**PRACTICAL 1**

**AIM:**

Create a program that asks the user to enter their name and their age. Printout a message addressed to them that tells them the year that they will turn 100 years old.

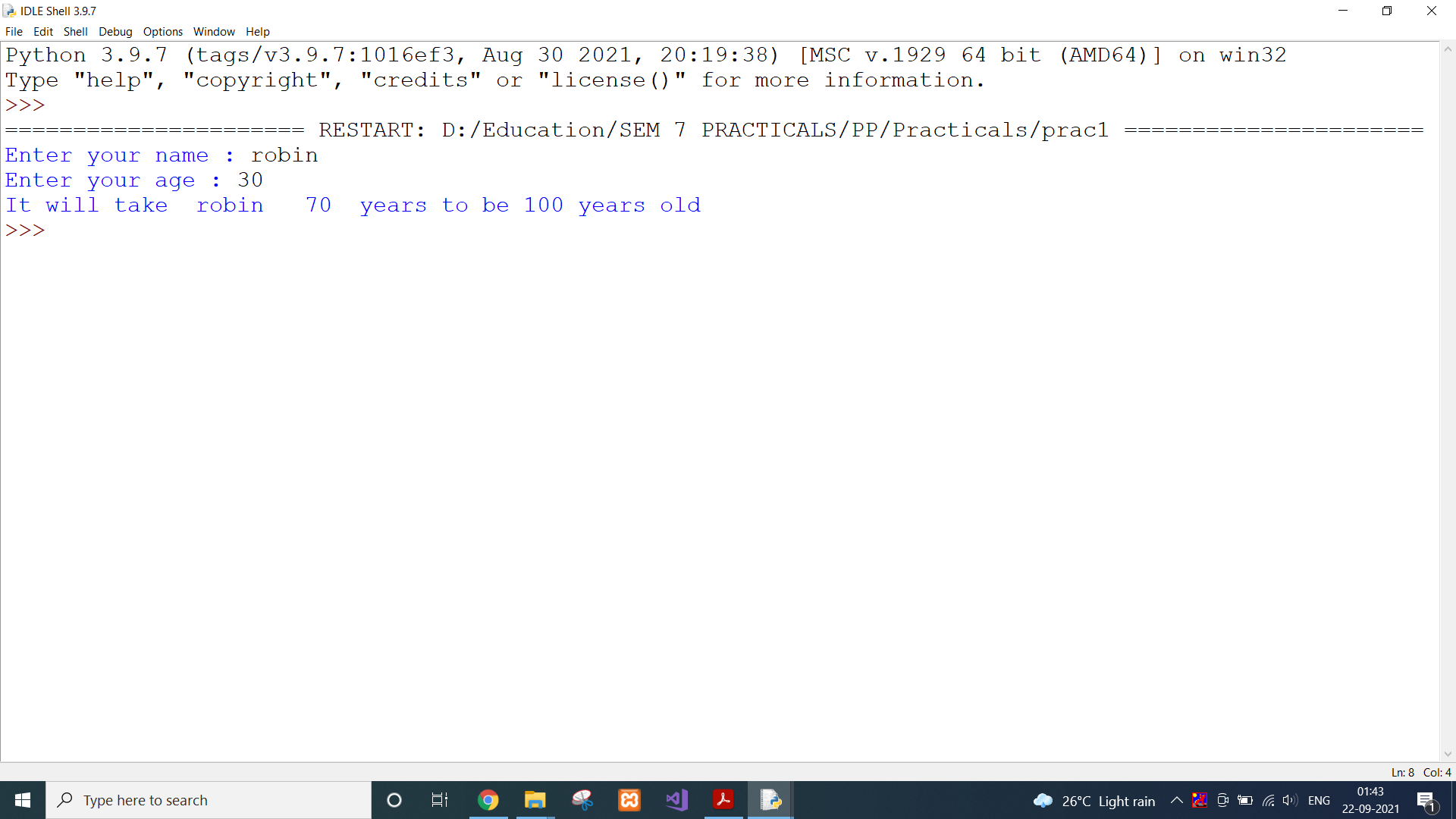
**CODE:**

name = input("Enter your name : ")

age = int(input("Enter your age : "))

print("It will take ",name+str(100-age)," to turn 100")

**OUTPUT:**



**CONCLUSION:**

In this practical, we got familiar with Python and wrote our first program which takses input from user and writes the output on the screen.

**PRACTICAL 2.1**

**AIM:**

Ask the user for a number. Depending on whether the number is even or 2 odd, print out an appropriate message to the user. Hint: how does an even / odd number react differently when divided by 2?

**CODE:**

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

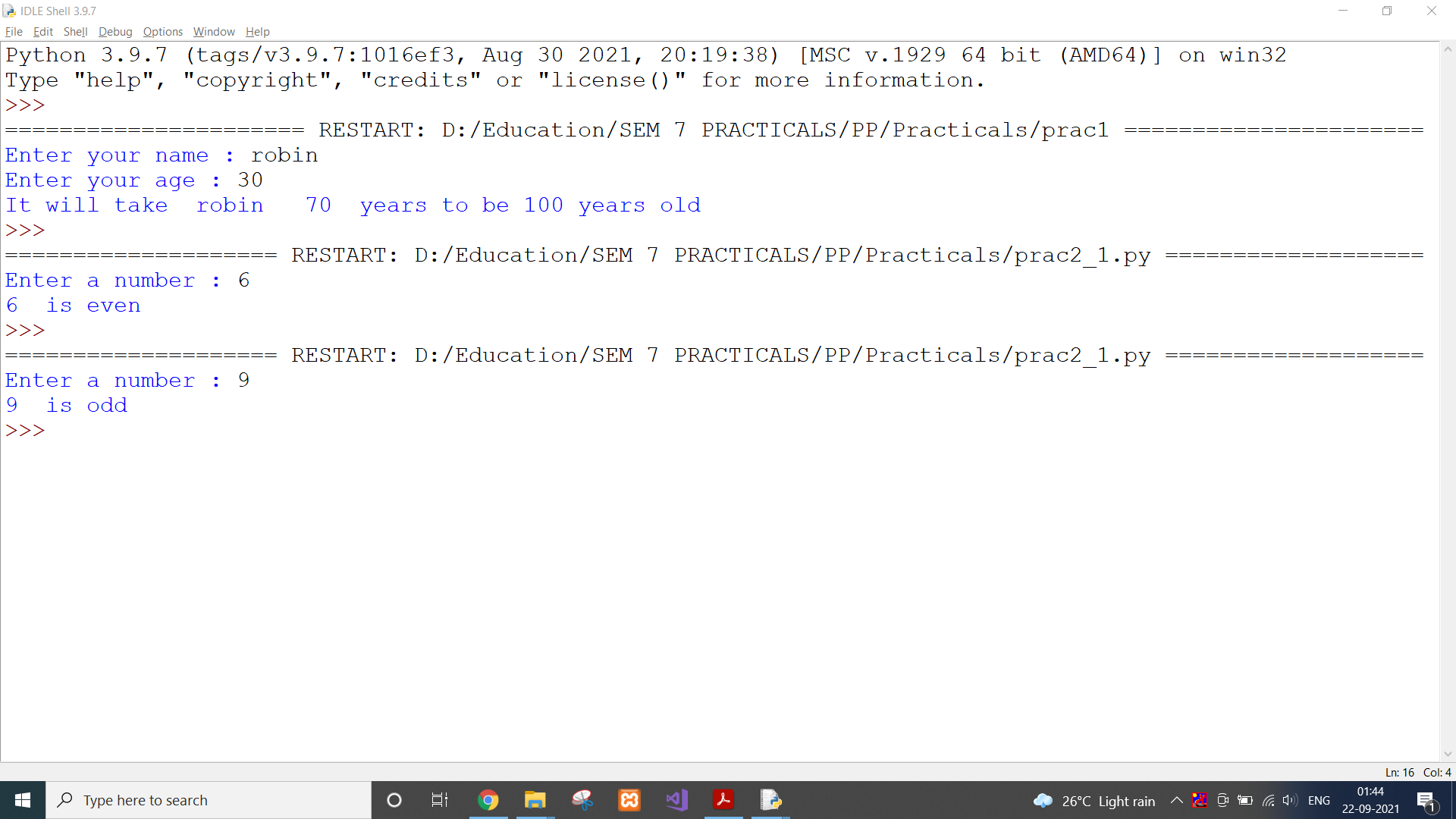
if(number%2==0):

print(number," is even")

else:

print(number," is odd")

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt about if else statement.

**PRACTICAL 2.2**

**AIM:**

Take a list, say for example this one:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89], and write a program that prints out all the elements of the list that are less than 5.

**CODE:**

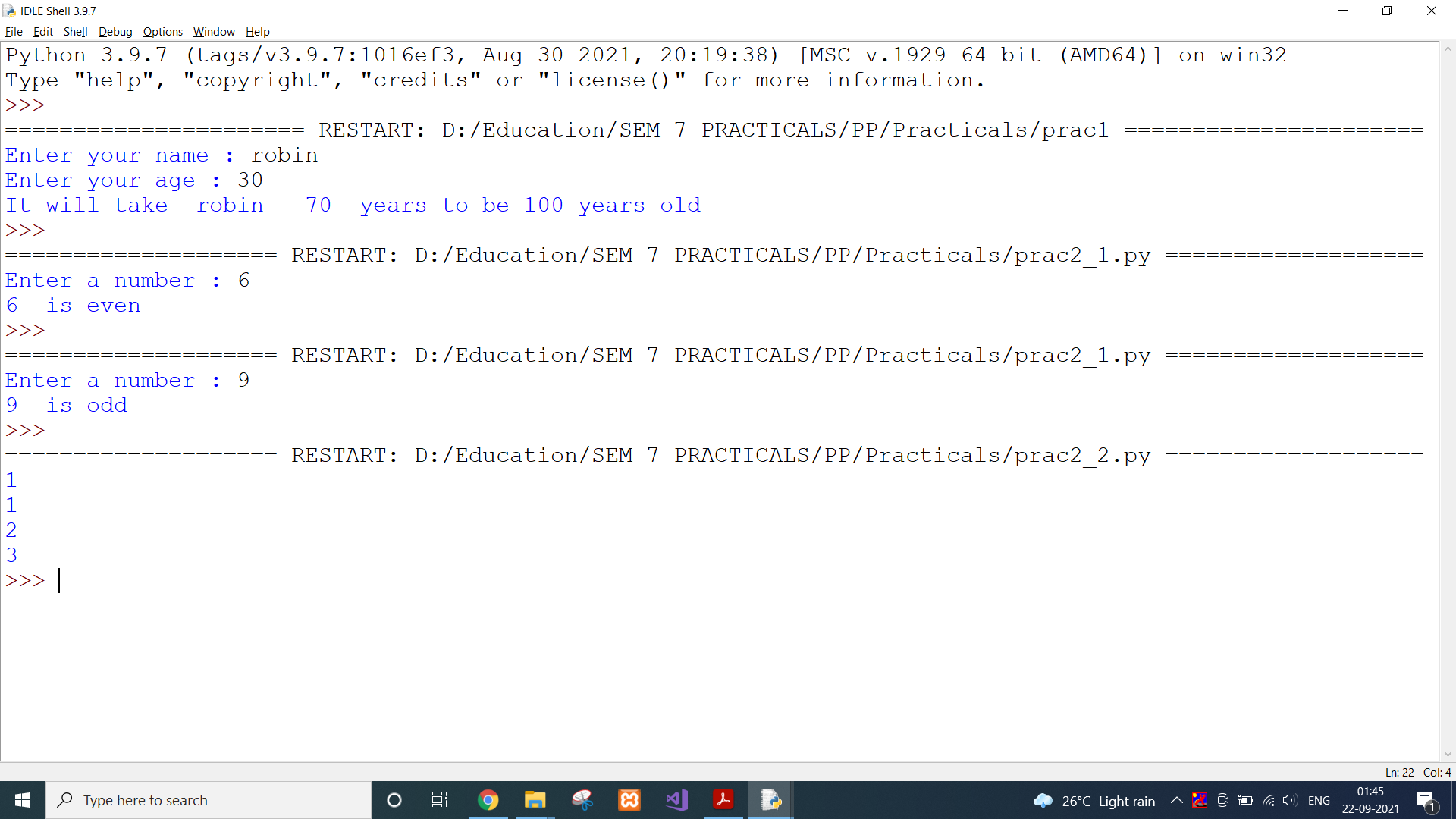
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

for i in a:

if(i < 5):

print(i)

**OUTPUT:**



**CONCLUSION:**

In this practical, we used for loop to loop through every element of list.

**PRACTICAL 3.1**

**AIM:**

Create a program that asks the user for a number and then prints out a list of all the divisors of that number. (If you don’t know what a divisor is, it is a number that divides evenly into another number. For example, 13 is a divisor of 26 because 26 / 13 has no remainder.)

**CODE:**

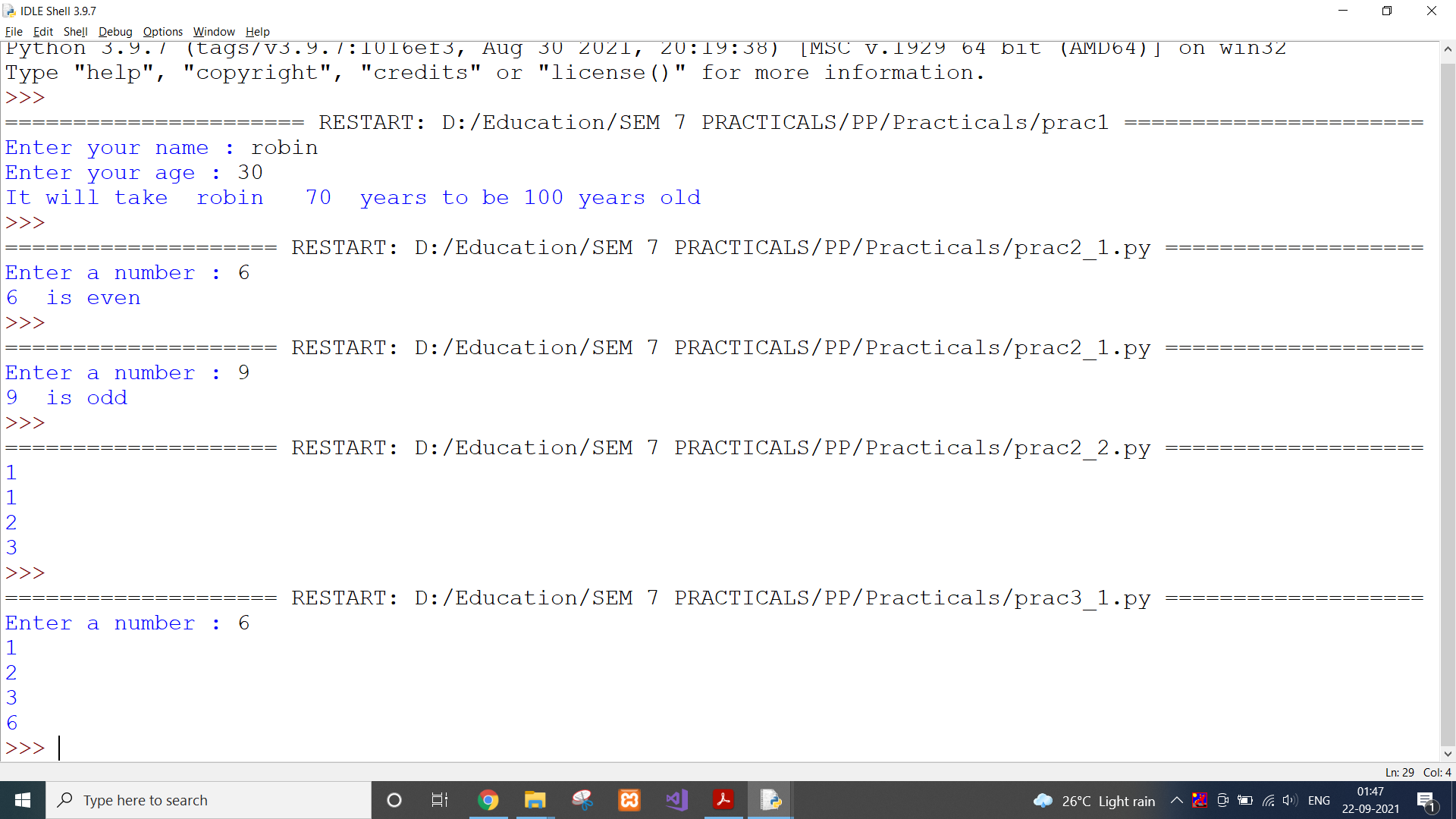
number = int(input("Enter a number : "))

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

if(number%i==0):

print(i)

**OUTPUT:**



**CONCLUSION:**

In this practical, we listed the factors of a number using loop and if condition.

**PRACTICAL 3.2**

**AIM:**

Take two lists, say for example these two:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes.

**CODE:**

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

temp\_list = []

for i in a:

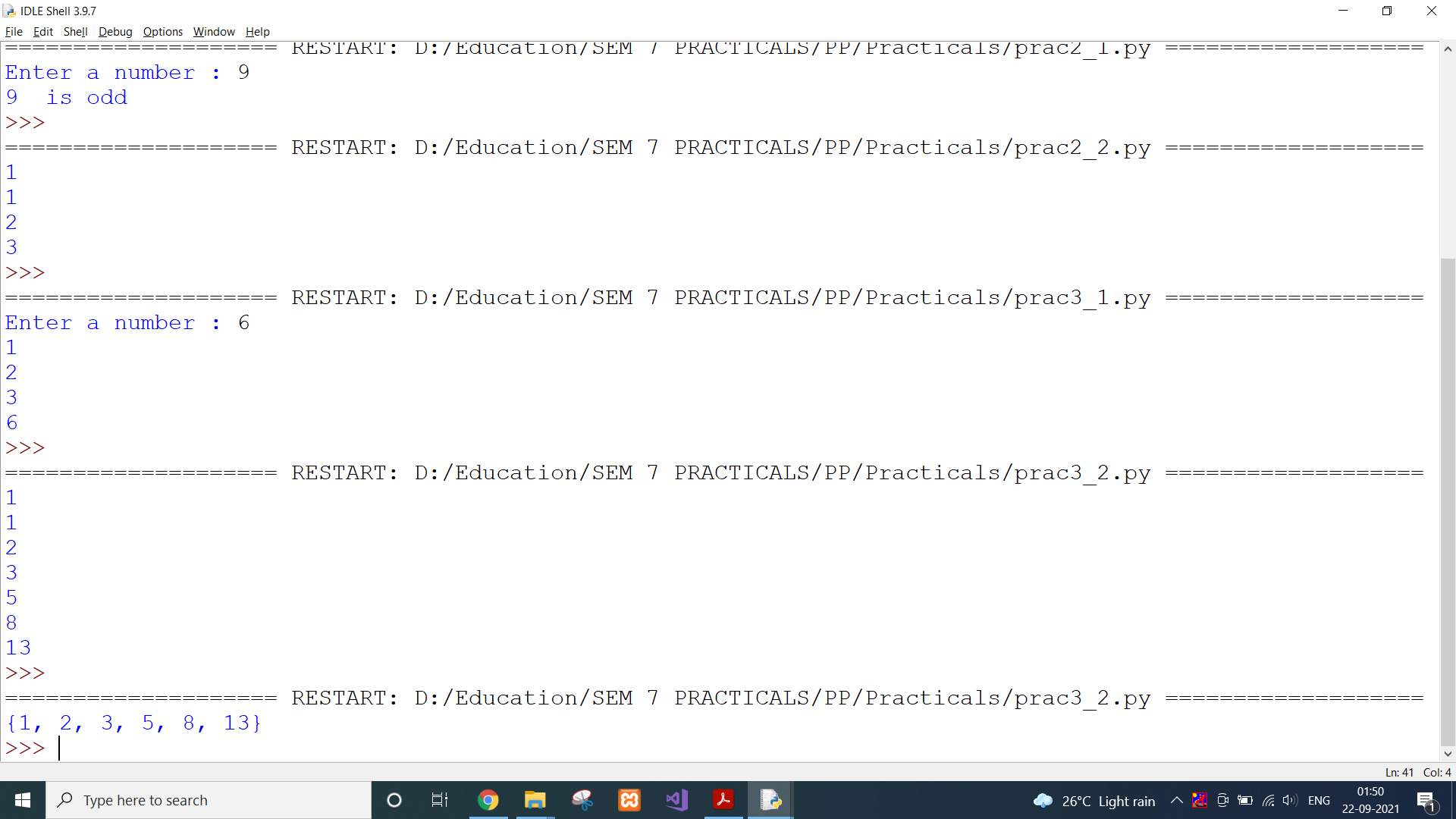
for j in b:

if(i==j):

temp\_list.append(i)

print(set(temp\_list))

**OUTPUT:**



**CONCLUSION:**

In this practical, we used list and its method append to create sublist from a given list.

