**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.1**

**Title: Simple Interest**

**Theory:**  A list is created to store BRICS nations: Brazil, Russia, India, China, and South Africa.

**Code:**

brics\_countries = ["Brazil", "Russia", "India",

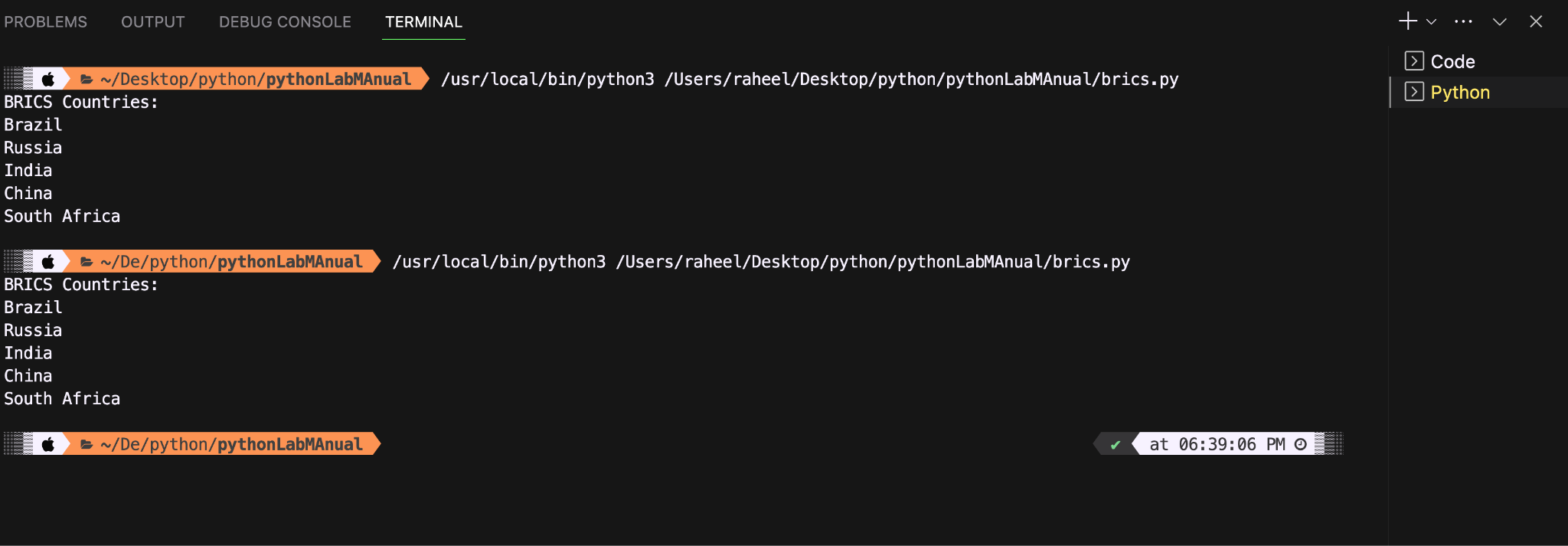
"China", "South Africa"]

print("BRICS Countries:")

for country in brics\_countries:

print(country)

**Output:(screenshot):**



**Conclusion:**

**The code has a hardcoded list of countries in the brics list. And prints it properly with help of an iterator.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.2**

**Title: Reversing of list**

**Theory:**  The code below uses reverse function and slicing method to reverse a list

**Code:**

sample\_list = [1, 2, 3, 4, 5]

reversed\_list1 = list(sample\_list) #stores the sample list to a new variable reversed list

# Method 1: Using reverse() function

reversed\_list1.reverse()

