**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.1**

**Title: Simple Interest**

**Theory:**  The following code takes user input for principal, rate of interest and number of years; Then uses the simple interest formula to calculate the interest received.

**Code:**

p=float(input("Enter the principal amount : "))

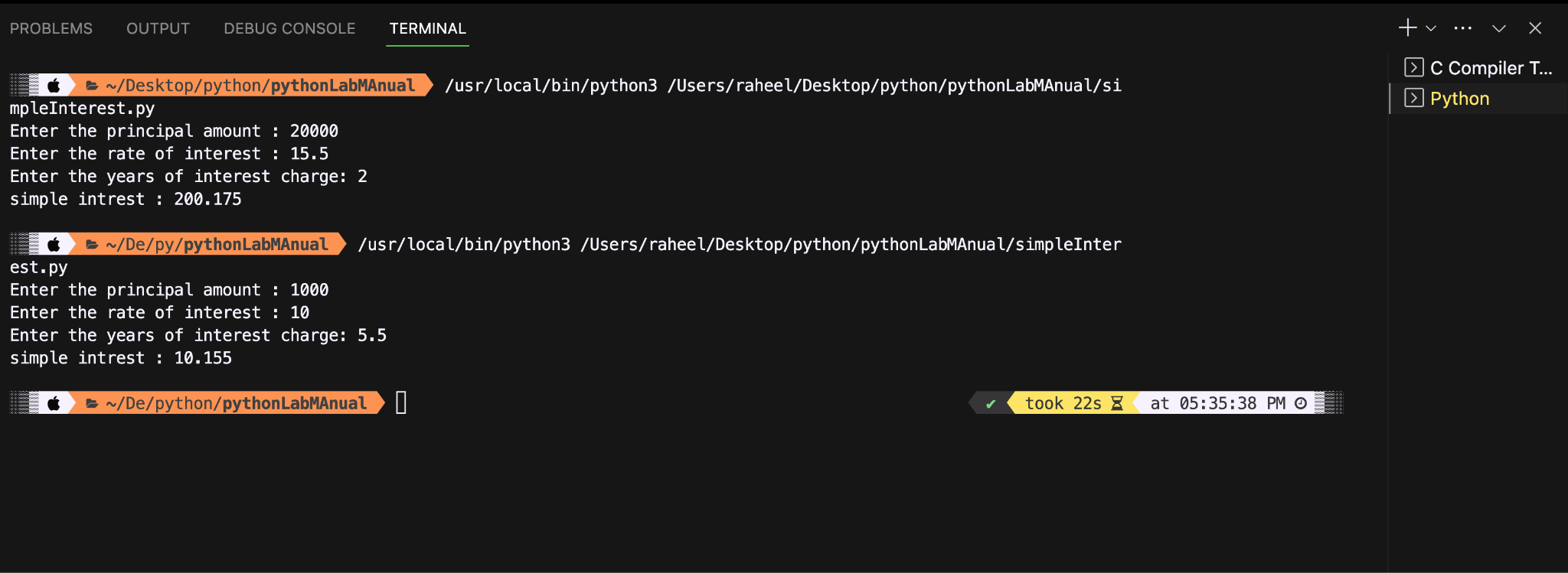
r=float(input("Enter the rate of interest : "))

t=float(input("Enter the years of interest charge: "))

simple\_interest=(p+r+t)/100

print(f"simple intrest : {simple\_interest}")

**Output:(screenshot):**



**Conclusion:**

**The code uses the simple interest formula, which directly provides the interest amount received. The code is restricted only on basis of raw user inputs, which means if the user is unaware of any of the three values (p,r,t), the code won’t be of their use.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.2**

**Title: Simple Interest**

**Theory:**  The code below takes basic arithemtic operations and relational operations

**Code:**

num1 = float(input("Enter the first number: "))

num2 = float(input("Enter the second number: "))

# Arithmetic operations:

print(f"Arithmetic Operations:")

print(f"{num1} + {num2} = {num1 + num2}") #Add.

print(f"{num1} - {num2} = {num1 - num2}") #Sub.

print(f"{num1} \* {num2} = {num1 \* num2}") #Mul.

print(f"{num1} / {num2} = {num1 / num2}") #Div.

print(f"{num1} % {num2} = {num1 % num2}") #Rem.