**PRACTICAL 3.3**

**AIM:**

Ask the user for a string and print out whether this string is a palindrome or not. (A palindrome is a string that reads the same forwards and backwards.)

**CODE:**

string = input("Enter a string : ")

rev\_string = string[::-1]

if(string == rev\_string):

print(string," is palindrome")

else:

print(string," is not palindrome")

**OUTPUT:**



**CONCLUSION:**

In this practical, we used list slicing to check if string is palindrome or not.

**PRACTICAL 4.1**

**AIM:**

Let’s say I give you a list saved in a variable: a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]. Write one line of Python that takes this list and makes a new list that has only the even elements of this list in it.

**CODE:**

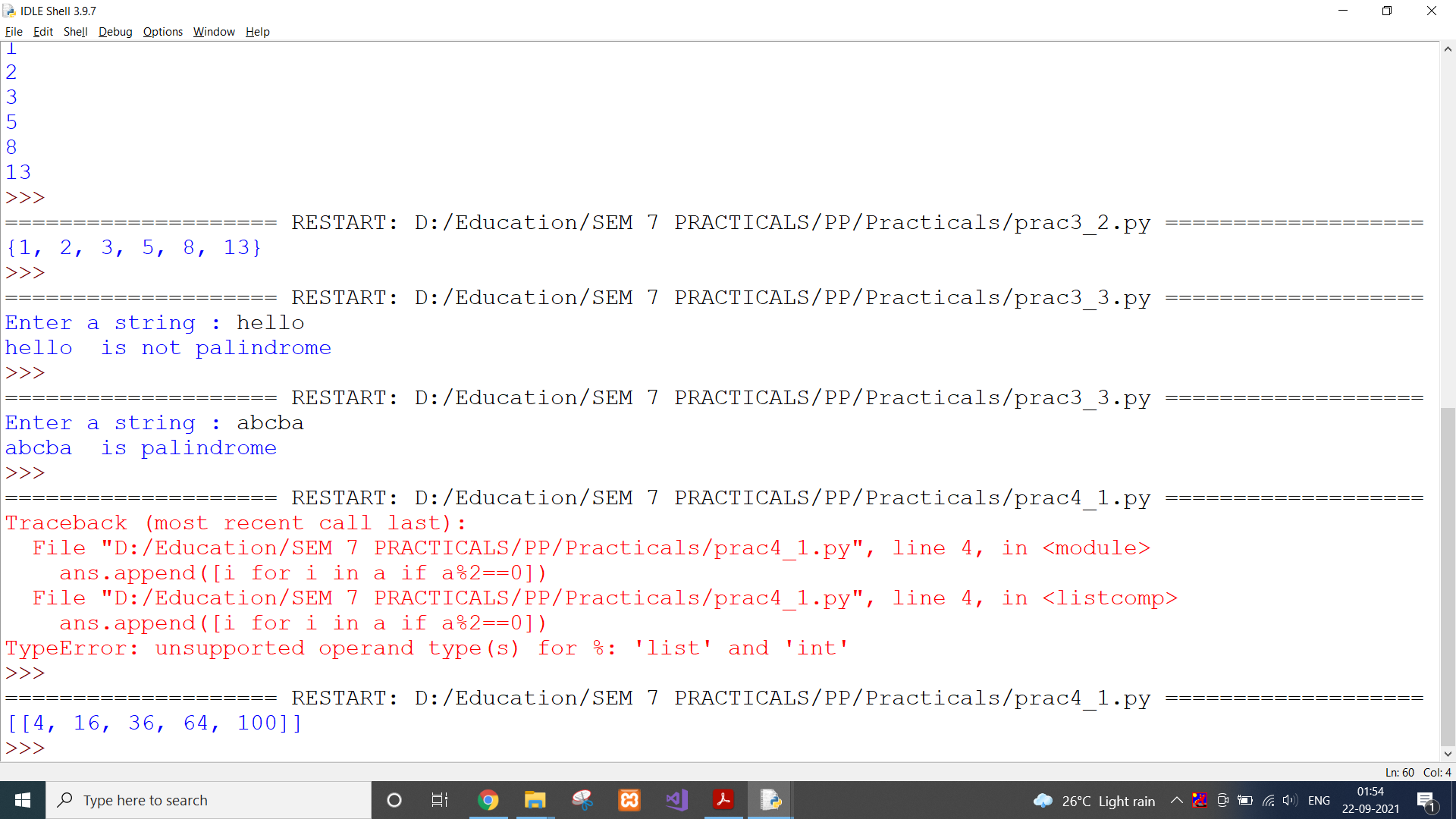
a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

ans = []

ans.append([i for i in a if i%2==0])

print(ans)

**OUTPUT:**



**CONCLUSION:**

In this practical, we used list and if condition to create new list with only even numbers.

**PRACTICAL 4.2**

**AIM:**

Make a two-player Rock-Paper-Scissors game. (Hint: Ask for player plays (using input), compare them, print out a message of congratulations to the winner, and ask if the players want to start a new game)

Remember the rules:

Rock beats scissors, Scissors beats paper, Paper beats rock

**CODE:**

player1 = ""

player2 = ""

winner = ""

while(winner!="N" and winner!="n"):

print("Starting new game")

player1 = input("Player 1's input : ")

player2 = input("Player 2's input : ")

if(player1 == "rock" and player2 == "scissors"):

print("Player1 won")

elif(player1 == "scissors" and player2 == "paper"):

print("Player1 won")

elif(player1 == "paper" and player2 == "rock"):

print("Player1 won")

elif(player2 == "rock" and player1 == "scissors"):

print("Player2 won")

elif(player2 == "scissors" and player1 == "paper"):

print("Player2 won")

elif(player2 == "paper" and player1 == "rock"):

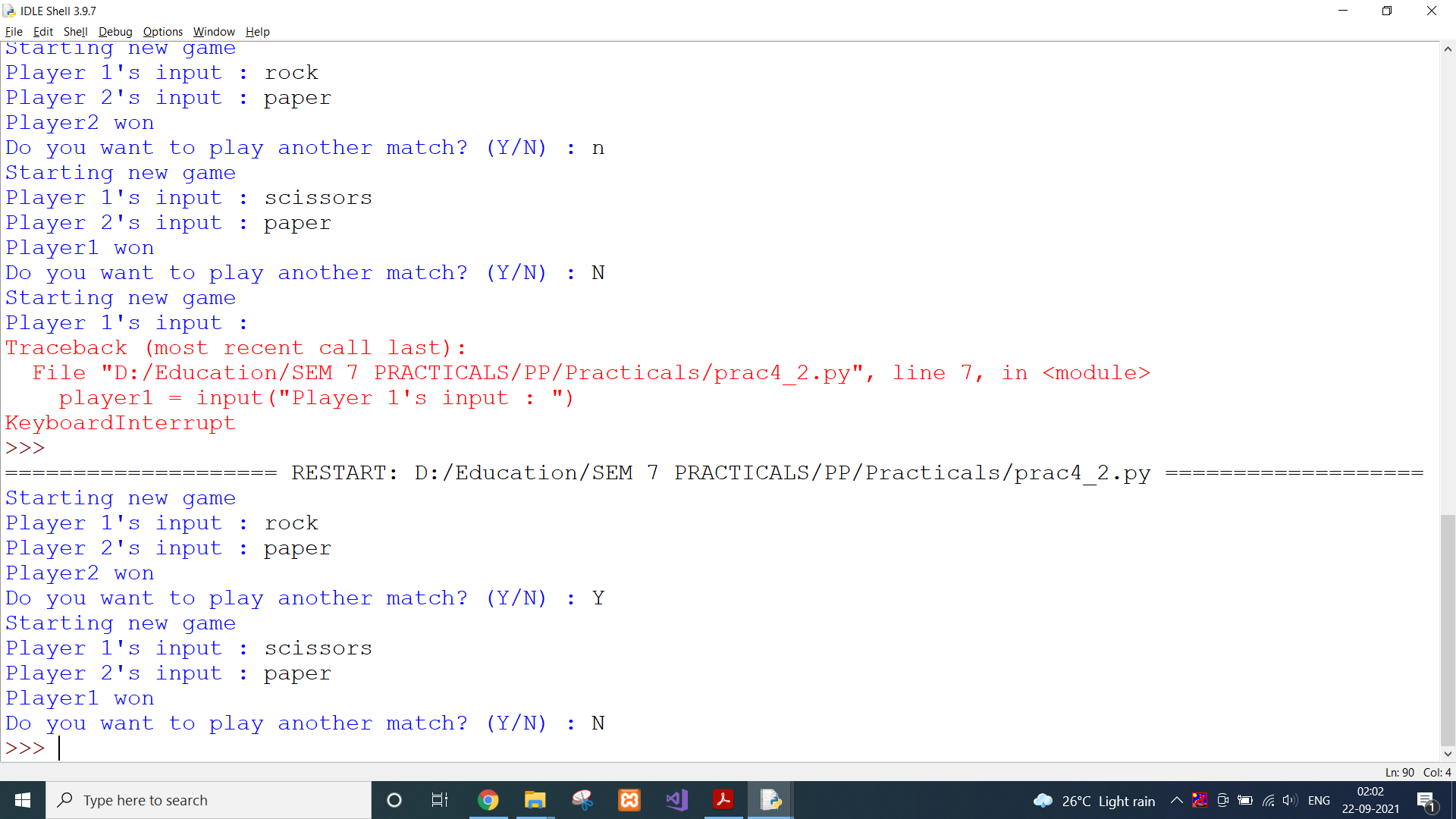
print("Player2 won")

else:

print("It's a draw")

winner = input("Do you want to play another match? (Y/N) : ")

**OUTPUT:**



**CONCLUSION:**

In this practical, we used all the concepts we learnt till now to create rock paper scissors game.

**PRACTICAL 4.3**

**AIM:**

Generate a random number between 1 and 9 (including 1 and 9). Ask the user to guess the number, then tell them whether they guessed too low, too high, or exactly right. (Hint: remember to use the user input lessons from the very first practical)

**CODE:**

from random import randint

num = randint(0,9)

user\_num = int(input("Guess the number : "))

if(user\_num < num):

print("Your value is too low")

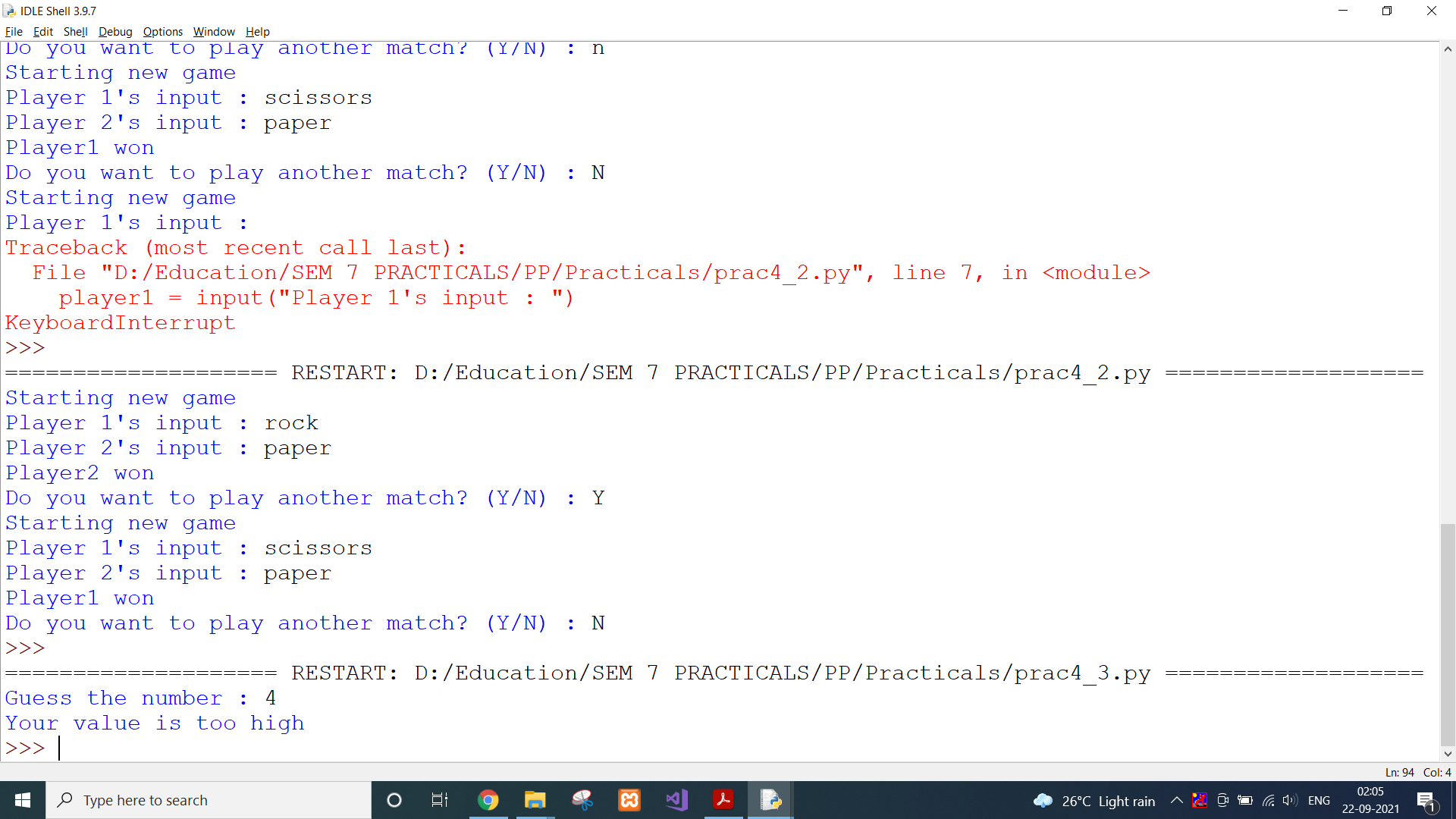
elif(user\_num > num):

print("Your value is too high")

else:

print("You guessed it!")

**OUTPUT:**



**CONCLUSION:**

In this practical, we imported python library random to generate a random number.

**PRACTICAL 5.1**

**AIM:**

This week’s exercise is going to be revisiting an old exercise (see Practical 3), except require the solution in a different way.

Take two lists, say for example these two:

a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13] and write a program that returns a list that contains only the elements that are common between the lists (without duplicates). Make sure your program works on two lists of different sizes. Write this in one line of Python using at least one list comprehension

**CODE:**

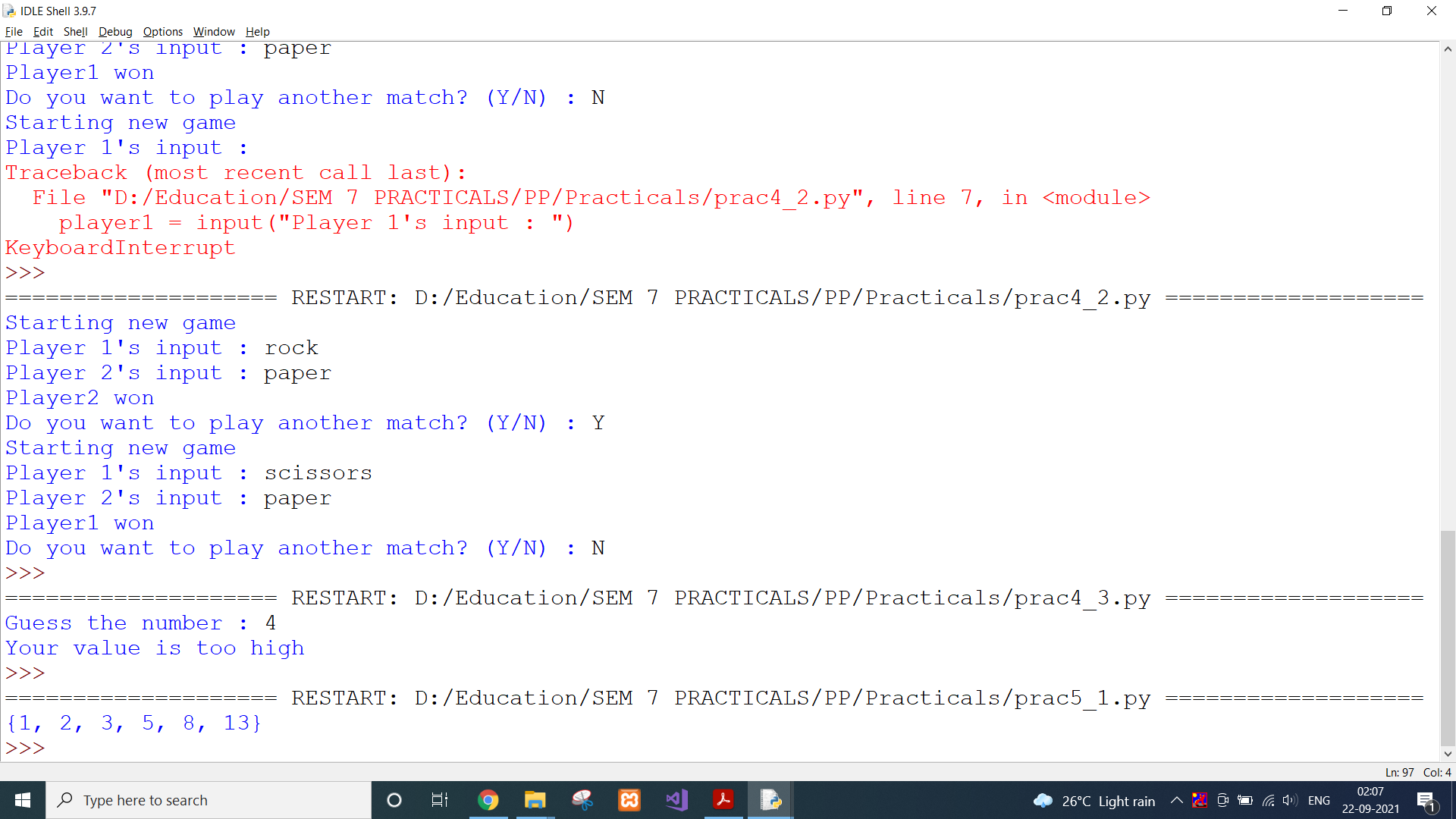
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

b = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]

ans = [i for i in a if i in b]

print(set(ans))

**OUTPUT:**



**CONCLUSION:**

In this practical, we revised the previous practical and created a list of intersection of two lists.