# Method 2: Using slicing

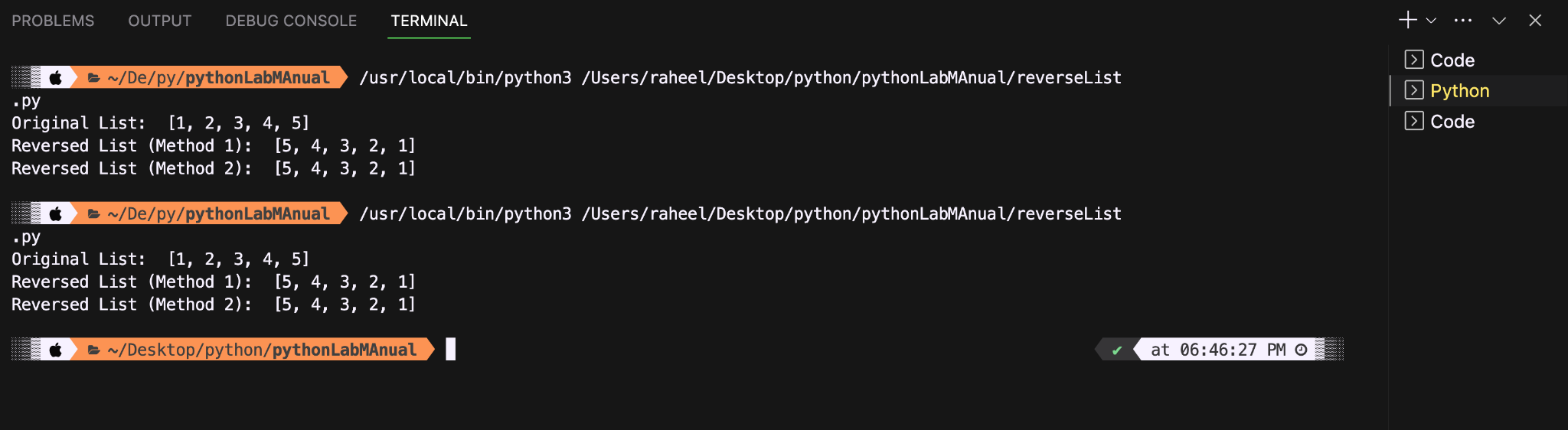
reversed2 = sample\_list[::-1]

print("Original List: ", sample\_list)

print("Reversed List (Method 1): ",reversed\_list1)

print("Reversed List (Method 2): ", reversed2)

**Output:(screenshot):**



**Conclusion:**

**The program effectively reverses the list using two different methods.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.3**

**Title: Tuple of username and domain from an email**

**Theory:** The code should extract the domain and username from a given email. We can thus use the split function with ‘@’ as the parameter.

**Code:**

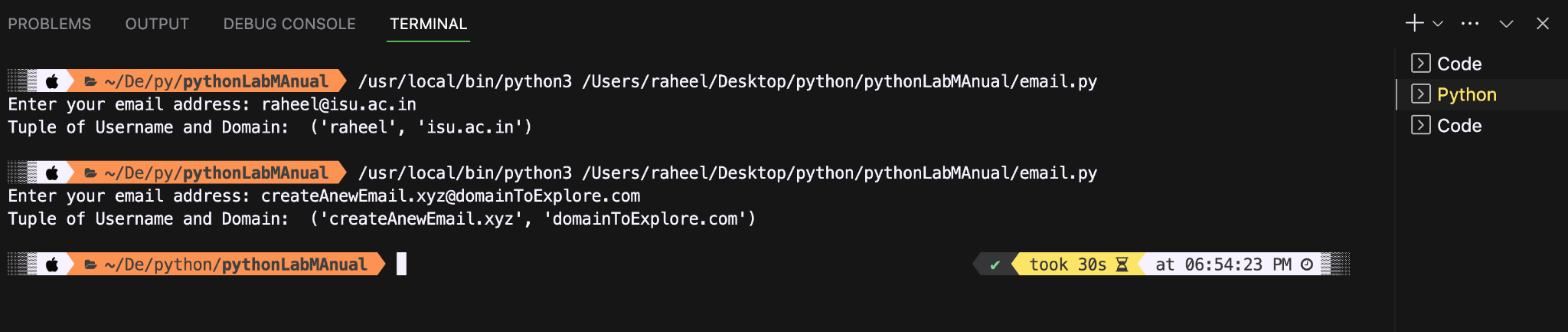
email = input("Enter your email address: ")

username, domain = email.split('@')

email\_tuple = (username, domain)

print("Tuple of Username and Domain: ",email\_tuple)

**Output:(screenshot):**



**Conclusion:**

**The code successfully forms a tuple of username and domain using basic string manipulation.**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.4**

**Title: Cube from a given list**

**Theory:** The code takes a list of numbers and creates a tuple list that has the number and its corresponding cubic value.