# Relational operations:

print("\nRelational Operations:") #EXTRA LINE

print(f"{num1} == {num2}: {num1 == num2}") #equality

print(f"{num1} != {num2}: {num1 != num2}") #non-equality

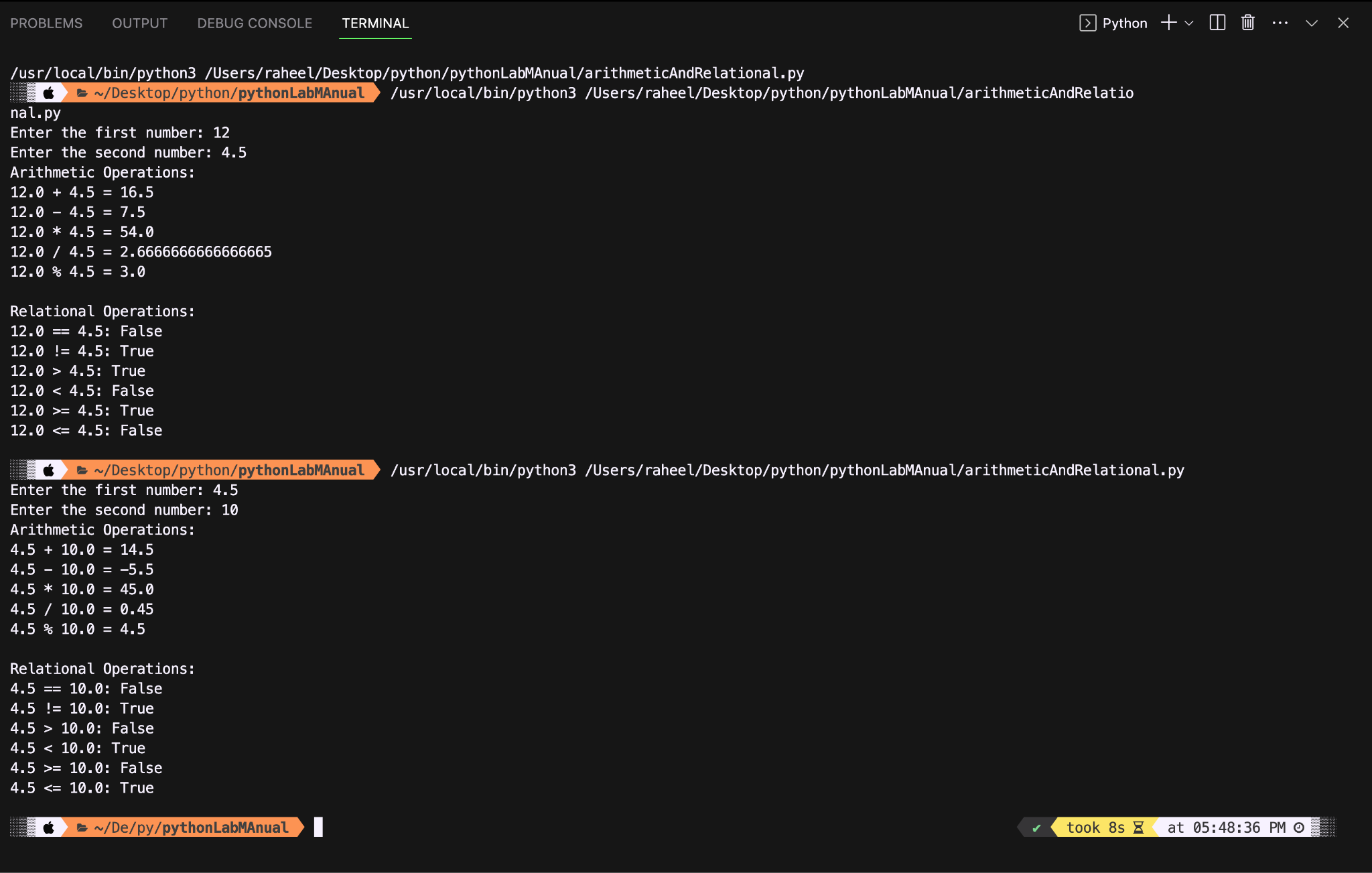
print(f"{num1} > {num2}: {num1 > num2}") #greater than

print(f"{num1} < {num2}: {num1 < num2}") #lesser than

print(f"{num1} >= {num2}: {num1 >= num2}") #greater than equal to

print(f"{num1} <= {num2}: {num1 <= num2}") #lesser than equal to

**Output:(screenshot):**



**Conclusion:**

**The program successfully handles arithmetic calculations and logics of relational operations.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.3**