**PRACTICAL 5.2**

**AIM:**

Ask the user for a number and determine whether the number is prime or not. (For those who have forgotten, a prime number is a number that has no divisors.). You can (and should!) use your answer to Practical 2 to help you. Take this opportunity to practice using functions, described below.

**CODE:**

def prime(num):

for i in range(2,num):

if(num%i==0):

return 0

return 1

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

if(prime(num)):

print(num," is prime")

else:

print(num," is not prime")

**OUTPUT:**



**CONCLUSION:**

In this practical, we created a user-defined function to check if number is prime or not.

**PRACTICAL 5.3**

**AIM:**

Write a program that takes a list of numbers (for example, a = [5, 10, 15, 20, 25]) and makes a new list of only the first and last elements of the given list. For practice, write this code inside a function.

**CODE:**

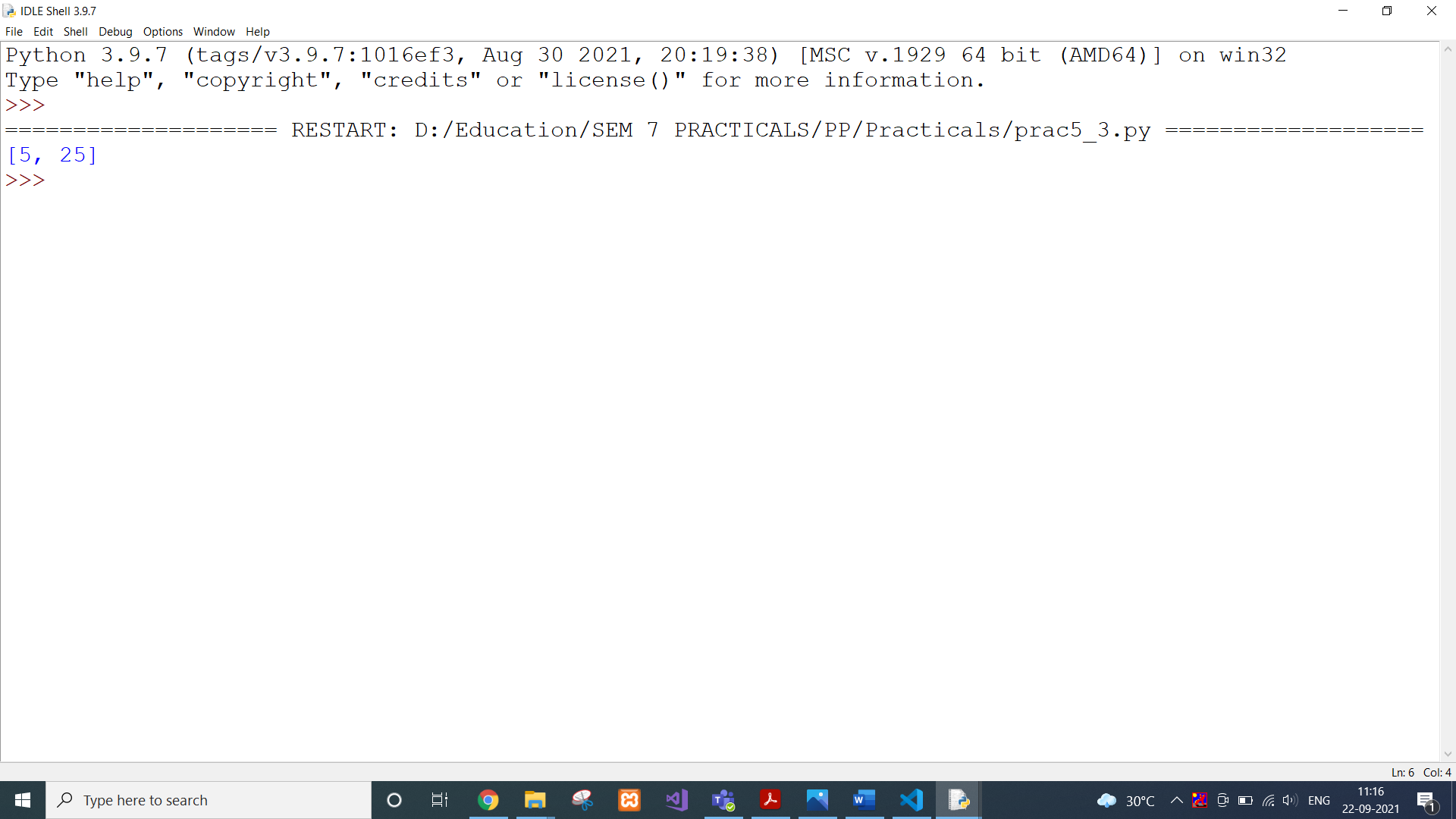
def firstLast(inp):

return [inp[0],inp[-1]]

a = [5, 10, 15, 20, 25]

print(firstLast(a))

**OUTPUT:**



**CONCLUSION:**

In this practical, we created a function and returned a list of first and last element of given list.

**PRACTICAL 6.1**

**AIM:**

Write a program that asks the user how many Fibonacci numbers to generate and then generates them. Take this opportunity to think about how you can use functions. Make sure to ask the user to enter the number of numbers in the sequence to generate. (Hint: The Fibonacci sequence is a sequence of numbers where the next number in the sequence is the sum of the previous two numbers in the sequence. The sequence looks like this: 1,

1, 2, 3, 5, 8, 13, …)

**CODE:**

def fibo(num):

a = 0

b = 1

for i in range(0,num):

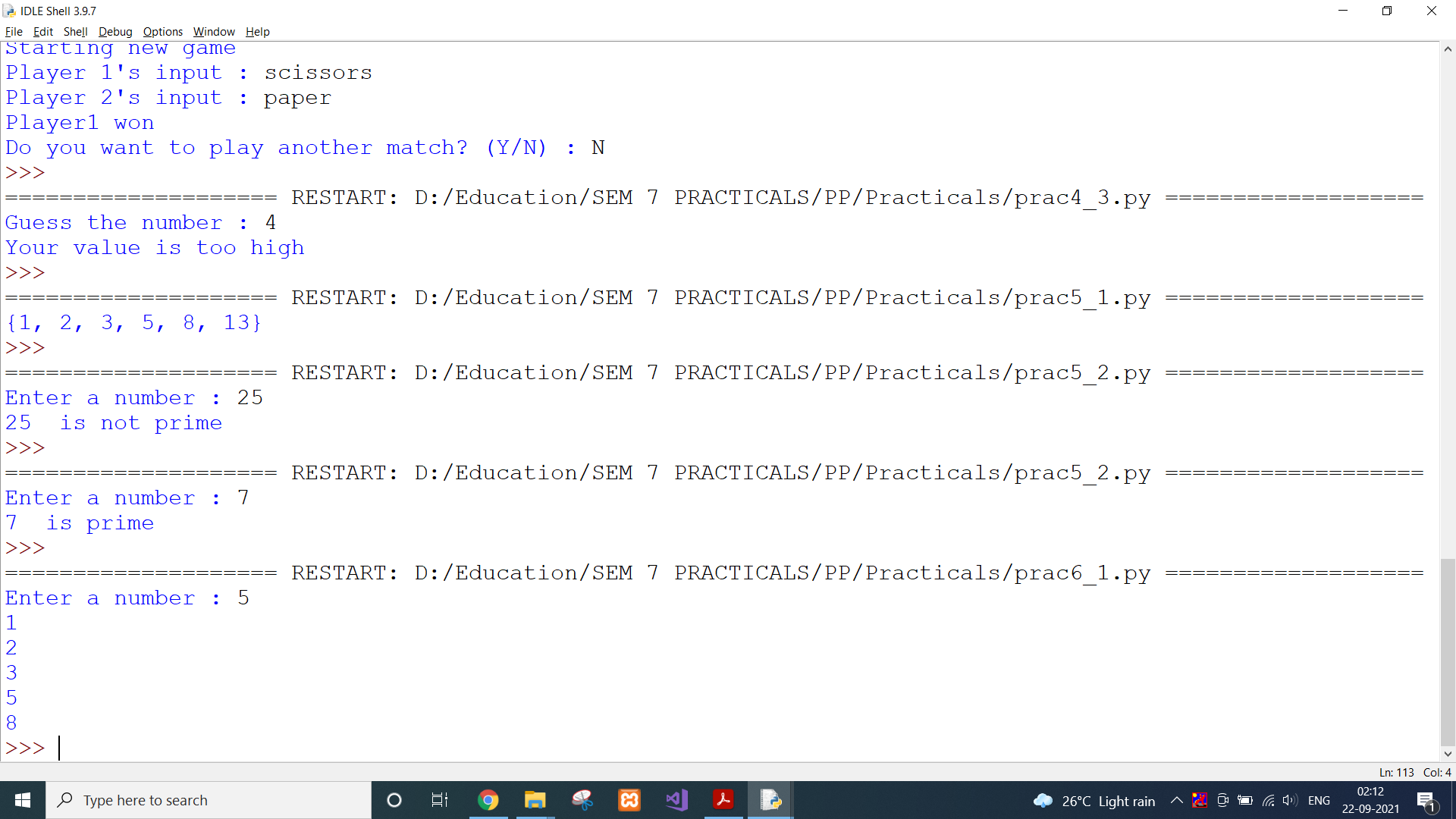
print(str(a+b))

a,b = b,a+b

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

fibo(num)

**OUTPUT:**



**CONCLUSION:**

In this practical, we sued the concept of function and listed required number of elements of Fibonacci serier.

**PRACTICAL 6.2**

**AIM:**

Write a program (function!) that takes a list and returns a new list that contains all the elements of the first list minus all the duplicates.

**CODE:**

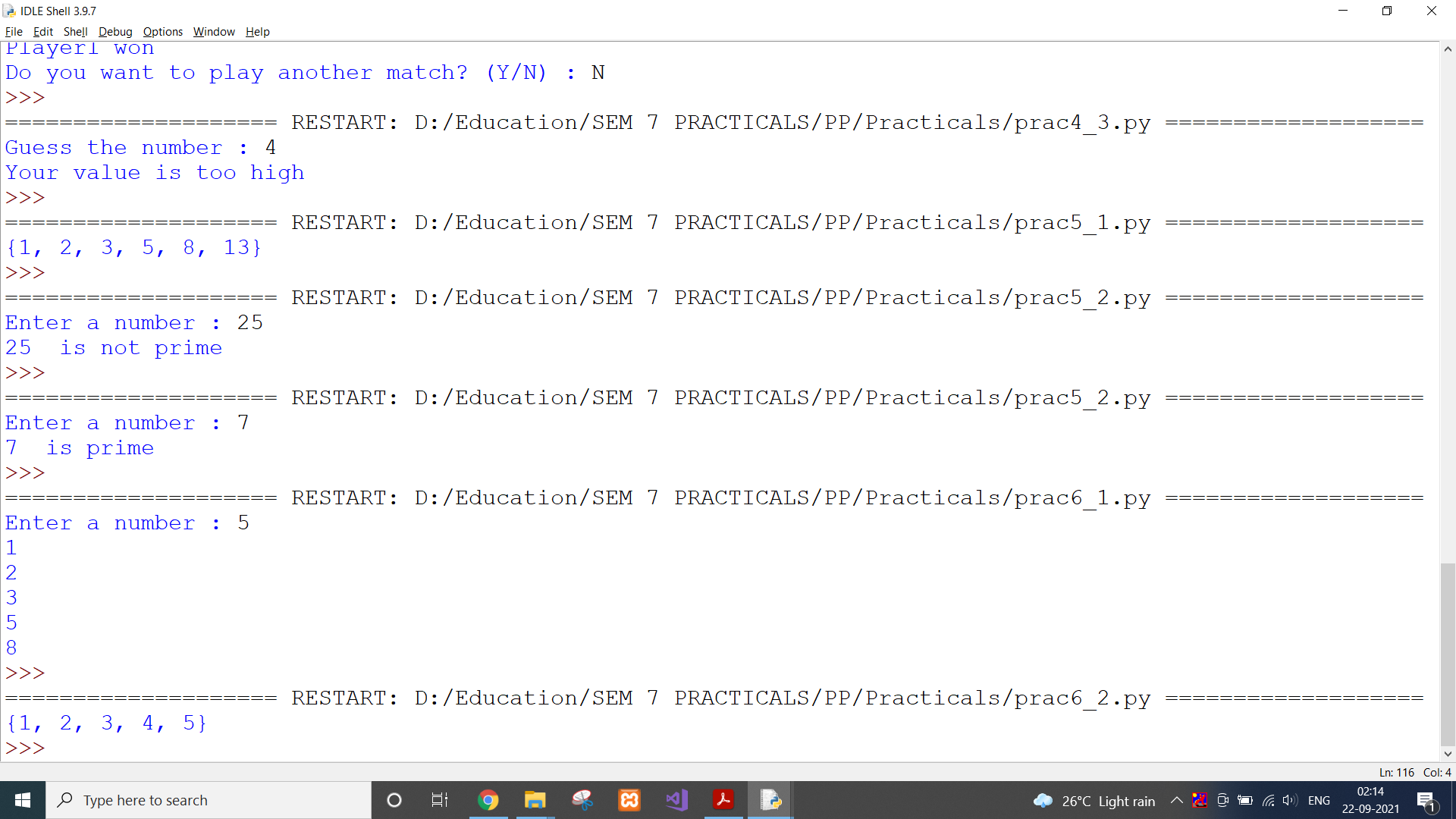
def unique(inp):

return set(inp)

inp = [1,2,3,4,2,5,2,1,5,3,2]

print(unique(inp))

**OUTPUT:**



**CONCLUSION:**

In this practical, we created a list of unique elements from given list.

**PRACTICAL 6.3**

**AIM:**

Write a program (using functions!) that asks the user for a long string function. Containing multiple words. Print back to the user the same string, except with the words in backwards order. For example, say I type the string: My name is Michele

Then I would see the string: Michele is name My shown back to me.

**CODE:**

def reverse(string):

temp\_list = string.split(" ")

temp\_list\_reverse = temp\_list[::-1]

new\_string = ""

for i in temp\_list\_reverse:

new\_string += str(i) + " "

return new\_string

inp = input("Enter a long string : ")

print(reverse(inp))

**OUTPUT:**



**CONCLUSION:**

In this practical, we used a function to reverse the string as required.

**PRACTICAL 7.1**

**AIM:**

Write a password generator in Python. Be creative with how you generate passwords - strong passwords have a mix of lowercase letters, uppercase letters, numbers, and symbols. The passwords should be random, generating a new password every time the user asks for a new password.

Include your run-time code in a main method.

**CODE:**

from random import randint

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

length = randint(8,12)

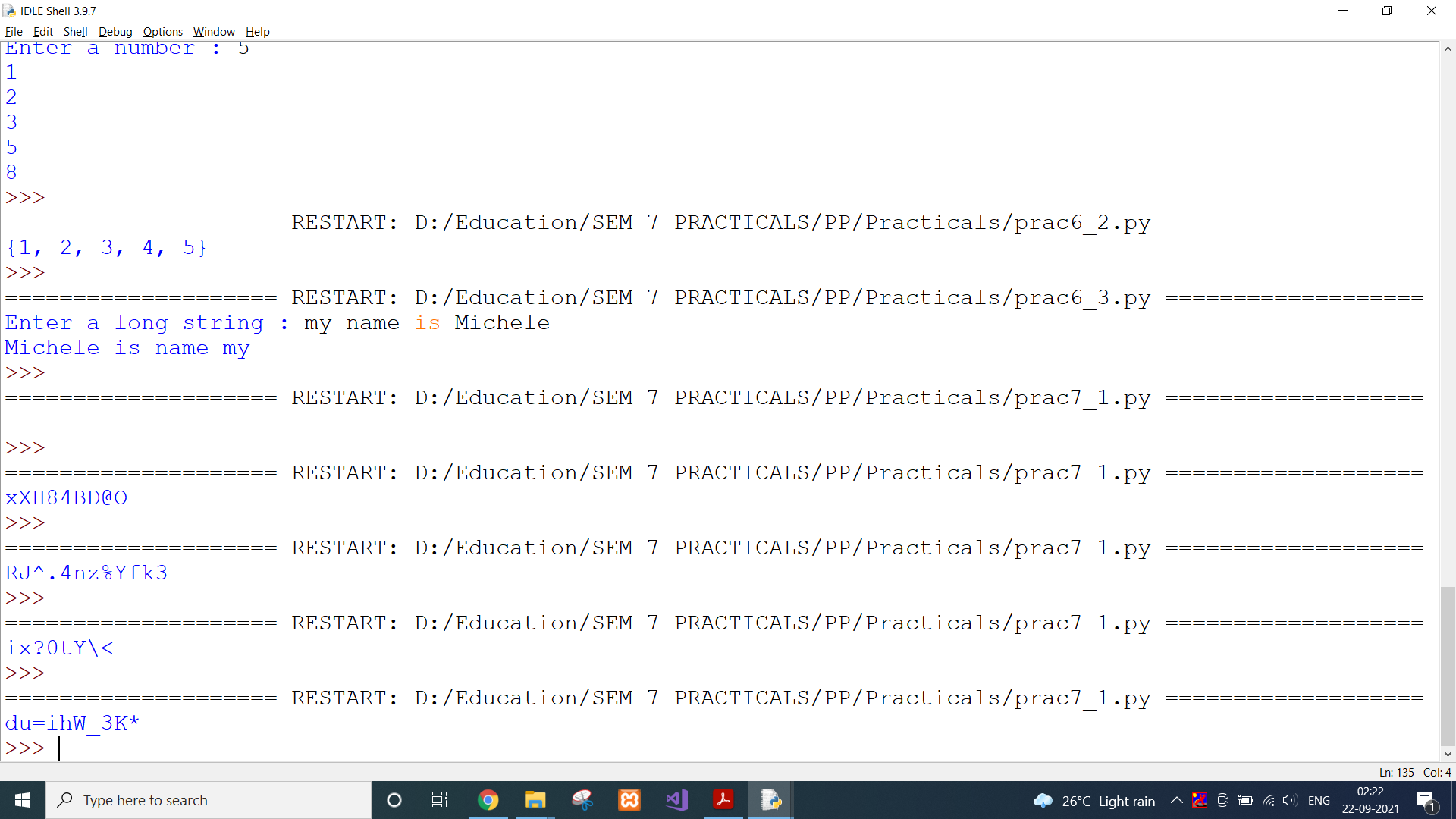
password = ""

for i in range(0,length):

password += chr(randint(33,122))

print(password)

**OUTPUT:**



**CONCLUSION:**

In this practical, we used random library to generate a strong password.