**Code:**

c = [2, 3, 4, 5, 6, 7, 8, 9]

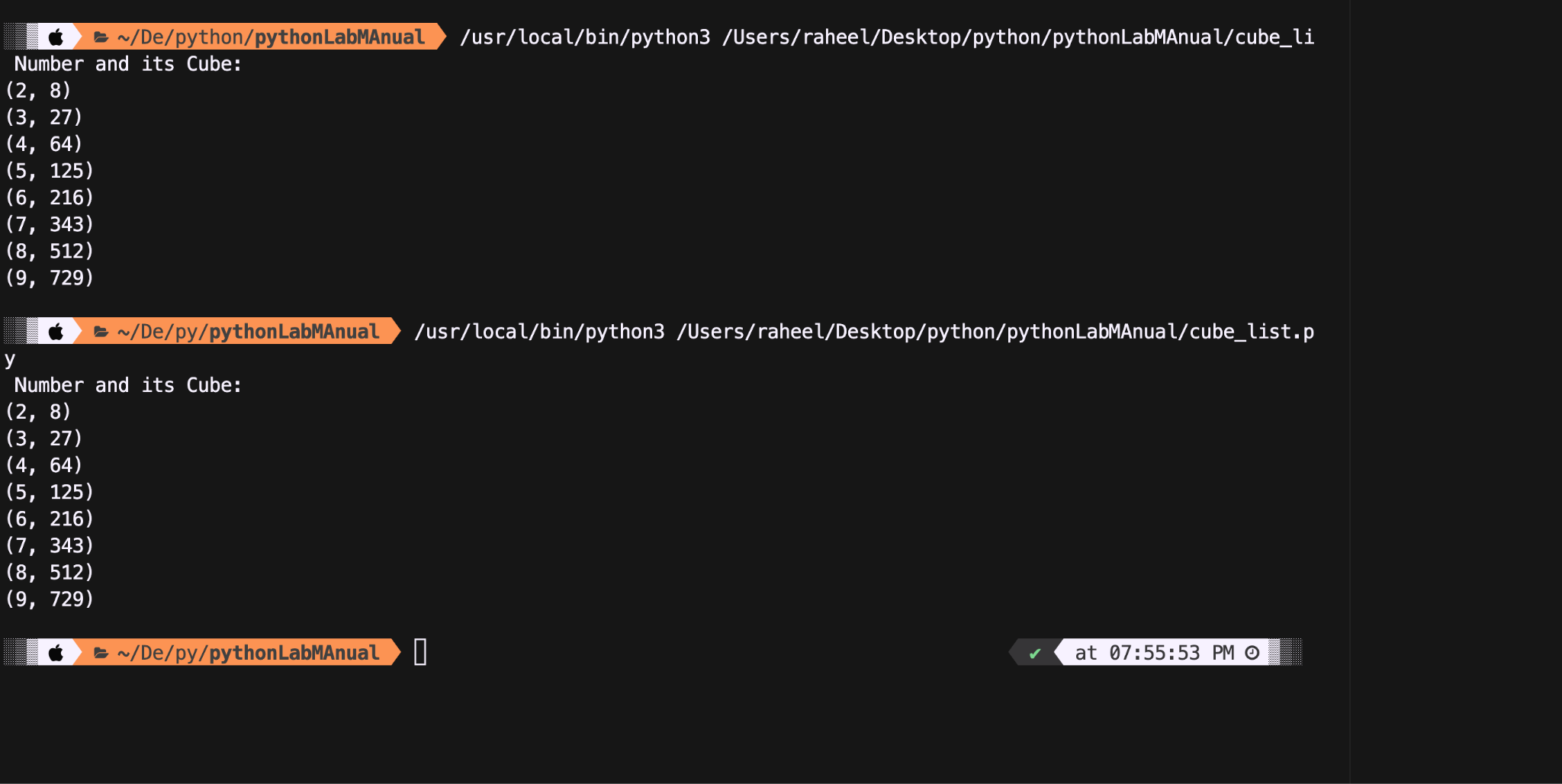
tuple\_list = [(num, num\*\*3) for num in c]

print(" Number and its Cube:")

for i in tuple\_list:

print(i)

**Output:(screenshot):**



**Conclusion:**

**The code successfully creates a tuple of lists where each list has the corresponding number from the original list and its cube. The output remains unchanged as values for which cube is to be calculated is hard coded (given in question)**

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.5**

**Title: Equality of dictionaries using == operator.**

**Theory:** The program compares two dictionaries using the == operator. Where both key-value pairs are checked.

**Code:**

d1 = {'one': 1, 'two': 2, 'three': 3}

d2 = {'one': 1, 'two': 2, 'three': 3}

if d1 == d2:

print("The dictionaries are equal.")

else:

print("The dictionaries are not equal.")

**Output:(screenshot):**

****

**Conclusion:**

The code uses two hard coded dictionaries named as d1 and d2, and successfully compares both using the ‘==’ operator and prints if those two are equal or not. Thus even output remains unchanged.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.6**

**Title: Armstrong**

**Theory:** The following code has a dictionary of cube of odd numbers in the range. Where range is specified by user. Composition can be used to successfully create such dictionary, which further decreases the length of code.