**Title: Even or Odd**

**Theory:**  The code bellow uses simple remainder operator with 2 being as an operand to find if number is even or odd

**Code:**

n=int(input("enter a number: "))

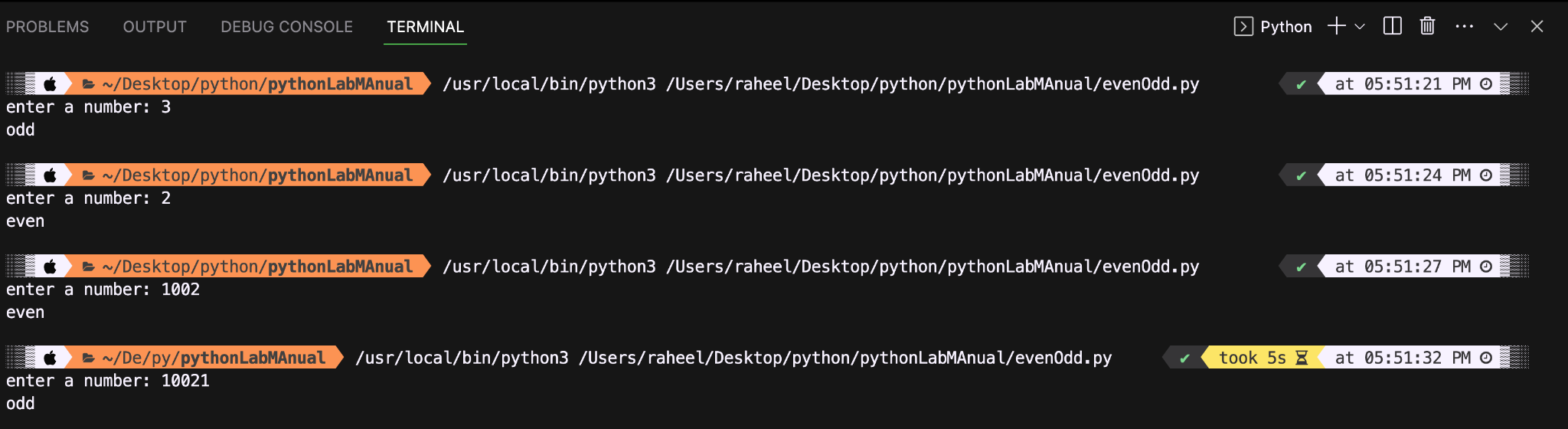
if(n%2==0):

print("even")

else:

print("odd")

**Output:(screenshot):**



**Conclusion:**

**The code successfully finds if a number is even or odd using remainder operator.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.4**

**Title: Natural numbers and sum**

**Theory:**  The code below gives first ‘n’ natural numbers and prints the sum of those numbers as well. This code uses a simple for loop starting from 1 till the value of n (inclusively).

**Code:**

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

sums = 0 #sum is a keyword thus used variable sums.

print(f"First {n} natural numbers: ")