**PRACTICAL 7.2**

**AIM:**

Write a Python class named Circle constructed by a radius and two methods which will compute the area and the perimeter of a circle.

**CODE:**

class Circle:

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

self.radius = radius

def area(self):

return 3.14\*self.radius\*self.radius

def perimeter(self):

return 2\*3.14\*self.radius

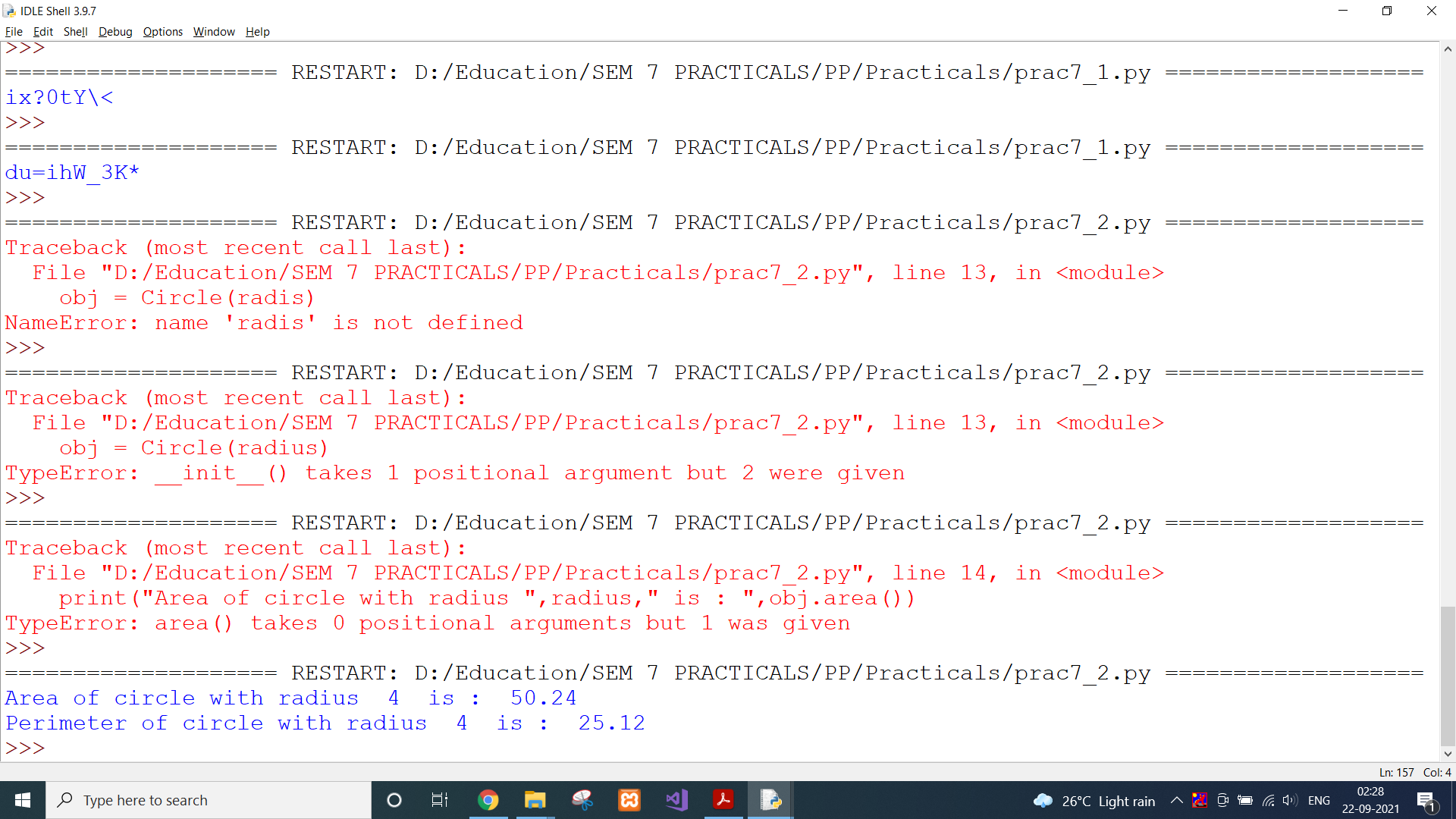
radius = 4

obj = Circle(radius)

print("Area of circle with radius ",radius," is : ",obj.area())

print("Perimeter of circle with radius ",radius," is : ",obj.perimeter())

**OUTPUT:**



**CONCLUSION:**

In this practical, we created a class with two methods to calculate area and perimeter of a circle.

**PRACTICAL 8.1**

**AIM:**

Python supports classes inheriting from other classes. The class being inherited is called the Parent or Superclass, while the class that inherits is called the Child or Subclass. How can we define the order in which the base classes are searched when executing a method?

**CODE:**

class A:

def \_init\_(self):

print("Constructor A")

def me(self):

print("In class A")

class B:

def \_init\_(self):

print("Constructor B")

def me(self):

print("In class B")

# classes ordering

class C(A, B):

def \_init\_(self):

print("Constructor C")

super().\_init\_()

r = C()

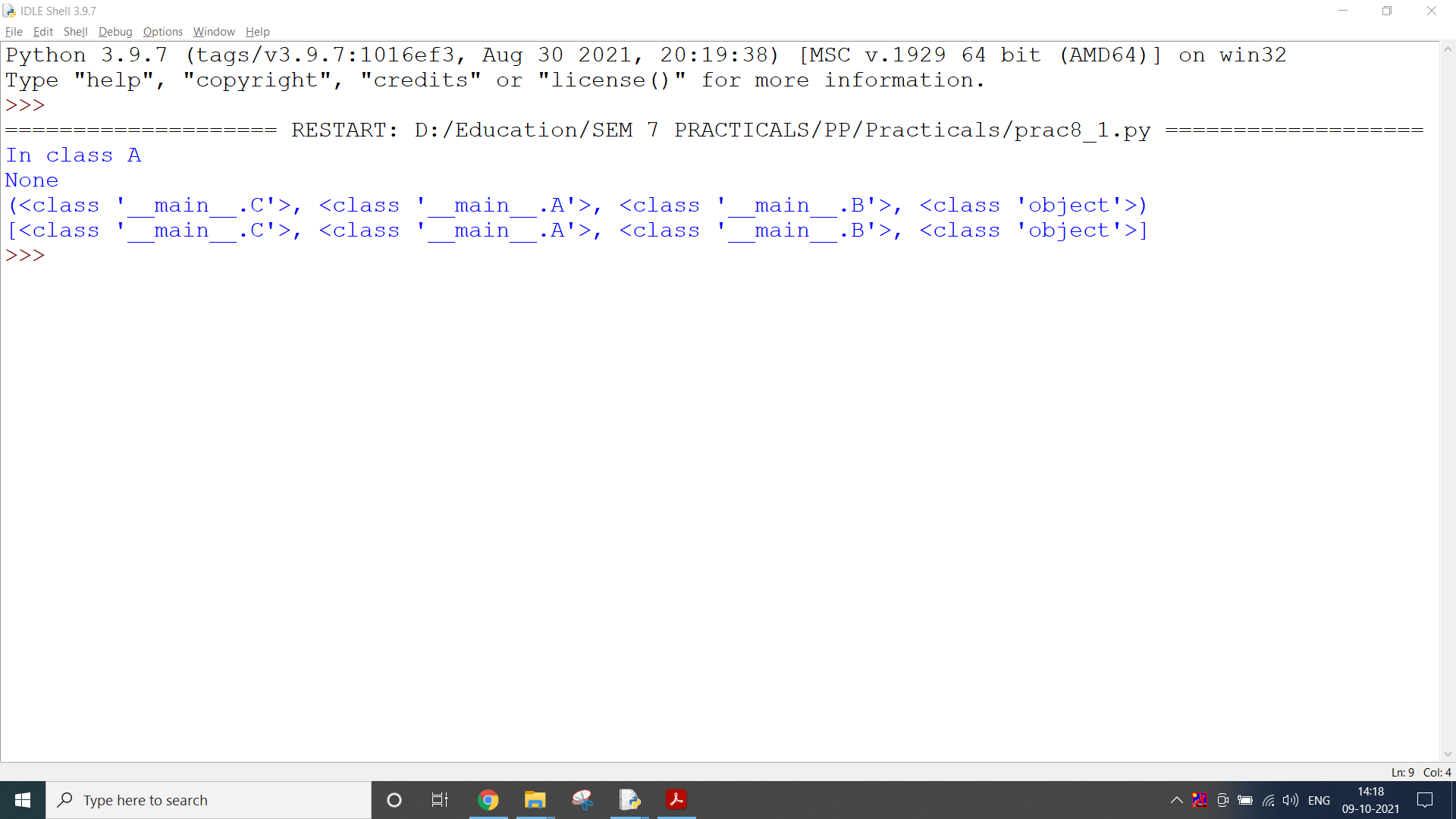
print(r.me())

# it prints the lookup order

print(C.\_\_mro\_\_)

print(C.mro())

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt about inheritance and relation between parent and child classes.

**PRACTICAL 8.2**

**AIM:**

Write a function that takes an ordered list of numbers (a list where the elements are in order from smallest to largest) and another number. The function decides whether or not the given number is inside the list and returns (then prints) an appropriate Boolean.

**CODE:**

def order(ordered\_list, num):

for element in ordered\_list:

if element == num:

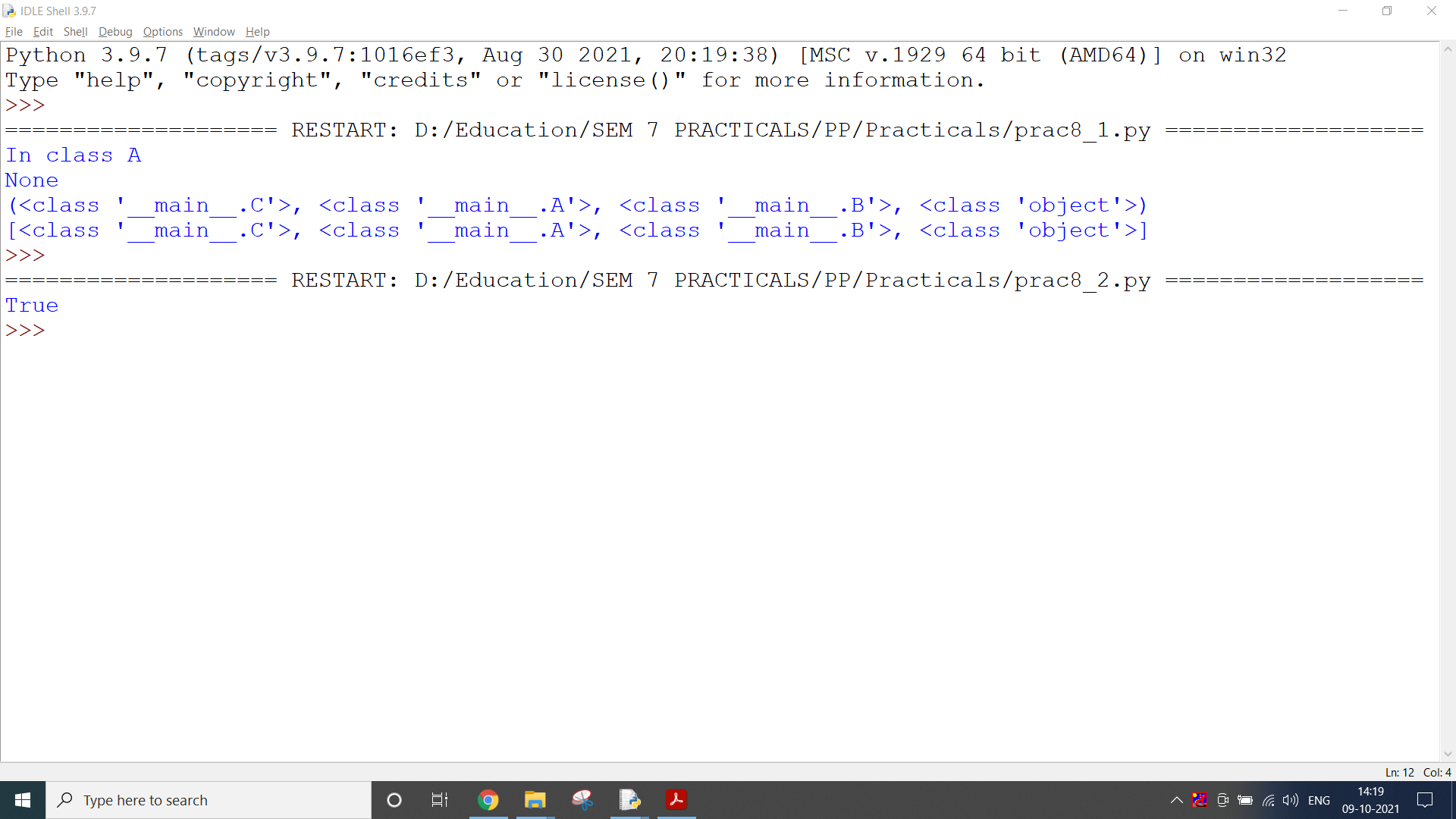
return True

return False

ordered\_list = [1, 2, 3, 4, 5, 6, 17]

print(order(ordered\_list, 4))

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt about sublist and defined a function to check if one list is part of other list or not.

**PRACTICAL 8.3**

**AIM:**

Given a .txt file that has a list of a bunch of names, count how many of each name there are in the file, and print out the results to the screen.

**CODE:**

counter\_dict = {}

with open("./names.txt") as var:

line = var.readline()

while line:

line = line.strip()

if line in counter\_dict:

counter\_dict[line] += 1

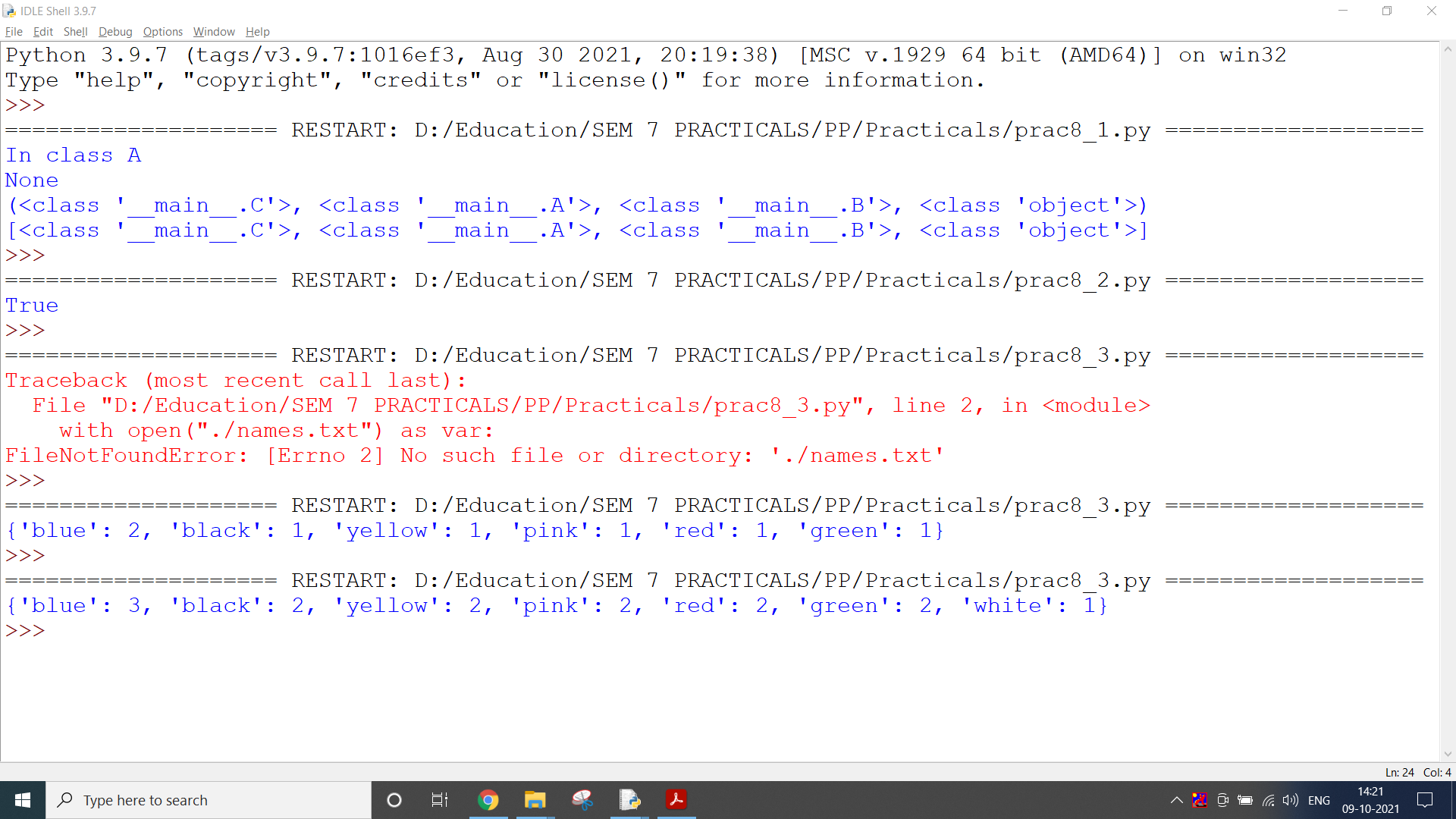
else:

counter\_dict[line] = 1

line = var.readline()

print(counter\_dict)

**OUTPUT:**



**CONCLUSION:**

In this practical, we created a program to count the words from input text file.

**PRACTICAL 9.1**

**AIM:**

Develop programs to learn regular expressions using python.