**Code:**

start\_range = int(input("Enter the start of the range: "))

end\_range = int(input("Enter the end of the range: "))

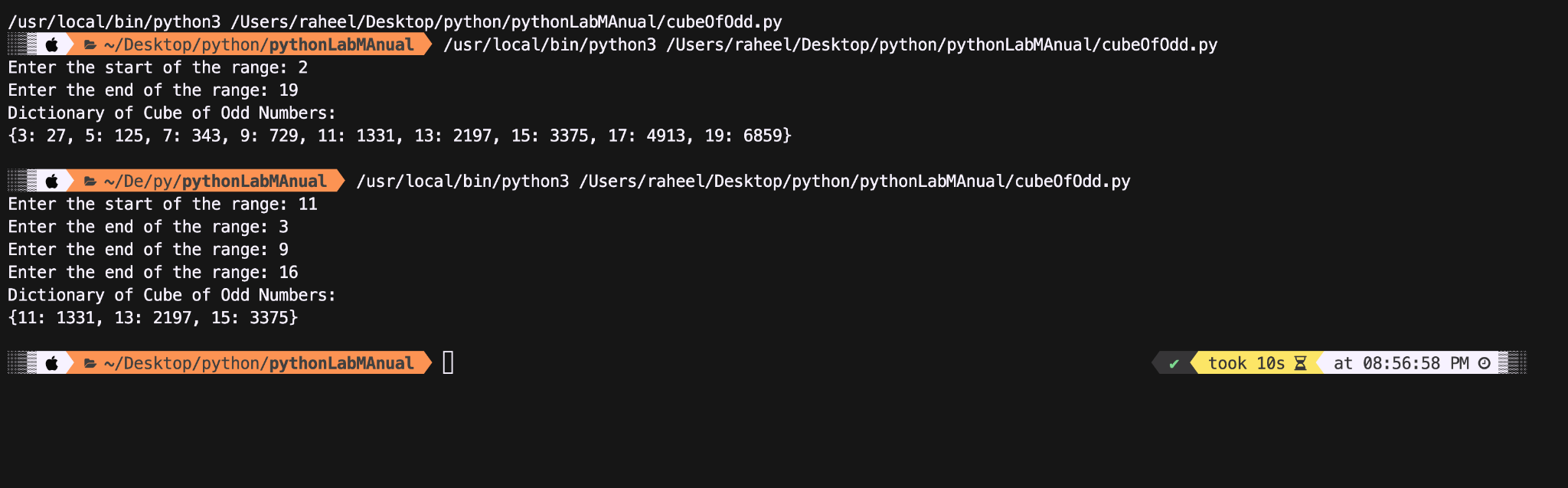
while end\_range<start\_range:

end\_range = int(input("Enter the end of the range: "))

odd\_cubes\_dict = {num: num\*\*3 for num in range(start\_range, end\_range + 1) if num % 2 != 0}

print("Dictionary of Cube of Odd Numbers:")

print(odd\_cubes\_dict)

**Output:(screenshot):**

**Conclusion:**

The program effectively executes the logic to create a dictionary where key is the odd number corresponding to its value pair of cube of that number.

**Name of Student: Raheel Kotwal**

**Roll Number: 45**

**Experiment Number: 2.7**

**Title: List Manipulation**

**Theory:** The code below manipulates a list (hard coded) in different ways and prints it. The manipulation is done using most predefined methods such as: append(), sort(), reverse() and extend(). The code further utilizes splicing methods to print certain desired outcomes.