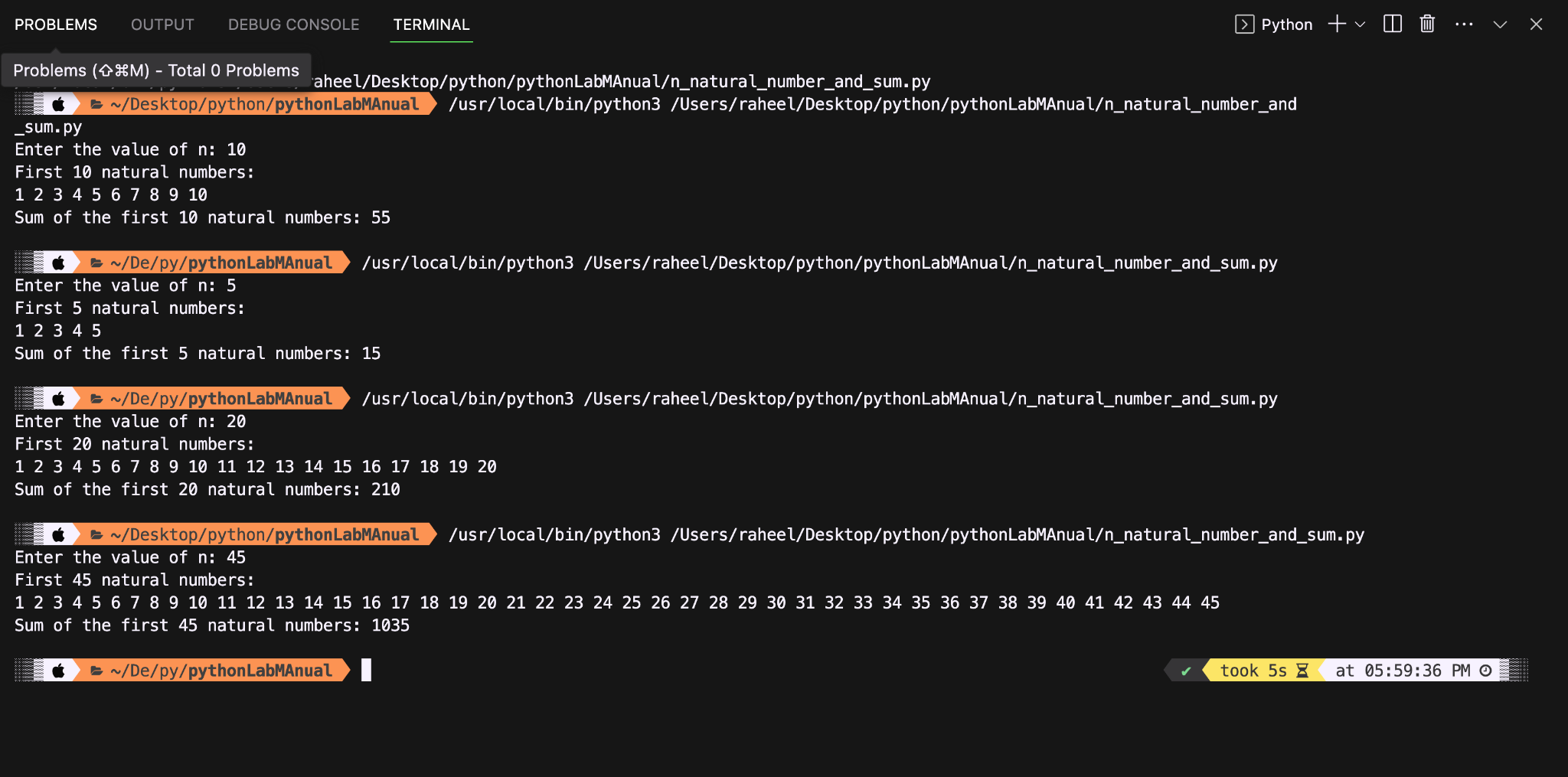
for i in range(1, n + 1):

print(i, end=" ")

sums += i

print(f"\nSum of the first {n} natural numbers: {sums}")

**Output:(screenshot):**



**Conclusion:**

The program effectively prints the first n natural numbers and it accurately calculates and displays the sum of the first ‘n’ natural numbers.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.5**

**Title: Vowel or not**

**Theory:** The code simply checks if the entered character by user is a vowel or not i.e. a,e,i,o, and u.

**Code:**

char = input("Enter a character: ")[0] #to take only the first character incase user inputs more than one.