**CODE:**

# re.findall() - returns a list of strings containing all matches

# Extracting numbers from a string

import re

string = "Hello, I am DEPSTAR student, 18dcs, CSE"

pattern = "\d+"

result = re.findall(pattern, string)

print(result)

# re.split() - splits the string where there is a match and returns a list of strings where the splits have occurred.

# If the pattern is not found, re.split() returns a list containing the original string.

result1 = re.split(pattern, string)

print(result1)

# re.sub() - returns a string where matched occurrences are replaced with the content of replace variable.

# If the pattern is not found, re.sub() returns the original string.

pattern1 = "\s+"

replace = ""

new\_string = re.sub(pattern1, replace, string)

print(new\_string)

# re.search() - takes two arguments: a pattern and a string. The method looks for the first location where the RegEx

# pattern produces a match with the string.

match = re.search("\AHello", string)

if match:

print("pattern found inside the string")

else:

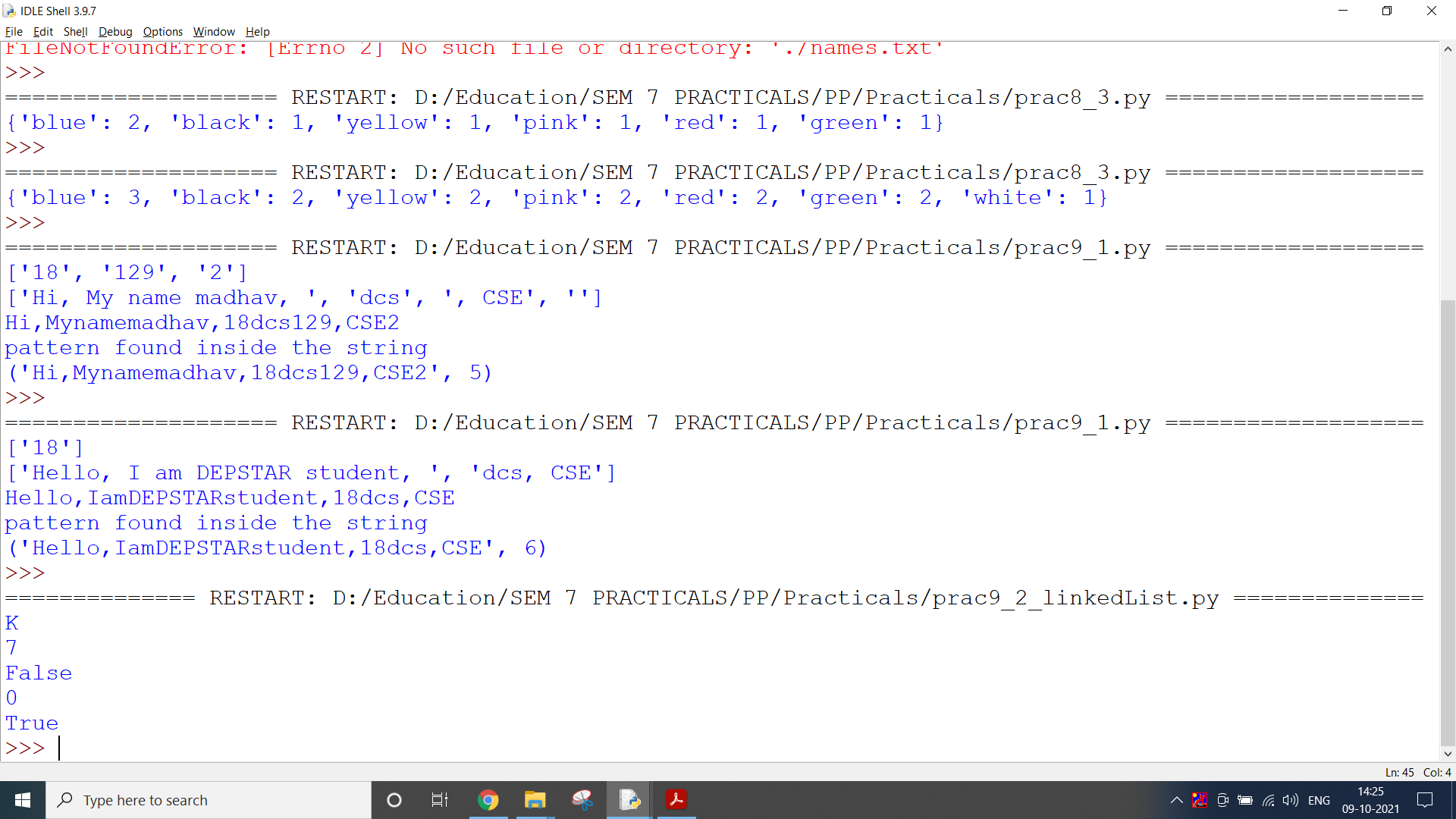
print("pattern not found")

# re.subn() - returns a tuple of 2 items containing the new string and the number of substitutions made.

new\_string1 = re.subn(pattern1, replace, string)

print(new\_string1)

**OUTPUT:**



**CONCLUSION:**

In this practical, we used Regular expression to find certain patterns from given string.

**PRACTICAL 9.2**

**AIM:**

Implementation of different Data structures using Python.

* Linked List
* Stack
* Queue
* Binary Tree
* Binary Search Tree
* Heap
* Hashing
* Graph

**CODE:**

*Linked List*

# Linked List

class LinkedListStack:

class Node:

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

self.element = e

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

self.\_size = 0

self.head = None

def push(self, e):

newest = self.Node(e)

newest.next = self.head

self.head = newest

self.\_size += 1

def pop(self):

if self.is\_empty():

raise IndexError('Stack is empty')

elementToReturn = self.head.element

self.head = self.head.next

self.\_size -= 1

return elementToReturn

def peek(self):

if self.is\_empty():

raise IndexError('Stack is empty')

return self.head.element

def is\_empty(self):

return self.\_size == 0

def size(self):

return self.\_size

LLS = LinkedListStack()

LLS.push("L")

LLS.push("L")

LLS.push("S")

LLS.push("T")

LLS.push("A")

LLS.push("C")

LLS.push("K")

print(LLS.peek())

print(LLS.size())

print(LLS.is\_empty() )

LLS.pop()

LLS.pop()

LLS.pop()

LLS.pop()

LLS.pop()

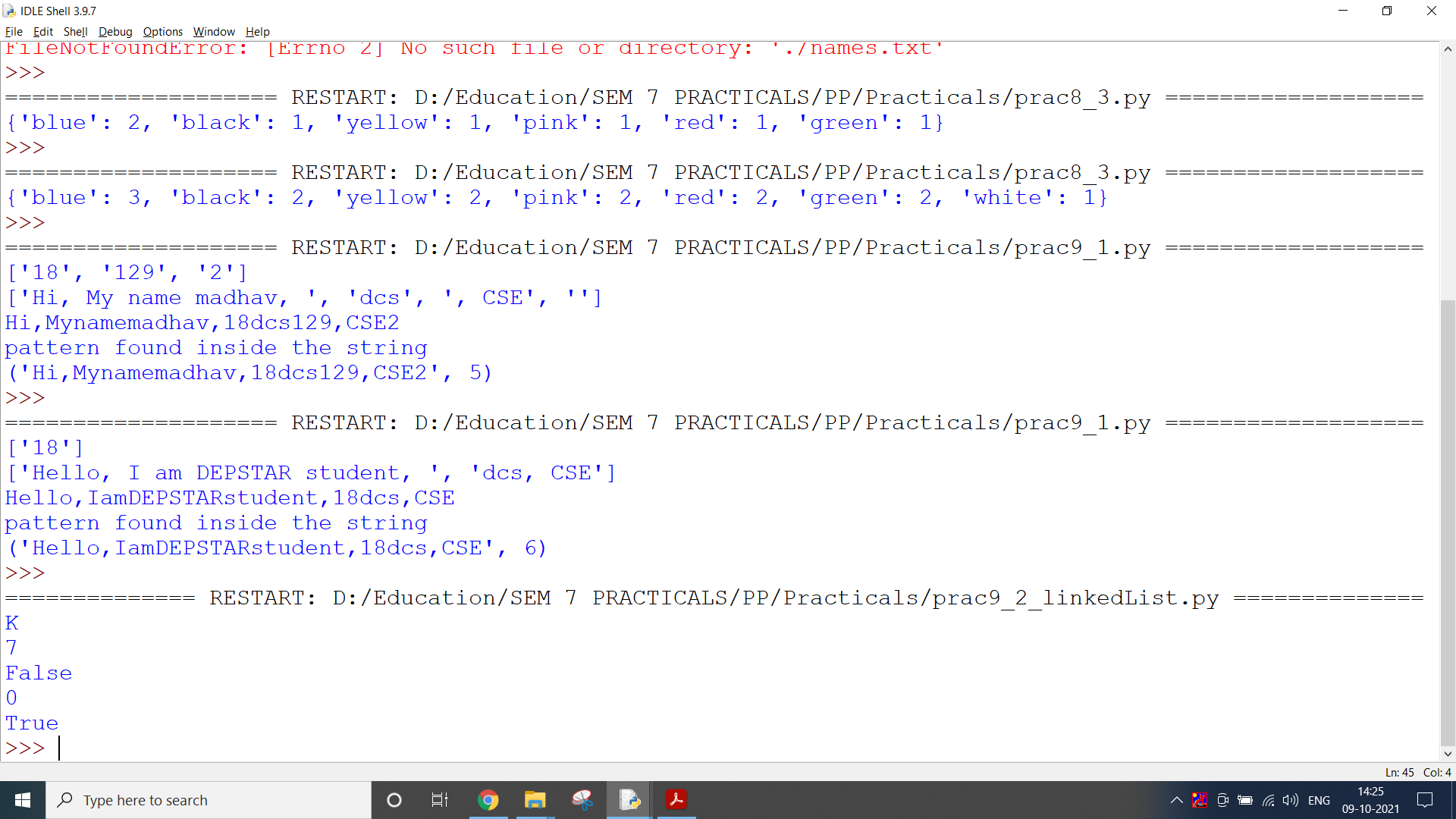
LLS.pop()

LLS.pop()

print(LLS.size())

print(LLS.is\_empty() )

**OUTPUT:**



**CODE:**

*Stack*

# Stack

class StackADT:

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

self.data = []

def push(self, e):

self.data.append(e)

def pop(self):

if self.is\_empty():

raise IndexError('Stack is empty')

else:

return self.data.pop()

def peek(self):

if self.is\_empty():

raise IndexError('Stack is empty')

else:

return self.data[-1]

def is\_empty(self):

return len(self.data) == 0

def size(self):

return len(self.data)

S = StackADT()

S.push("S")

S.push("T")

S.push("A")

S.push("C")

S.push("K")

print(S.peek())

print(S.size())

print(S.is\_empty())

S.pop()

S.pop()

S.pop()

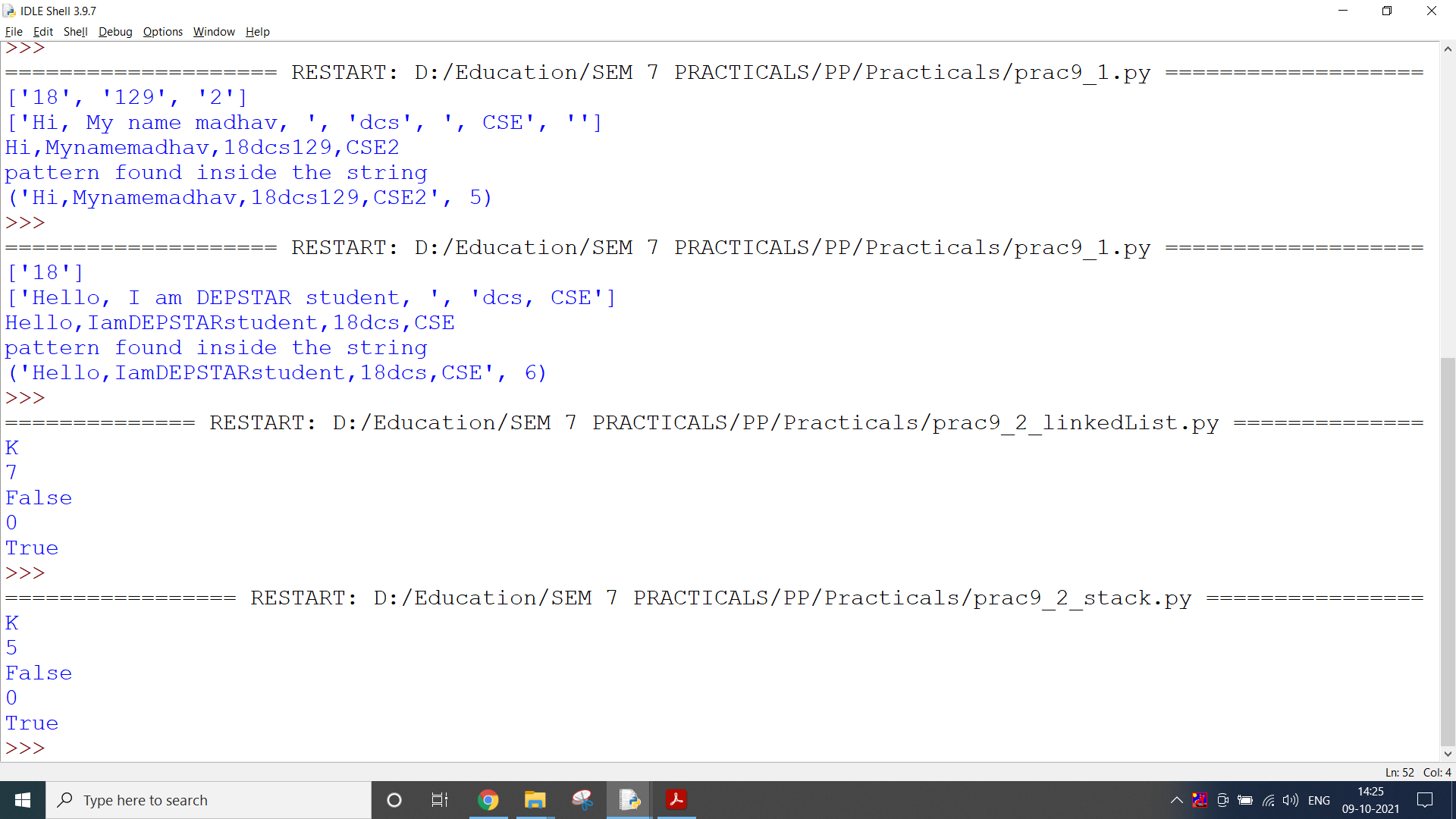
S.pop()

S.pop()

print(S.size())

print(S.is\_empty())

**OUTPUT:**



**CODE:**

*Queue*

# Queue

class QueueADT:

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

self.data = []

def enqueue(self, e):

self.data.insert(0, e)

def dequeue(self):

if self.is\_empty():

raise IndexError('Queue is empty')

else:

return self.data.pop()

def peek(self):

if self.is\_empty():

raise IndexError('Queue is empty')

else:

return self.data[-1]

def is\_empty(self):

return len(self.data) == 0

def size(self):

return len(self.data)

Q = QueueADT()

Q.enqueue("Q")

Q.enqueue("U")

Q.enqueue("E")

Q.enqueue("U")

Q.enqueue("E")

print(Q.peek())

print(Q.size())

print(Q.is\_empty())

Q.dequeue()

Q.dequeue()

Q.dequeue()

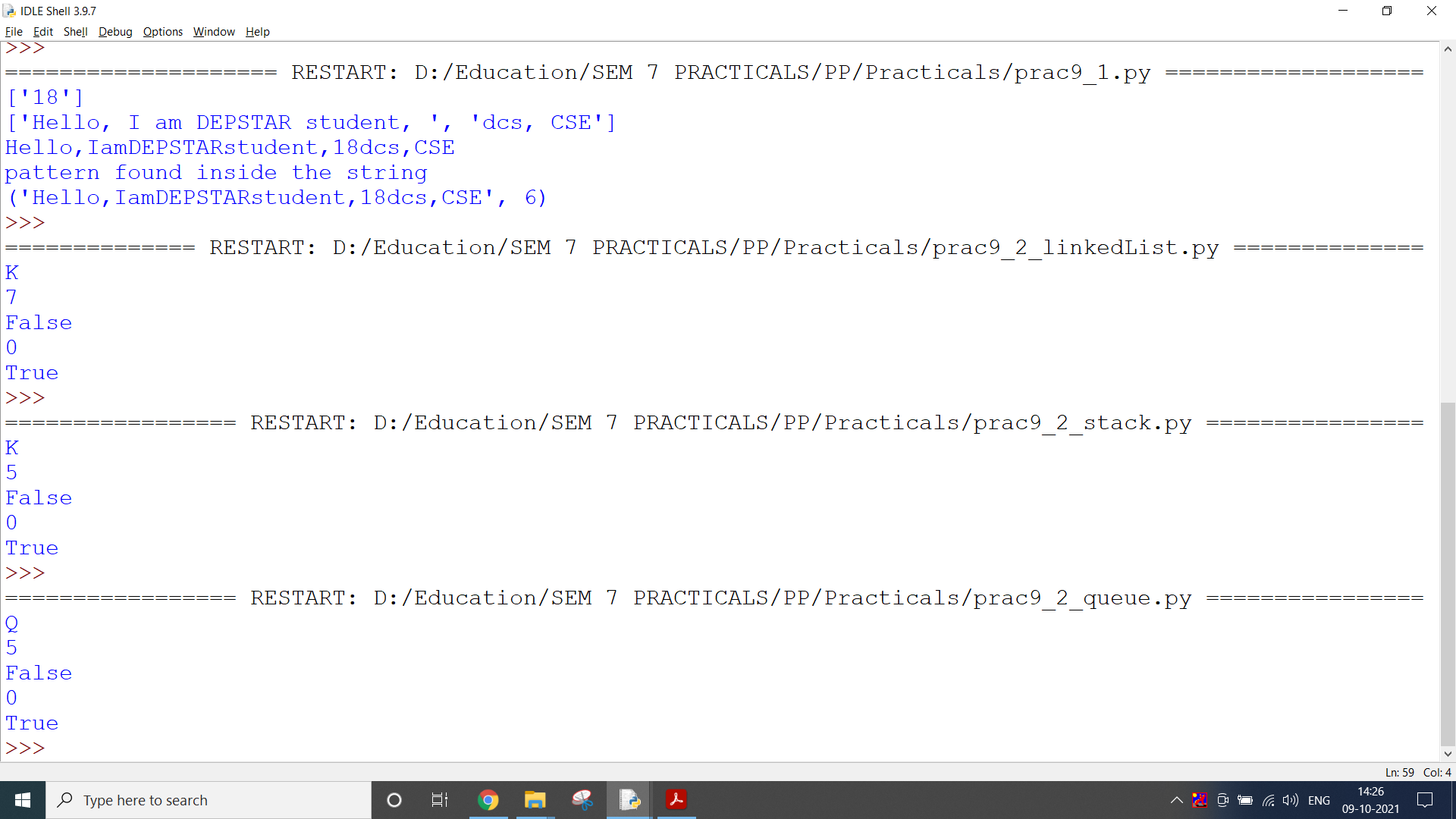
Q.dequeue()

Q.dequeue()

print(Q.size())

print(Q.is\_empty())

**OUTPUT:**



**CODE:**

*Binary Tree*

# Binary Tree

class Node:

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

self.left = None

self.right = None

self.data = data

root = Node(10)

root.left = Node(34) # Setting the left child of the root to 34

root.right = Node(89) # Setting the right child of the root to 89

def inorder(node):

if node:

inorder(node.left)

print(node.data)

inorder(node.right)

def preorder(node):

if node:

print(node.data)

preorder(node.left)

preorder(node.right)

def postorder(node):

if node:

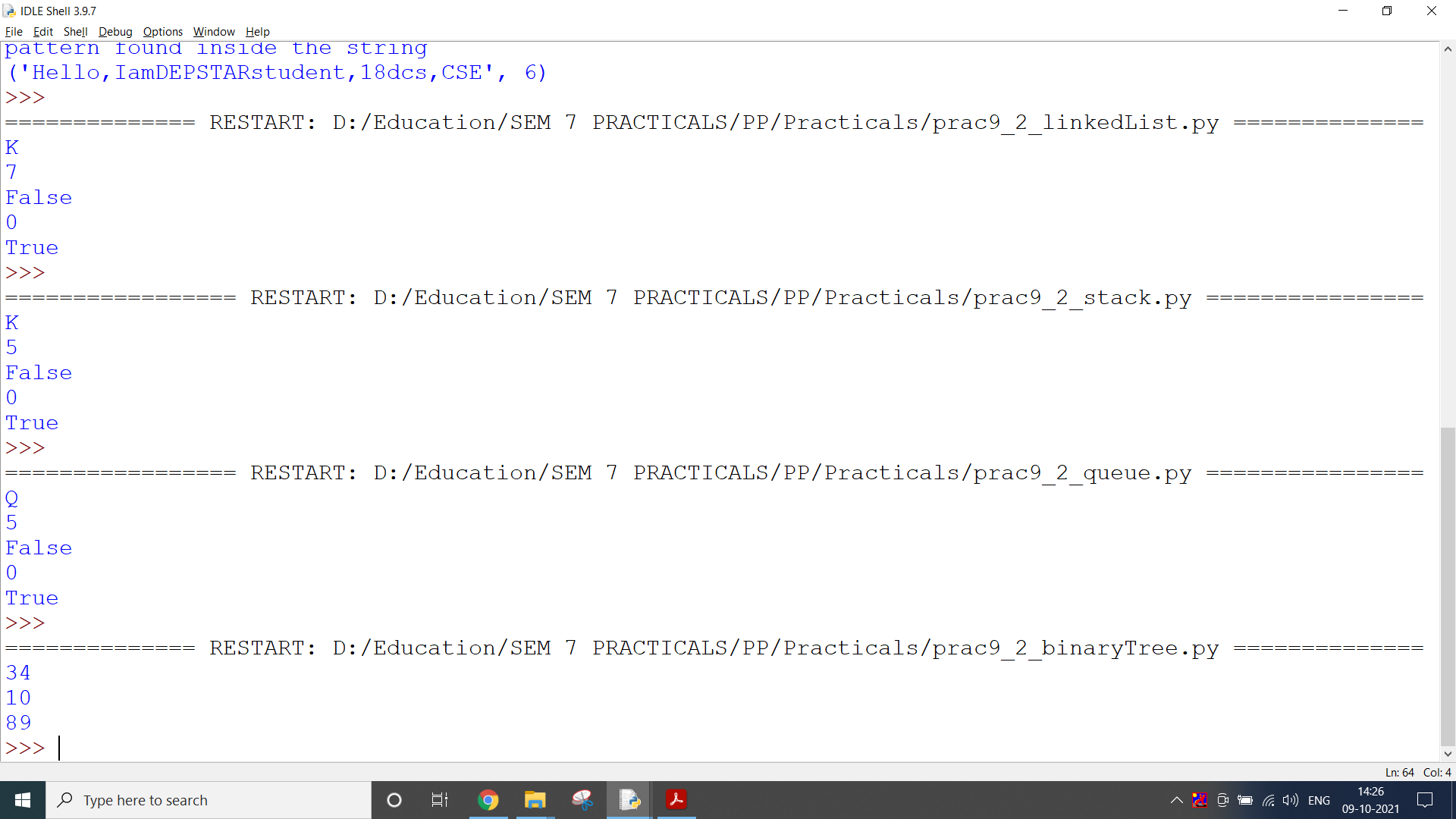
postorder(node.left)

postorder(node.right)

print(node.data)

inorder(root)

**OUTPUT:**



**CODE:**

*Binary Search Tree*

# Binary Search Tree

class BinarySearchTree:

class Node:

\_\_slots\_\_ = 'element', 'left', 'right'

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

self.element = e

self.left = None # reference to the left Child

self.right = None # reference to the right Child

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

self.root = None

self.\_size = 0

def add\_node(self, root, e):

if self.root == None:

self.root = self.Node(e)

self.\_size += 1

return

if e < root.element:

if root.left == None:

root.left = self.Node(e)

self.\_size += 1

else:

self.add\_node(root.left, e)

else:

if root.right == None:

root.right = self.Node(e)

self.\_size += 1

else:

self.add\_node(root.right, e)

def traverse\_in\_order(self, root):

if root != None:

self.traverse\_in\_order(root.left)

print(root.element, end=" ")

self.traverse\_in\_order(root.right)

def height(self, root):

if root == None:

return 0

return max(self.height(root.left),

self.height(root.right)) + 1

def is\_empty(self):

return self.\_size == 0

def size(self):

return self.\_size

t = BinarySearchTree()

print(t.is\_empty())

t.add\_node(t.root, 56)

t.add\_node(t.root, 13)

t.add\_node(t.root, 1)

t.add\_node(t.root, 6)

t.add\_node(t.root, 3)

t.add\_node(t.root, 67)

t.add\_node(t.root, 3)

t.add\_node(t.root, 45)

t.add\_node(t.root, 99)

t.add\_node(t.root, 2)

print(t.is\_empty())

print(t.size())

print(t.height(t.root))

print(t.traverse\_in\_order(t.root))

**OUTPUT:**



**CODE:**

*Heap*

# Heap

class MaxHeap:

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

# Initialize a heap using list

self.heap = []

def getParentPosition(self, i):

# The parent is located at floor((i-1)/2)

return int((i-1)/2)

def getLeftChildPosition(self, i):

# The left child is located at 2 \* i + 1

return 2\*i+1

def getRightChildPosition(self, i):

# The right child is located at 2 \* i + 2

return 2\*i+2

def hasParent(self, i):

# This function checks if the given node has a parent or not

return self.getParentPosition(i) < len(self.heap)

def hasLeftChild(self, i):

# This function checks if the given node has a left child or not

return self.getLeftChildPosition(i) < len(self.heap)

def hasRightChild(self, i):

# This function checks if the given node has a right child or not

return self.getRightChildPosition(i) < len(self.heap)

def insert(self, key):

self.heap.append(key) # Adds the key to the end of the list

# Re-arranges the heap to maintain the heap property

self.heapify(len(self.heap) - 1)

def getMax(self):

# Returns the largest value in the heap in O(1) time.

return self.heap[0]

def heapify(self, i):

# Loops until it reaches a leaf node

while(self.hasParent(i) and self.heap[i] > self.heap[self.getParentPosition(i)]):

self.heap[i], self.heap[self.getParentPosition(

i)] = self.heap[self.getParentPosition(i)], self.heap[i] # Swap the values

i = self.getParentPosition(i) # Resets the new position

def printHeap(self):

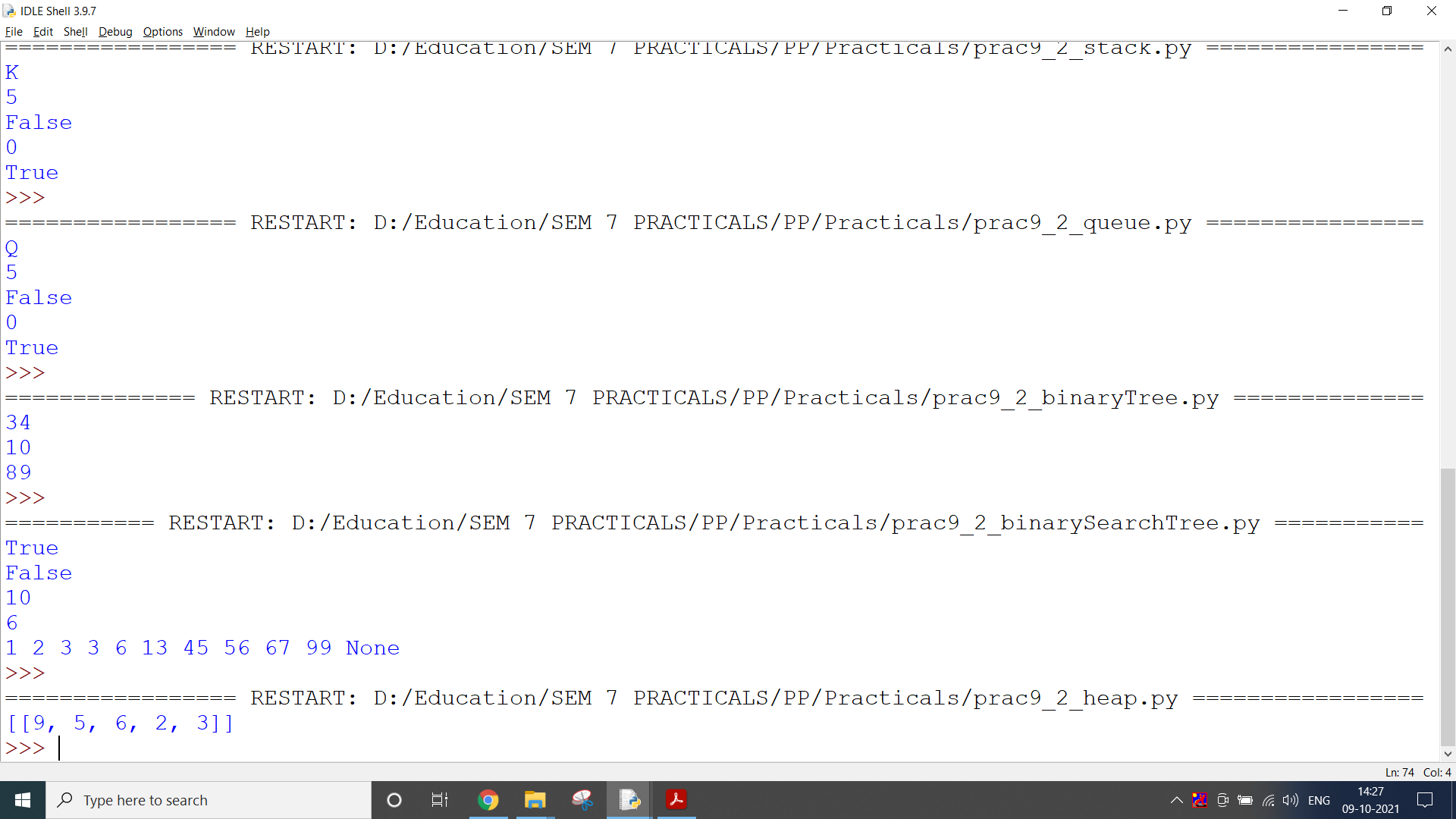
print(self.heap) # Prints the heap

bh = MaxHeap()

bh.insert([9, 5, 6, 2, 3])

bh.printHeap()

**OUTPUT:**



**CODE:**

*Hashing*

# Hashing

import pprint

class Hashtable:

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

self.bucket\_size = len(elements)

self.buckets = [[] for \_ in range(self.bucket\_size)]

self.\_assign\_buckets(elements)

def \_assign\_buckets(self, elements):

for key, value in elements: # calculates the hash of each key

hashed\_value = hash(key)

# positions the element in the bucket using hash

index = hashed\_value % self.bucket\_size

# adds a tuple in the bucket

self.buckets[index].append((key, value))

def get\_value(self, input\_key):

hashed\_value = hash(input\_key)

index = hashed\_value % self.bucket\_size

bucket = self.buckets[index]

for key, value in bucket:

if key == input\_key:

return(value)

return None

def \_\_str\_\_(self):

# pformat returns a printable representation of the object

return pprint.pformat(self.buckets)

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

capitals = [

('France', 'Paris'),

('United States', 'Washington D.C.'),

('Italy', 'Rome'),

('Canada', 'Ottawa')

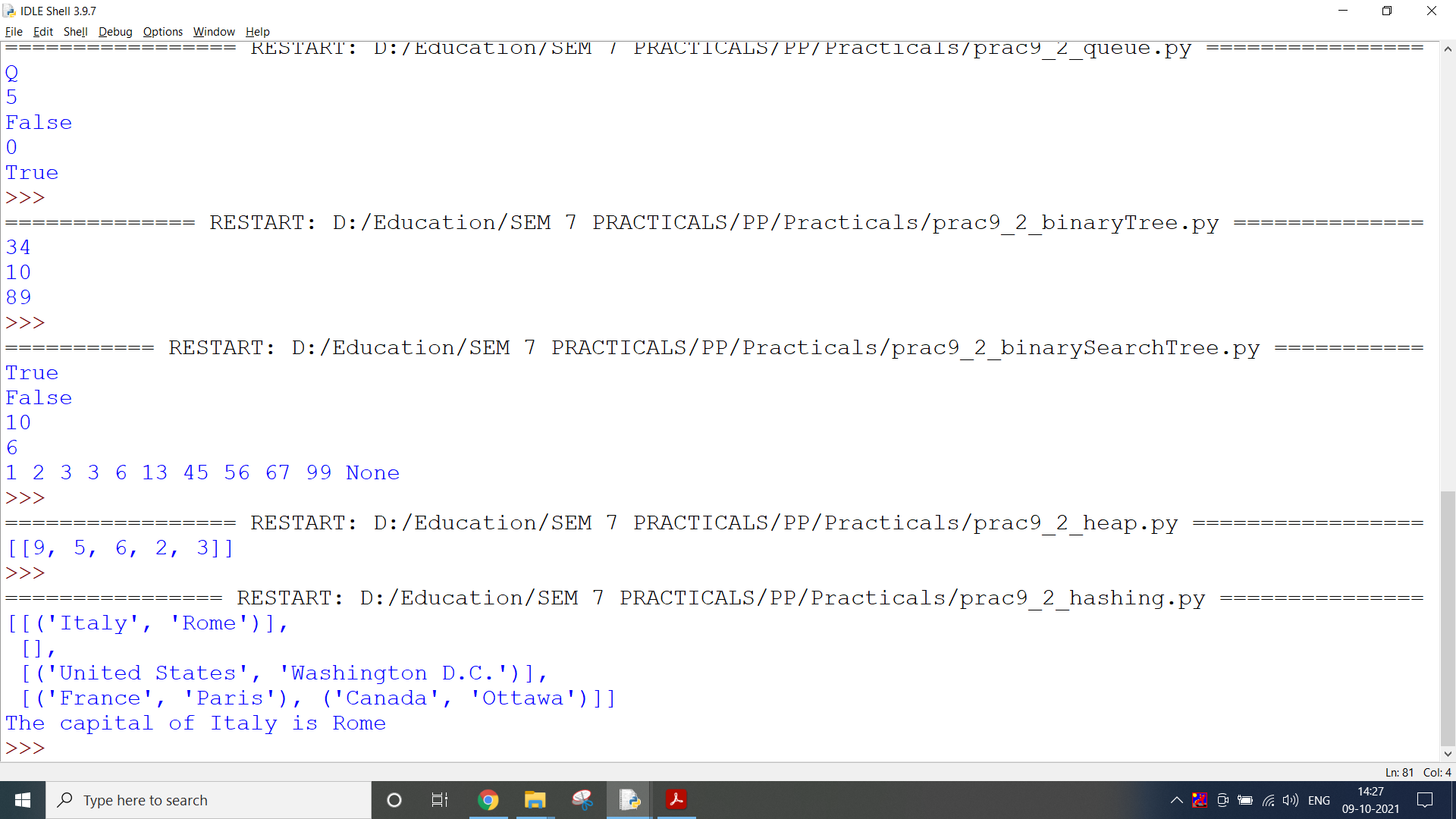
]

hashtable = Hashtable(capitals)

print(hashtable)

print(f"The capital of Italy is {hashtable.get\_value('Italy')}")

**OUTPUT:**



**CODE:**

*Graph*

# Graph

class AdjacencyList(object):

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

self.List = {}

def addEdge(self, fromVertex, toVertex):

# check if vertex is already present

if fromVertex in self.List.keys():

self.List[fromVertex].append(toVertex)

else:

self.List[fromVertex] = [toVertex]

def printList(self):

for i in self.List:

print(i, '->', ' -> '.join([str(j) for j in self.List[i]]))

al = AdjacencyList()

al.addEdge(0, 1)

al.addEdge(0, 4)

al.addEdge(4, 1)

al.addEdge(4, 3)

al.addEdge(1, 0)

al.addEdge(1, 4)

al.addEdge(1, 3)

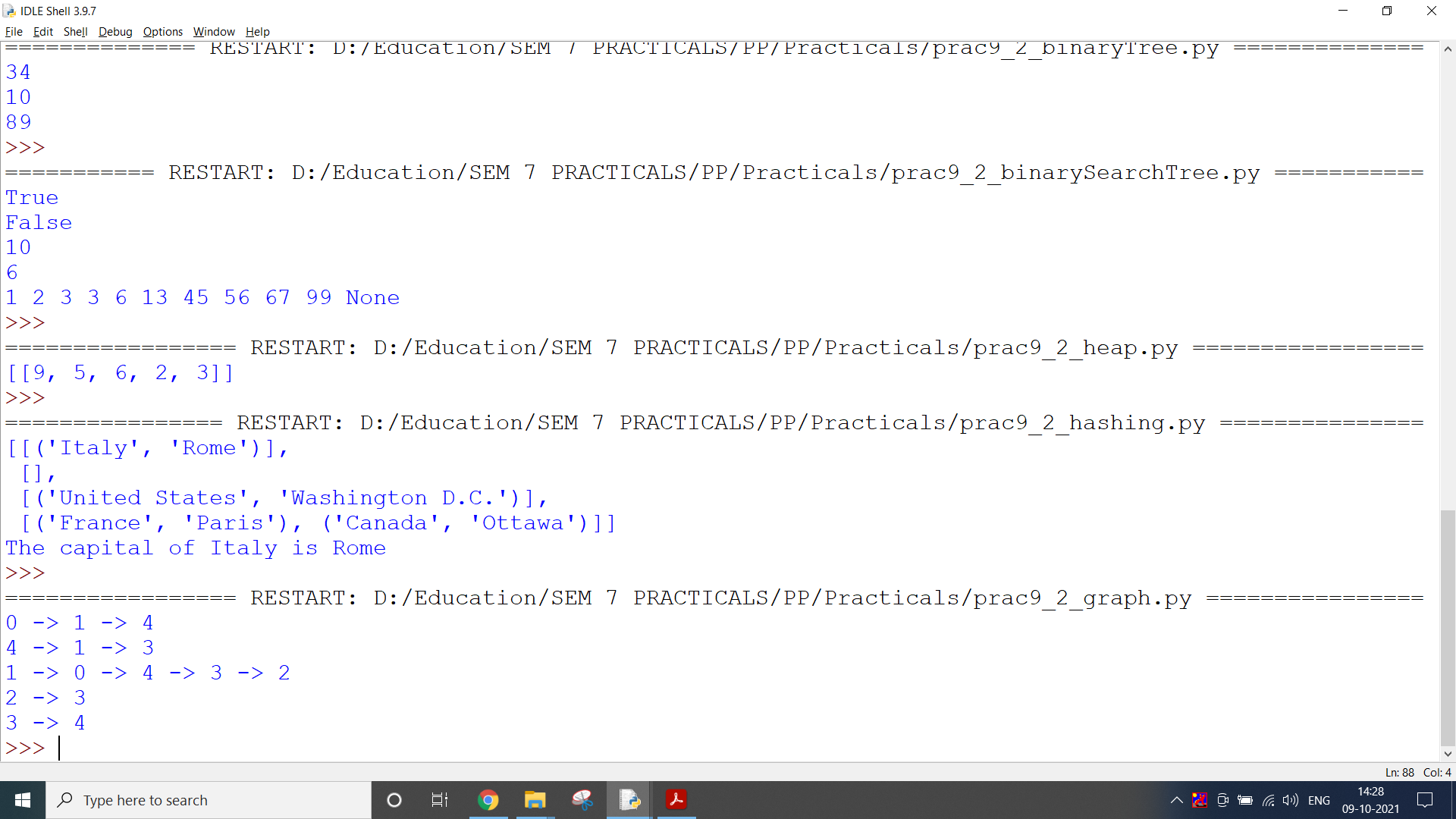
al.addEdge(1, 2)

al.addEdge(2, 3)

al.addEdge(3, 4)

al.printList()

**OUTPUT:**



**CONCLUSION:**

In this practical, we implemented different data structures and analysed their different operations.