**Code:**

**a = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]**

**print("\n\nList: ", a)**

**print("\n4th Element of the List: ", a[3])**

**print("\nList from 0th to 4th Index: ", a[0:5])**

**print("\nList from -7th to 3rd Element: ", a[-7:4])**

**a.append(110)**

**print("\nv. List after Appending 110:", a)**

**a.sort()**

**print("\nvi. Sorted List:", a)**

**popped\_element = a.pop()**

**print("\nvii. Popped Element:", popped\_element)**

**print(" List after Popping:", a)**

**specified\_element = 60**

**a.remove(specified\_element)**

**print("\nviii. List after Removing",**

**specified\_element, ":", a)**

**index\_to\_insert = 2**

**element\_to\_insert = 15**

**a.insert(index\_to\_insert, element\_to\_insert)**

**print("\nix. List after Inserting ", element\_to\_insert, " at index [", index\_to\_insert,"]: ", a)**

**element\_to\_count = 20 #Occurrence of specific element**

**count = a.count(element\_to\_count)**

**print("\nx. Count of", element\_to\_count, "in the List: ", count)**

**extended\_list = [120, 130, 140]**

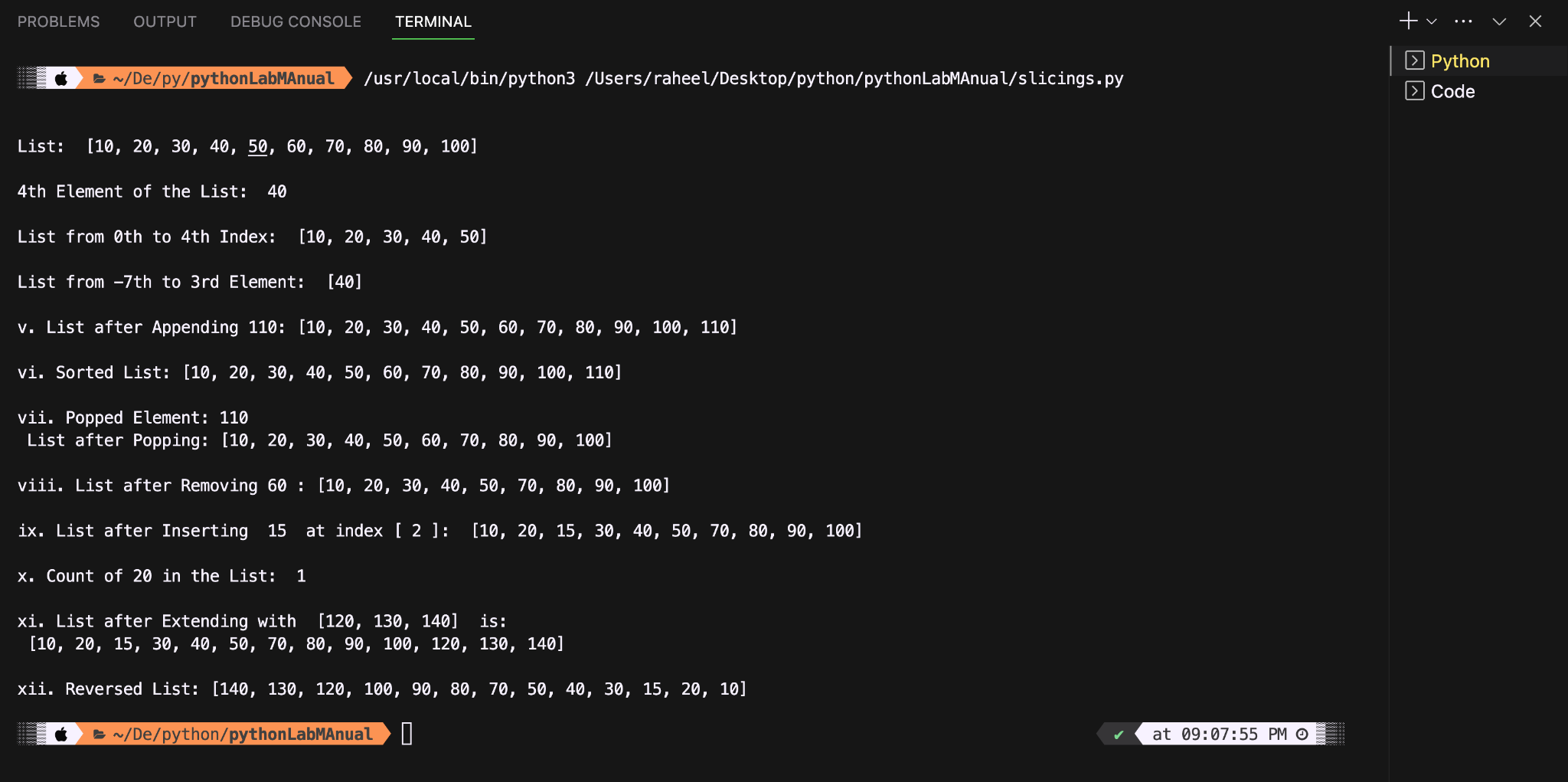
**a.extend(extended\_list)**

**print("\nxi. List after Extending with ", extended\_list, " is: \n", a)**

**a.reverse()**

**print("\nxii. Reversed List:", a)**

**Output:(screenshot):**

****

**Conclusion:**

Each operation is executed successfully, demonstrating diverse list manipulation techniques.