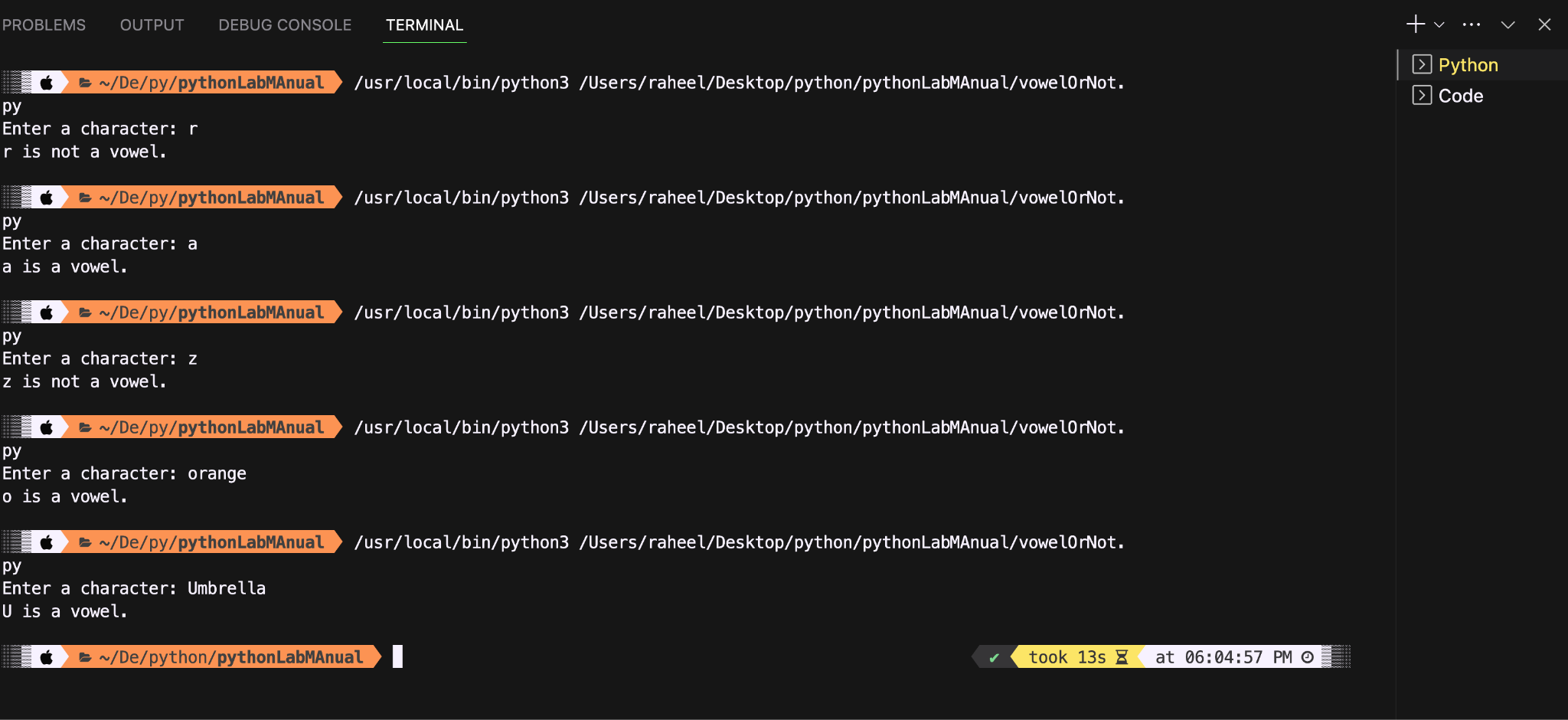
if char.lower() in ['a', 'e', 'i', 'o', 'u']:

print(f"{char} is a vowel.")

else:

print(f"{char} is not a vowel.")

**Output:(screenshot):**

****

**Conclusion:**

The program correctly identifies whether a character is vowel or not using a basic if-else statement and avoids case-sensitivity using lower() function.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.6**

**Title: Armstrong**

**Theory:** The code below simply finds if a number is armstrong or not. It converts the user input to string so that each digit can then be extracted. The code also uses composition to decrease the number of lines of code.

**Code:**

num = *int*(input("Enter a number: "))

num\_str = *str*(num)

num\_digits = len(num\_str)