**PRACTICAL 9.3**

**AIM:**

Develop programs for data structure algorithms using python – sorting (Bubble sort and Insertion sort)

**CODE:**

def bubble\_sort(nums):

swapped = True

while swapped:

swapped = False

for i in range(len(nums) - 1):

if nums[i] > nums[i + 1]:

nums[i], nums[i + 1] = nums[i + 1], nums[i]

swapped = True

num = [5, 2, 1, 8, 4]

bubble\_sort(num)

print("bubble sort:")

print(num)

def insertion\_sort(nums):

for i in range(1, len(nums)):

item\_to\_insert = nums[i]

j = i - 1

while j >= 0 and nums[j] > item\_to\_insert:

nums[j + 1] = nums[j]

j -= 1

nums[j + 1] = item\_to\_insert

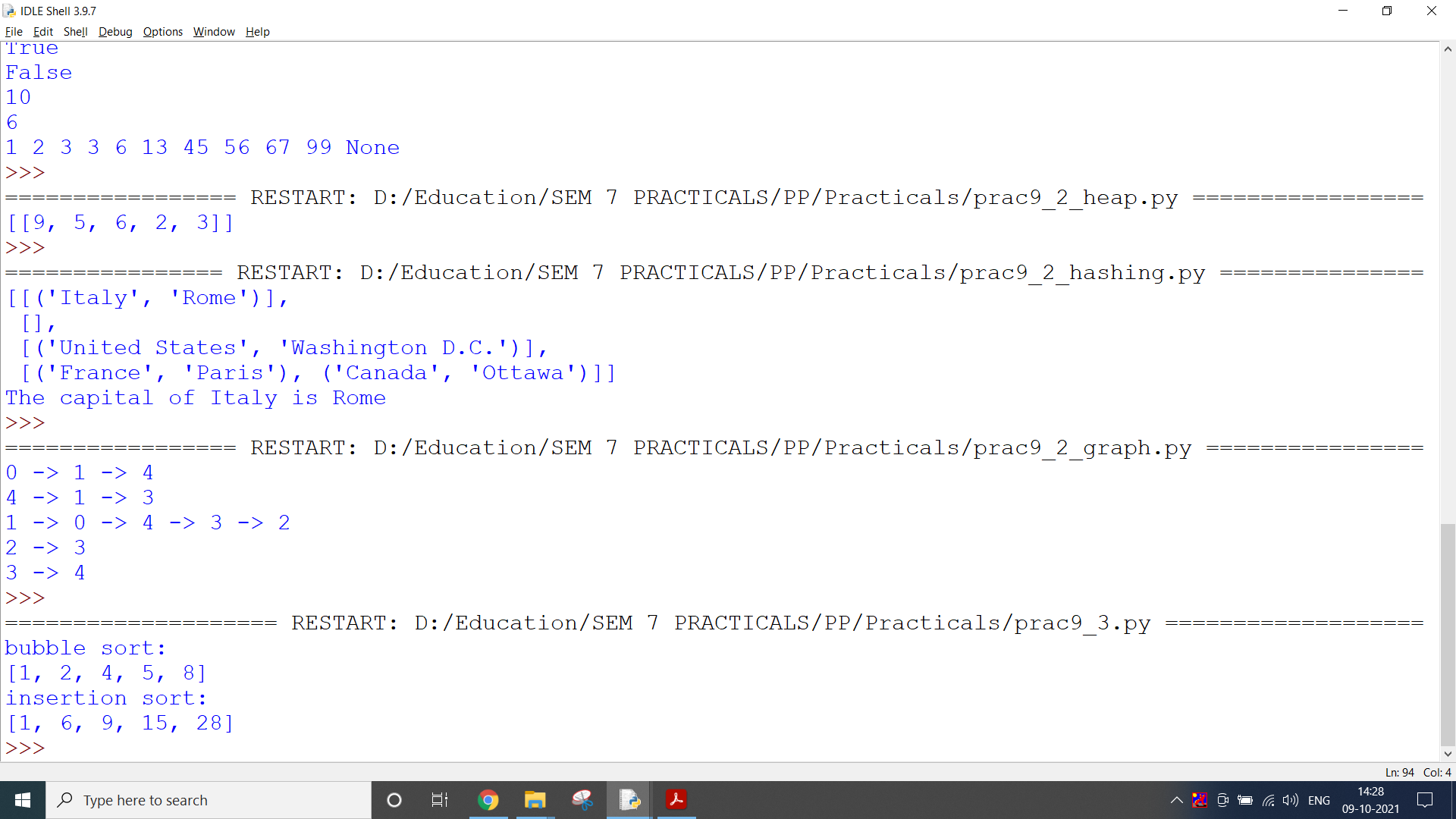
num = [9, 1, 15, 28, 6]

insertion\_sort(num)

print("insertion sort:")

print(num)

**OUTPUT:**



**CONCLUSION:**

In this practical, we implemented different sorting algorithms like bubble sort and insertion sort.

**PRACTICAL 9.4**

**AIM:**

Develop programs to understand working of exception handling and assertions.

**CODE:**

try:

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

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

if b == 0:

raise ArithmeticError

else:

print("a/b = ", a / b)

except ArithmeticError:

print("The value of b can't be 0")

def discount(price, discount\_percentage):

Final\_price = int(price \* (1 - (discount\_percentage / 100)))

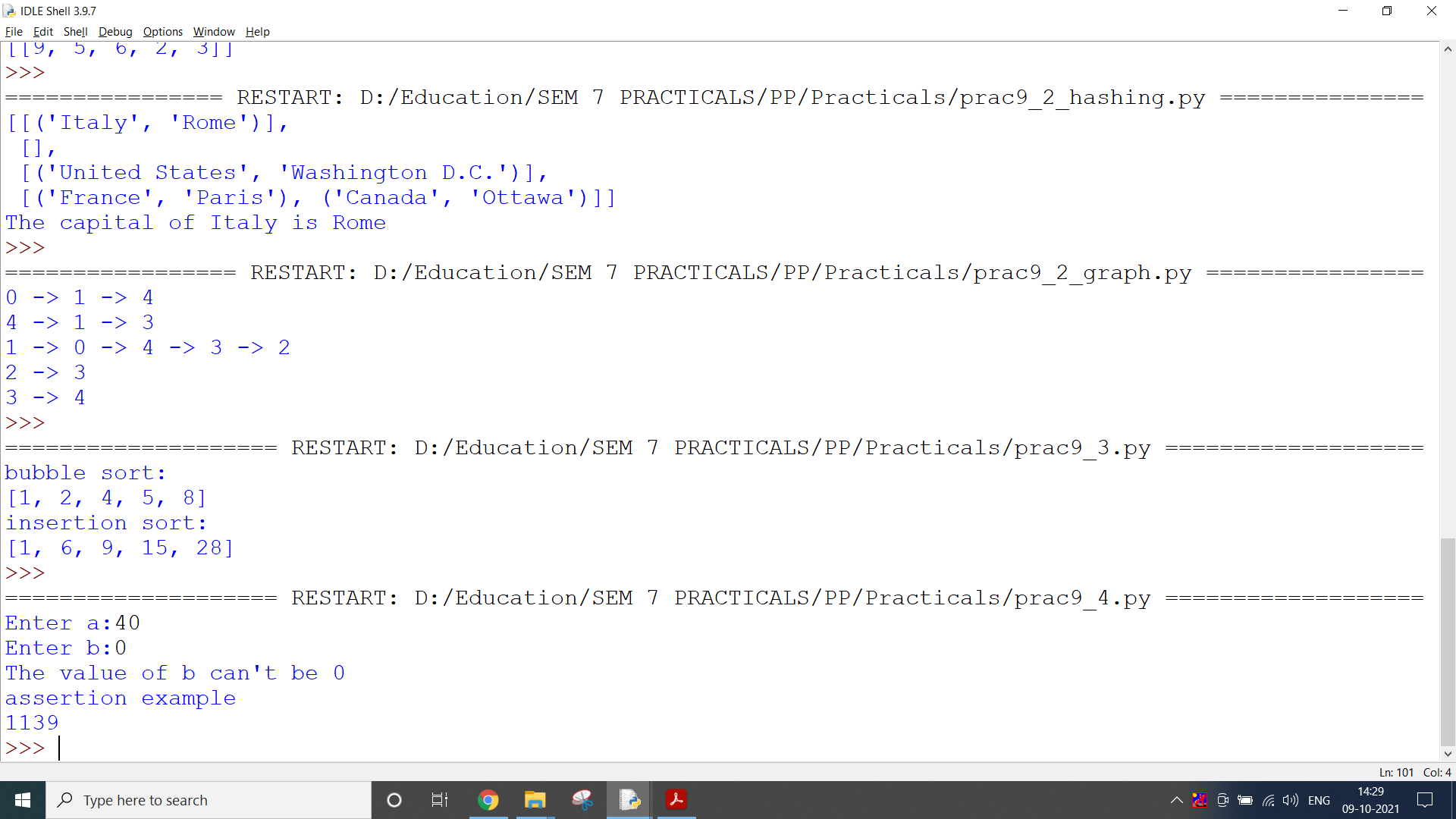
assert 0 <= Final\_price <= price

print(Final\_price)

print("assertion example")

discount(1212, 6)

**OUTPUT:**



**CONCLUSION:**

In this practical, we learnt about error handling using try and except.

**PRACTICAL 10.1**

**AIM:**

For Numpy library, create two 2D Numpy arrays with random numbers and concatenate them. After Concatenation, reshape the resulting Numpy array such that the number of rows and columns is reversed.

**CODE:**

import numpy as np

arr1 = np.random.randint(100,size=(3,3))

arr2 = np.random.randint(100,size=(3,3))

print(arr1, "\n \n", arr2)

print(type(arr1), type(arr2))

print(arr1.ndim, arr2.ndim)

print("\n\n----------Two ways to concatenate----------\n")

arr3 = np.concatenate((arr1, arr2), axis=1)

print(arr3)

arr4 = np.concatenate((arr1, arr2), axis=0)

print(arr4)

print("\n\n----------Reshaping----------\n")

print(arr1)

print(arr1.shape)

temp\_arr = arr1.reshape(9,1)

print(temp\_arr)

try:

temp\_arr = arr1.reshape(2,4)

except:

print(f"Cannot reshape array with {arr1.size} elements into 2 X 4 array")

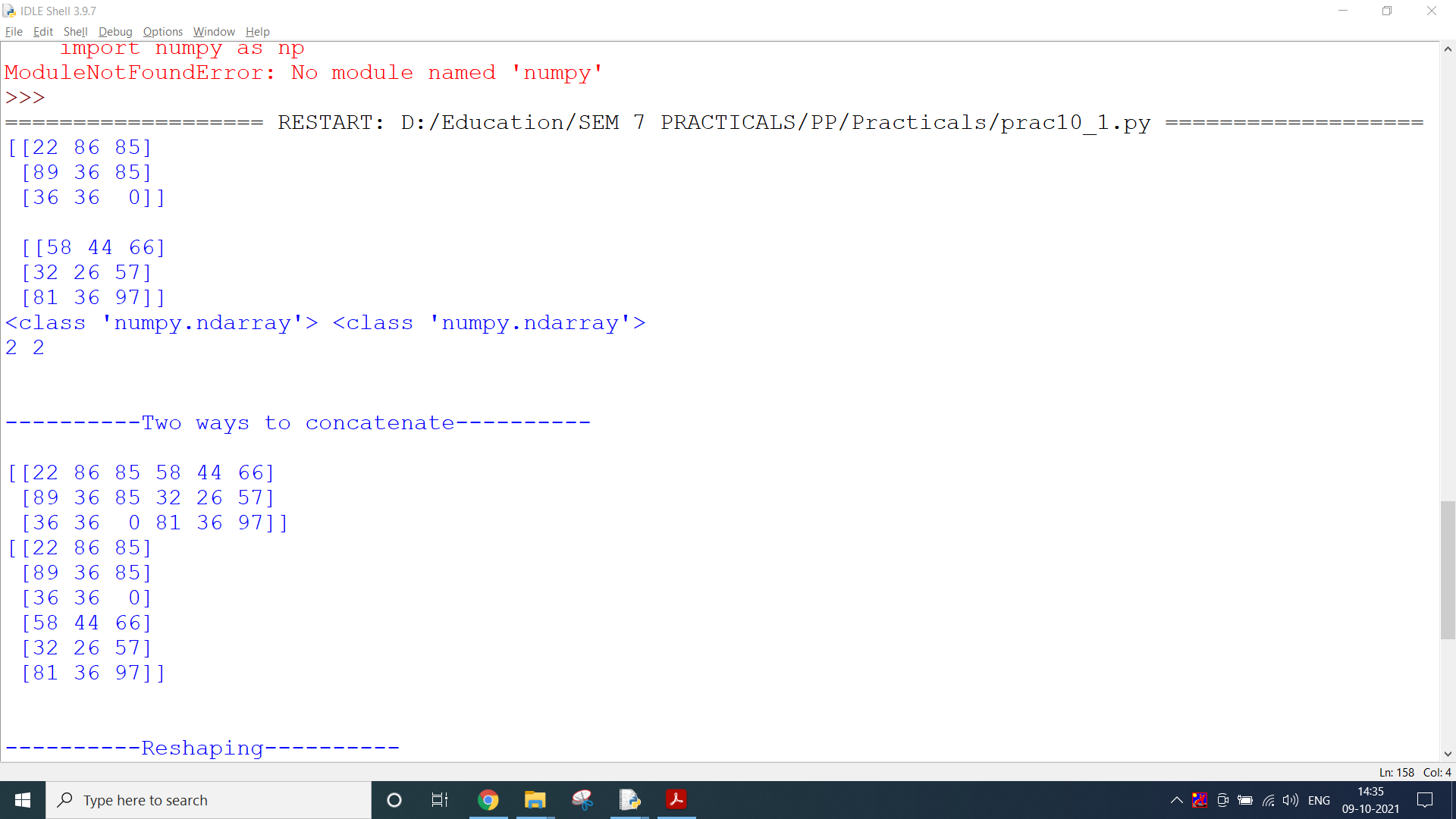
newarr = arr3.reshape(6, 3)

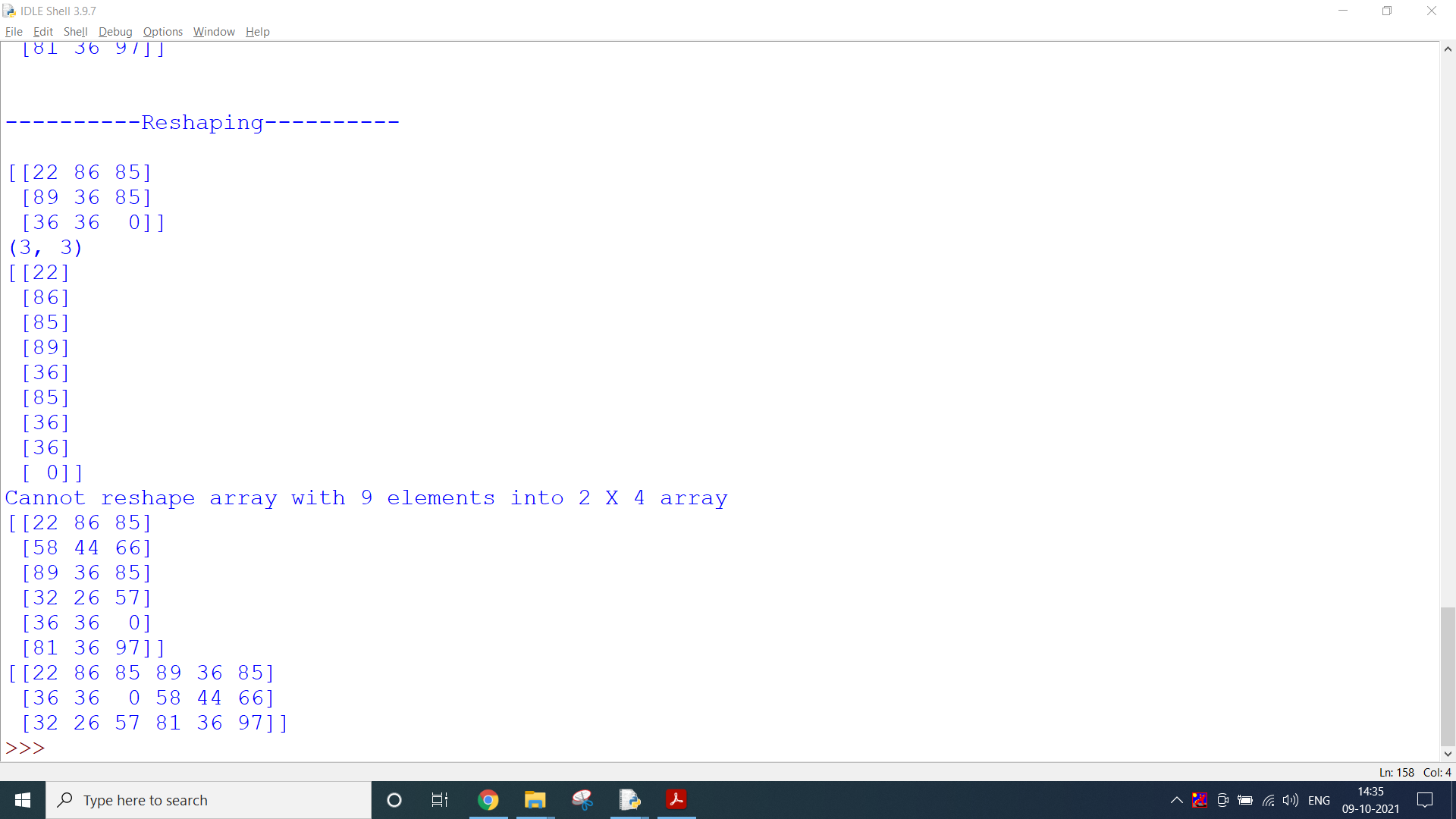
print(newarr)

newarr2 = arr4.reshape(3, 6)

print(newarr2)

**OUTPUT:**





**CONCLUSION:**

In this practical, we learnt about numpy library which is very useful complex mathematical operations.

**PRACTICAL 10.2**

**AIM:**

Create a Pandas series from a Python List. Find out the mean, median, mode, range and standard deviation of the series.

