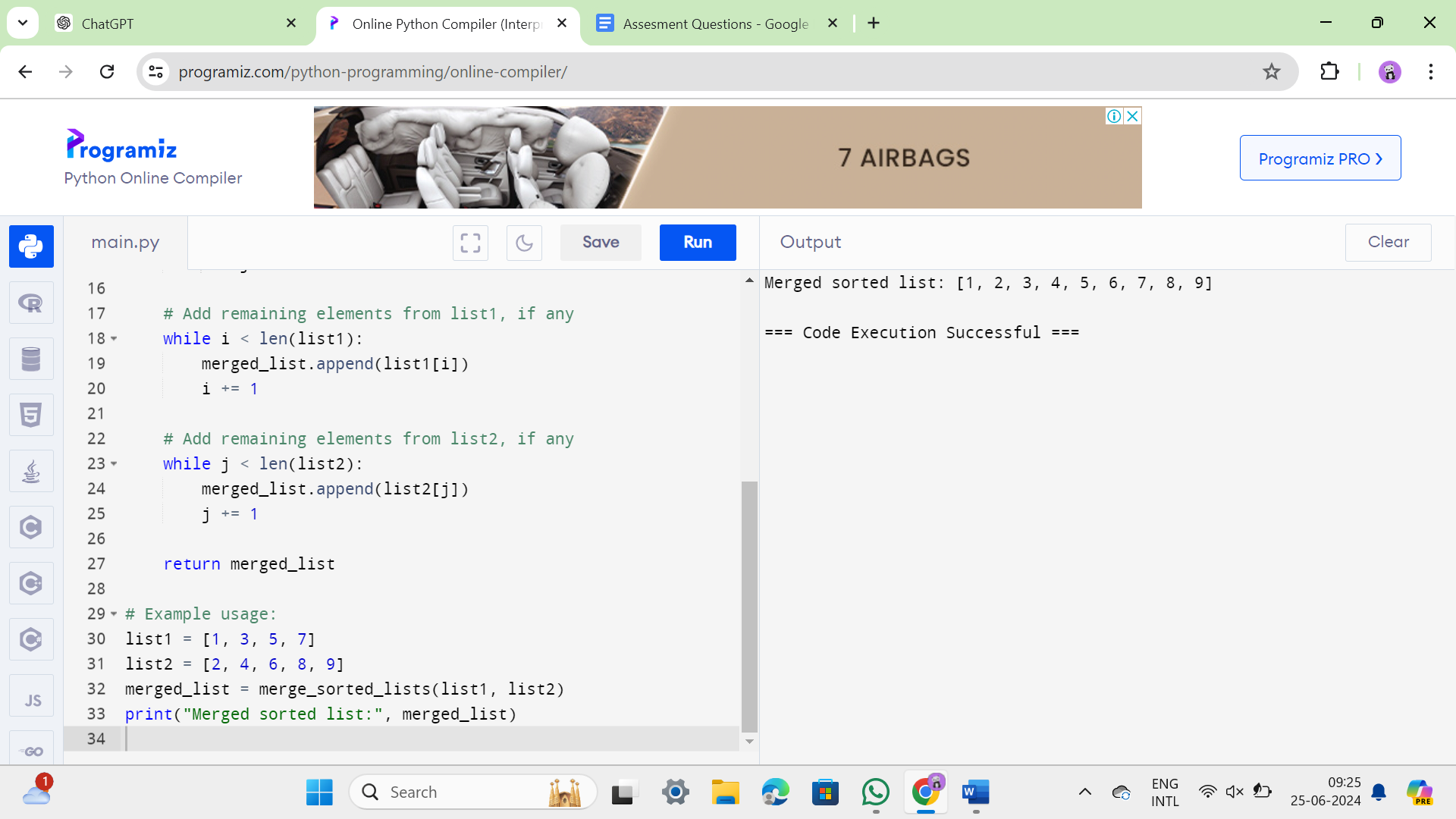
# Example usage:

list1 = [1, 3, 5, 7]

list2 = [2, 4, 6, 8, 9]

merged\_list = merge\_sorted\_lists(list1, list2)

print("Merged sorted list:", merged\_list)



40. def find\_mth\_maximum\_and\_nth\_minimum(array, m, n):

# Sort the array in ascending order

sorted\_array = sorted(array)

# Find the Mth maximum number (considering duplicates)

if m > 0:

mth\_max = sorted\_array[-m]

else:

mth\_max = None

# Find the Nth minimum number (considering duplicates)

if n > 0:

nth\_min = sorted\_array[n - 1]

else:

nth\_min = None

return mth\_max, nth\_min

def calculate\_sum\_and\_difference(mth\_max, nth\_min):

if mth\_max is None or nth\_min is None:

return None, None

# Calculate sum and difference

sum\_result = mth\_max + nth\_min

difference\_result = abs(mth\_max - nth\_min)

return sum\_result, difference\_result

def main():

# Sample Input

array = [14, 16, 87, 36, 25, 89, 34]

m = 1

n = 3

# Find Mth maximum and Nth minimum

mth\_max, nth\_min = find\_mth\_maximum\_and\_nth\_minimum(array, m, n)

if mth\_max is not None and nth\_min is not None:

# Calculate sum and difference

sum\_result, difference\_result = calculate\_sum\_and\_difference(mth\_max, nth\_min)

# Print results

print(f"{m}st Maximum Number = {mth\_max} {n}rd Minimum Number = {nth\_min}")

print(f"Sum = {sum\_result}")

print(f"Difference = {difference\_result}")

else:

print("Invalid input: Please ensure M and N are greater than 0 and within range.")

# Sample Input & Output

main()