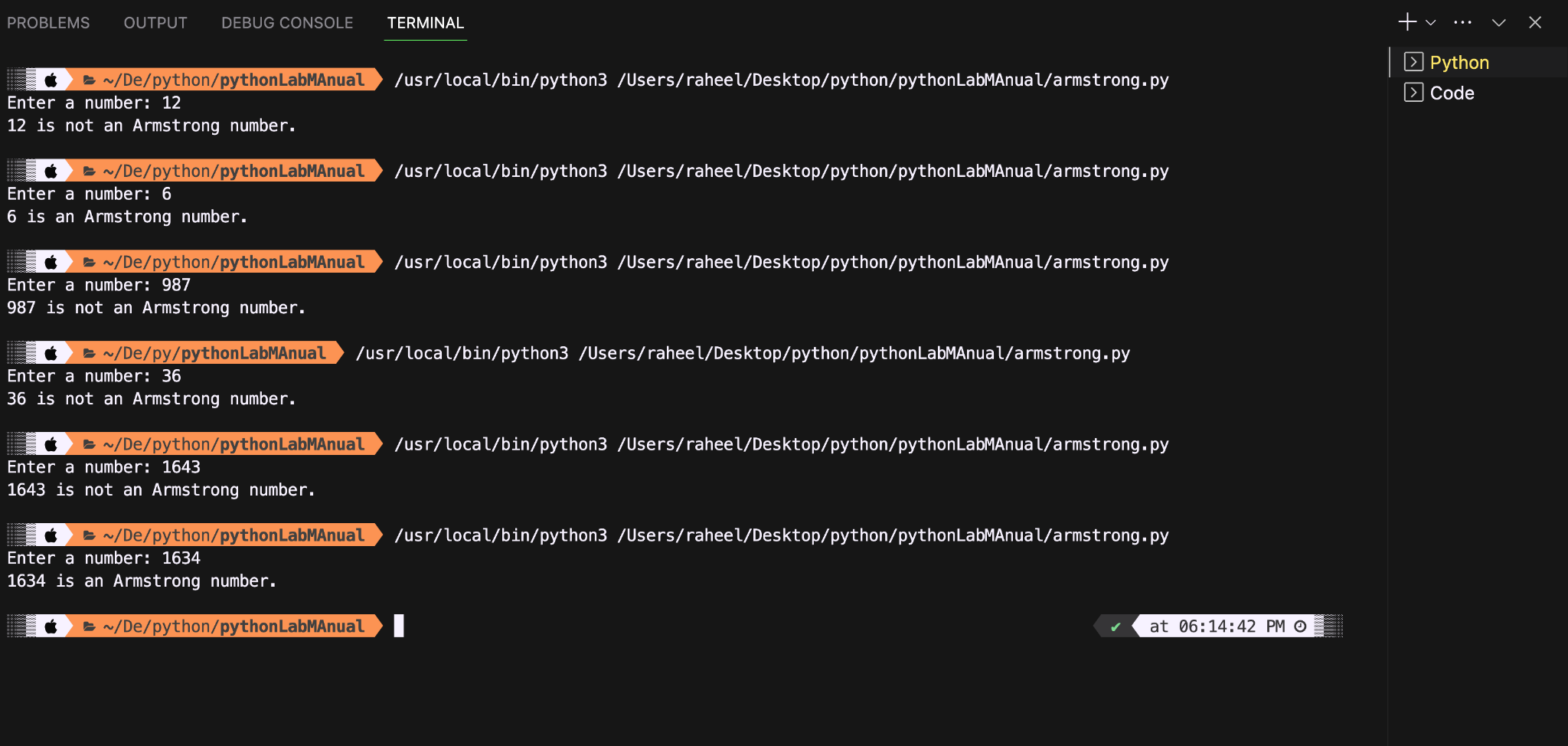
armstrong\_sum = sum(*int*(digit) \*\* num\_digits for digit in num\_str) #Simple composition is used where each digit has first the exponent of number of digit and then added with the next in line.

if armstrong\_sum == num:

print(*f*"{num} is an Armstrong number.")

else:

print(*f*"{num} is not an Armstrong number.")

**Output:(screenshot):**

**Conclusion:**

The program accurately determines if the given number is an Armstrong

Number. It further uses a loop to extract digits, calculates the sum, and validates against the original number.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.7**

**Title: Factorial**

**Theory:** Take user input, run through an iterator and multiply a number till iteration value, where final value of iteration is same as input number. Start iteration from 1.

**Code:**

**num = int(input("Enter a number: "))**

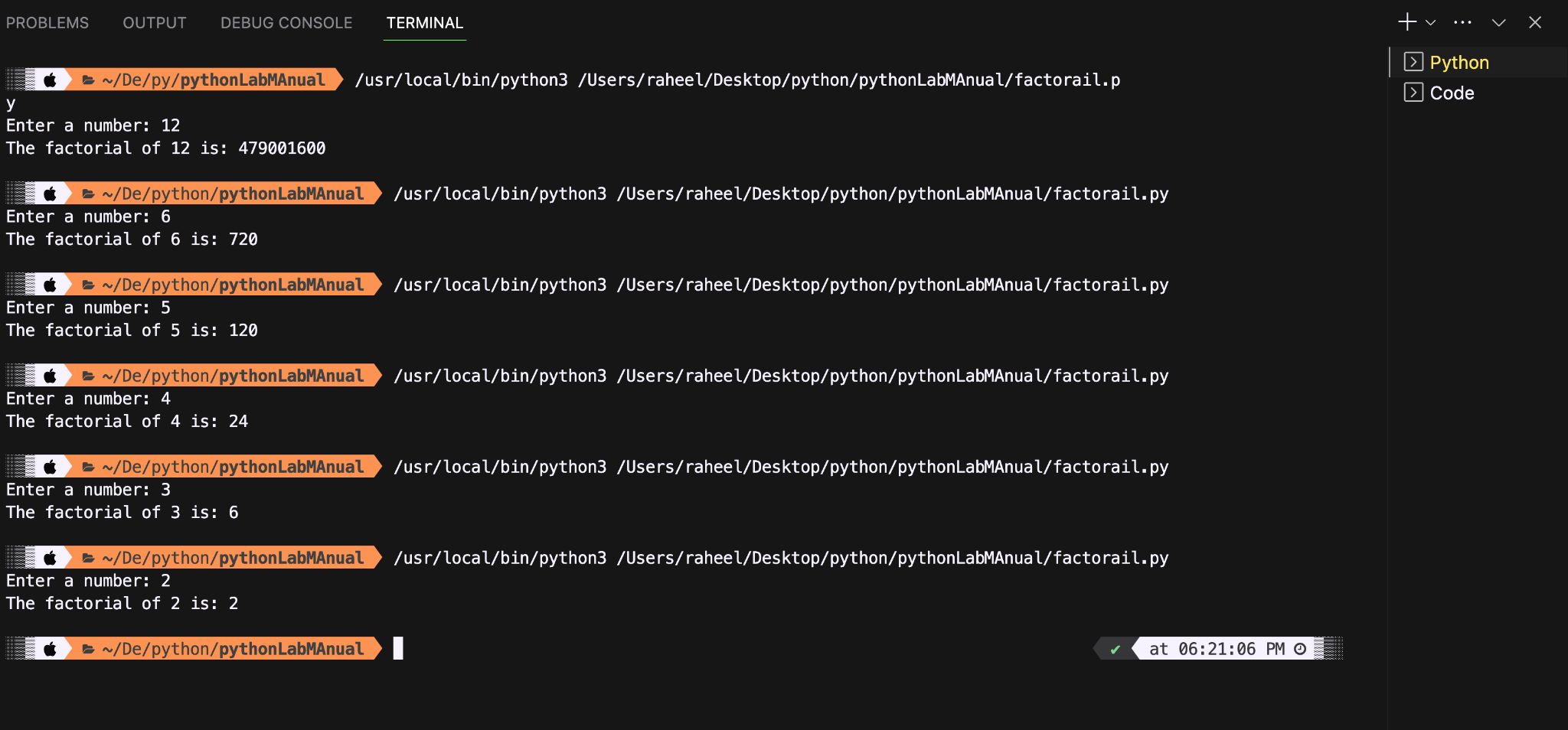
**factorail = 1 #if starts from 0 then all values stay 0**

**for i in range(1, num + 1):**

**factorail \*= i**

**print(f"The factorial of {num} is: {factorail}")**

**Output:(screenshot):**

****

**Conclusion:**

The program successfully calculates the factorial of the given number. It uses a for loop to iterate and multiply, providing an accurate factorial result.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 1.8**

**Title: Patterns**

**Theory:** The following code prints three different patterns based on pattern logic with respect to user input.

**Code:**

**n=int(input("Enter the no. : "))**

**#PATTERN 1**

**for i in range (1,n+1,1):**

**for j in range(i):**

**print(" \* ",end=" ")**

**print(" ")**

**print("\n\n\n") #For distinction of differnet patterns**

**#PATTERN 2**

**for i in range (1,n+1,1):**

**for j in range(i):**

**print( i ,end=" ")**

**print(" ")**

**print("\n\n\n") #For distinction of differnet patterns**

**#PATTERN 3**

**for i in range(1,n+1):**

**for j in range(n,i,-1):**

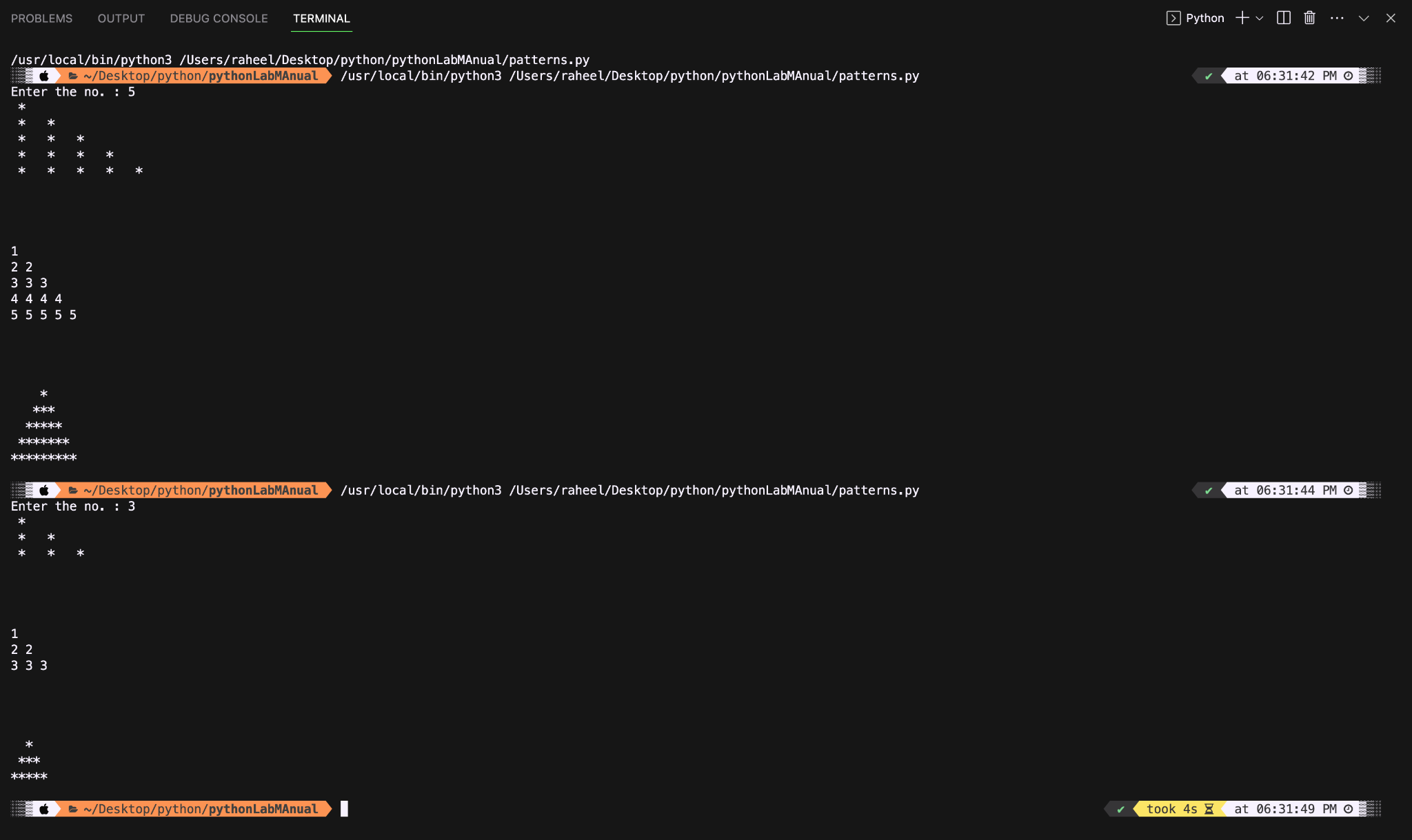
**print(" ",end="")**

**for k in range(2\*i-1):**

**print("\*",end="")**

**print("") #change in line**

**Output:(screenshot):**

****

**Conclusion:**

The code utilizes nested loops to generate three distinct patterns based on user input 'n': ascending '\*' symbols, ascending numbers, and a pyramid of '\*' symbols. It showcases the versatility of nested loops for creating varied visual patterns.