**CODE:**

import pandas as pd

s = pd.Series([1,2,3,4,5])

print(s)

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

s = pd.Series(our\_list)

print(s)

s = pd.Series([11,32,223,14,50], index=['a','b','c','d','e'])

print(s)

print(s.mean())

print(s.median())

print(s.mode())

s2 = pd.Series([11,11,22,11,50], index=['a','b','c','d','e'])

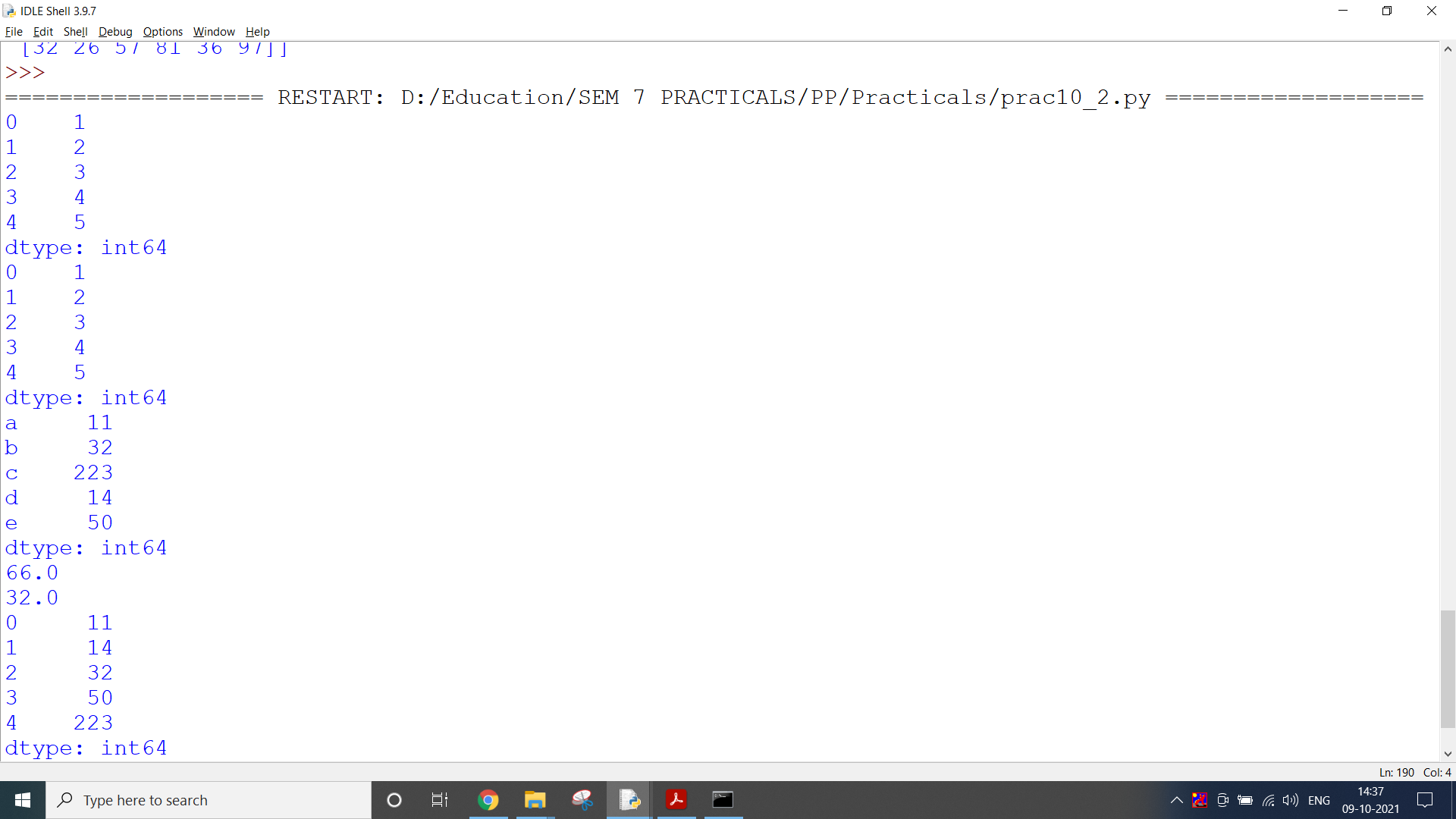
print(s2.mode())

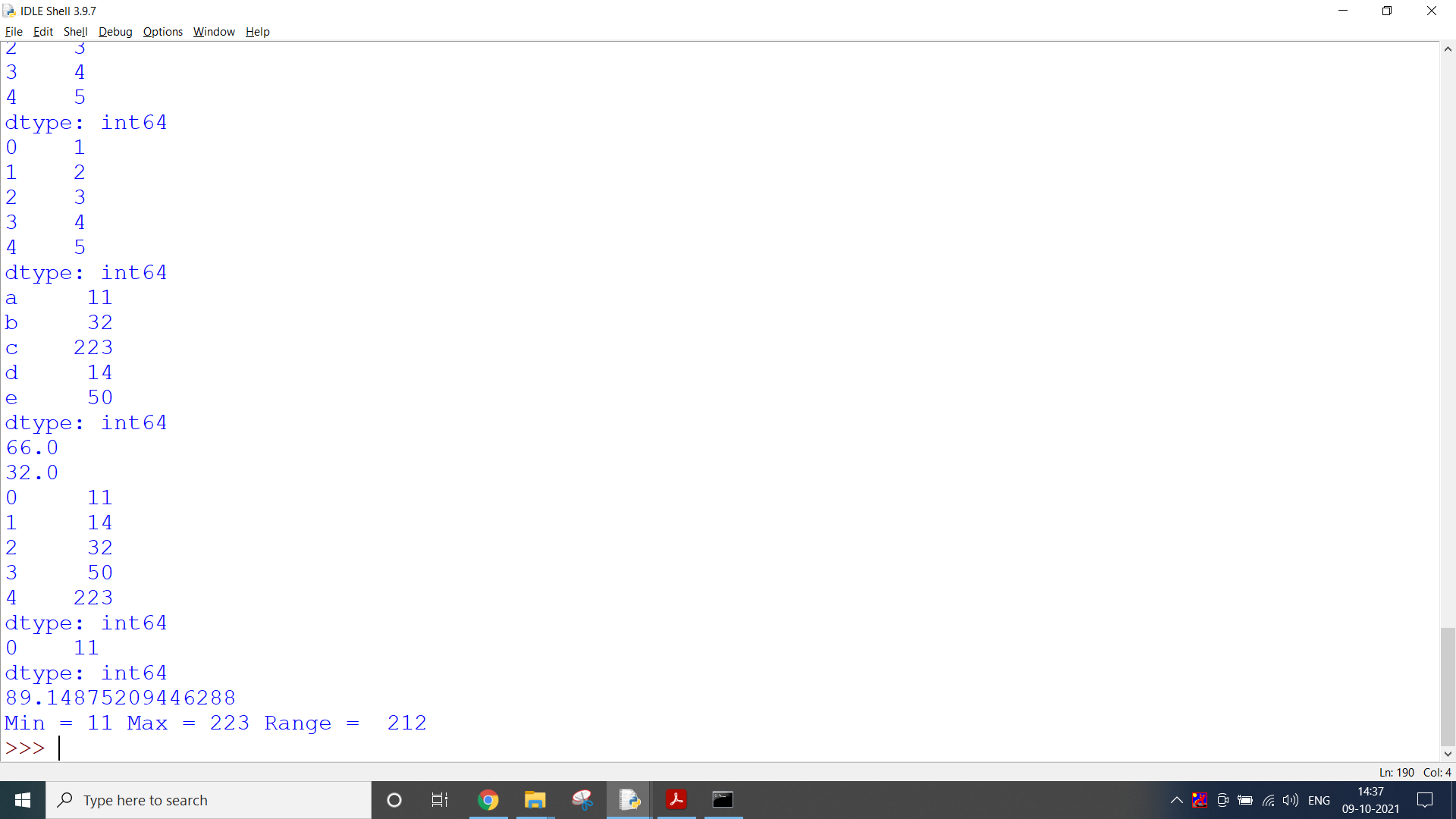
print(s.std())

range = s.max() - s.min()

print("Min =", s.min(),"Max =", s.max(),"Range = ", range)

**OUTPUT:**





**CONCLUSION:**

In this practical, we learnt about pandas and performed different statistical operations on data using it.

**PRACTICAL 10.3**

**AIM:**

Create a Pandas Dataframe each from: a) Python Dictionary b) CSV File c) JSON File and store the DataFrames in Python Pickle Format.

**CODE:**

import pandas as pd

df = pd.DataFrame([1,2,3,4])

df

data = [['Alex',10],['Bob',12],['Clarke',13]]

df = pd.DataFrame(data,columns=['Name','Age'])

df

dict = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}

df\_a = pd.DataFrame(dict)

print(df\_a)

dict = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}] #List of Dicts

temp\_df = pd.DataFrame(dict)

temp\_df

# Create URL to CSV file (alternatively this can be a filepath in your Google DRIVE)

url = 'https://raw.githubusercontent.com/cs109/2014\_data/master/countries.csv'

# Load the first sheet of the JSON file into a data frame

df\_b = pd.read\_csv(url)

# View the first ten rows

df\_b.head(10)

# Create URL to JSON file (alternatively this can be a filepath in your Google DRIVE)

url = 'https://raw.githubusercontent.com/chrisalbon/simulated\_datasets/master/data.json'

# Load the first sheet of the JSON file into a data frame

df\_c = pd.read\_json(url)

# View the first ten rows

df\_c.head(10)

df\_a.to\_pickle('df\_a.pickle')

del df\_a #deleting the original dataframe

df\_b.to\_pickle('df\_b.pickle')

del df\_b

df\_c.to\_pickle('df\_c.pickle')

del df\_c

try:

print(df\_a)

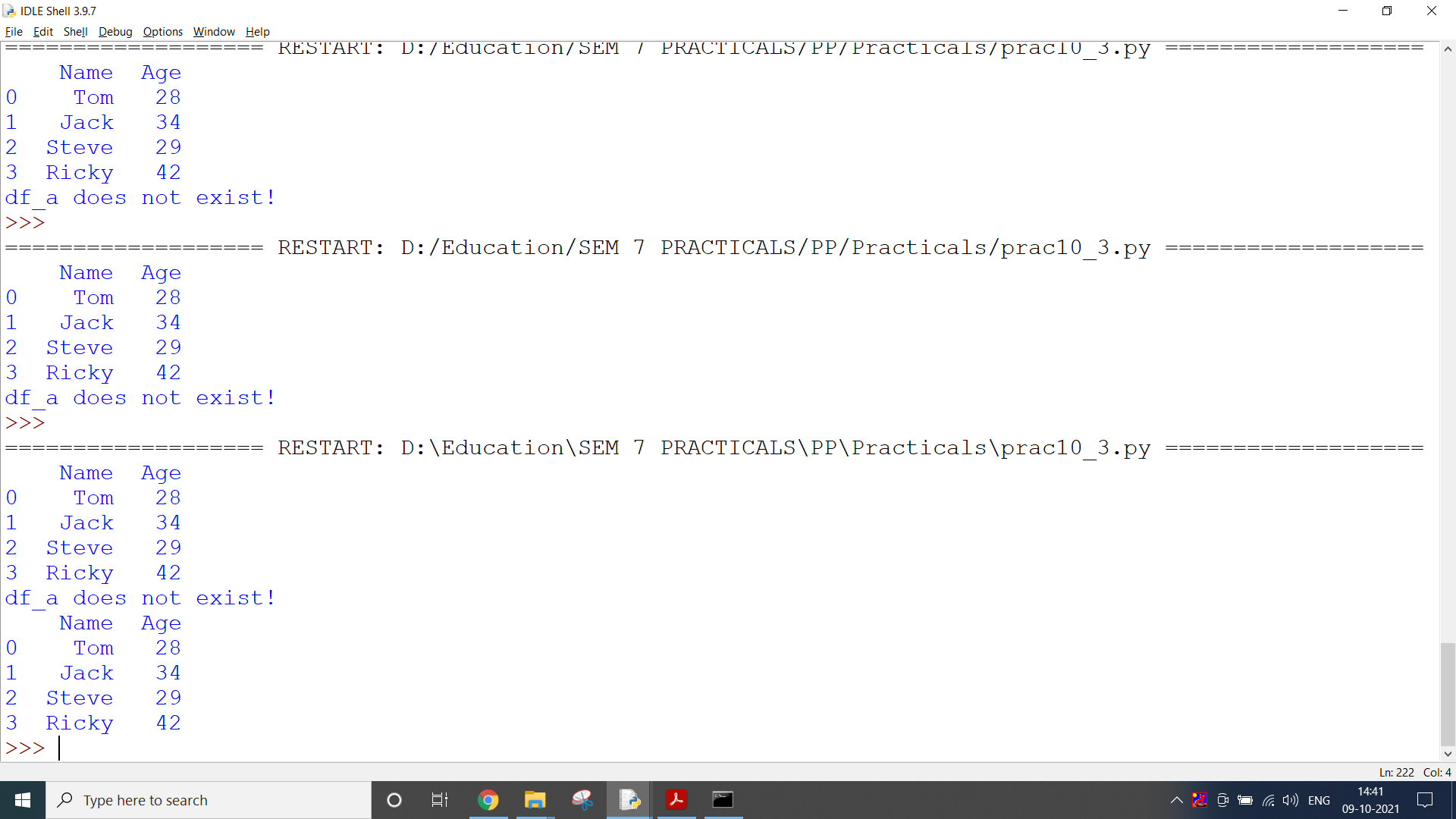
except:

print("df\_a does not exist!")

pickle\_to\_df\_demo = pd.read\_pickle('df\_a.pickle')

print(pickle\_to\_df\_demo)

**OUTPUT:**



**CONCLUSION:**

In this practical, we manipulated data into different formats like CSV and JSON using pandas.

**PRACTICAL 10.4**

**AIM:**

1. Develop a program to scrap the data related to laptop prices from the flipkart.
2. Develop a program to scrap the data of Covid-19 from the given URL and store it in CSV file and Database.

**CODE:**

*(i)*

from bs4 import BeautifulSoup

import requests

import pandas as pd

# URL of the website which you want to scrape

req = requests.get(

"https://www.flipkart.com/search?q=laptops&otracker=search&otracker1=search&marketplace=FLIPKART&as-show=on&as=off")

content = req.content # Get the content

soup = BeautifulSoup(content, 'html.parser')

#print(soup.prettify())

desc = soup.find\_all('div', class\_='\_4rR01T') # get class name using inspect element

print(desc)

descriptions = [] # Create a list to store the descriptions

for i in range(len(desc)):

descriptions.append(desc[i].text)

len(descriptions)

common\_class = soup.find\_all('li', class\_='rgWa7D')

#common\_class

# Create empty lists for the features

processors = []

ram = []

os = []

storage = []

inches = []

warranty = []

for i in range(0, len(common\_class)):

p = common\_class[i].text # Extracting the text from the tags

if("Processor" in p):

processors.append(p)

elif("RAM" in p):

ram.append(p)

# If RAM is present in the text then append it to the ram list.

# Similarly do this for the other features as well

elif("HDD" in p or "SSD" in p):

storage.append(p)

elif("Operating" in p):

os.append(p)

elif("Display" in p):

inches.append(p)

elif("Warranty" in p):

warranty.append(p)

print(len(processors))

print(len(os))

print(len(ram))

# inaccurate

# print(len(warranty))

if len(inches)>24:

inches = inches[0:24]

print(len(inches))

price = soup.find\_all('div', class\_='\_30jeq3 \_1\_WHN1')

# Extracting price of each laptop from the website

prices = []

for i in range(len(price)):

prices.append(price[i].text)

len(prices)

rating = soup.find\_all('div', class\_='\_3LWZlK')

# Extracting the ratings of each laptop from the website

ratings = []

for i in range(len(rating)):

ratings.append(rating[i].text)

if len(ratings)>24:

ratings = ratings[0:24]

len(ratings)

df = {'Description': descriptions, 'Processor': processors, 'RAM': ram, 'Operating System': os,

'Storage': storage, 'Display': inches, 'Price': prices, 'Ratings': ratings}

dataset = pd.DataFrame(data=df)

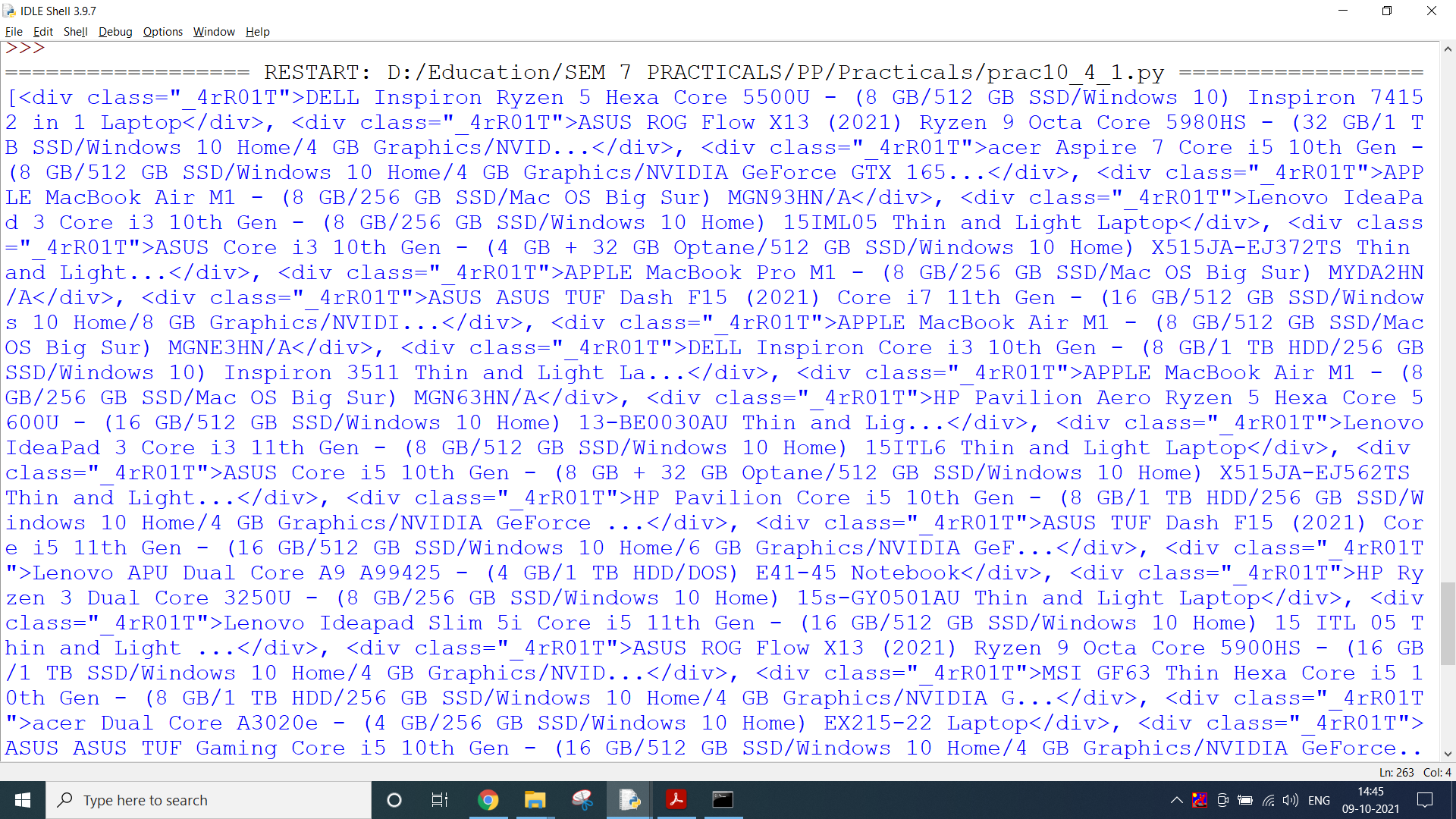
print(dataset)

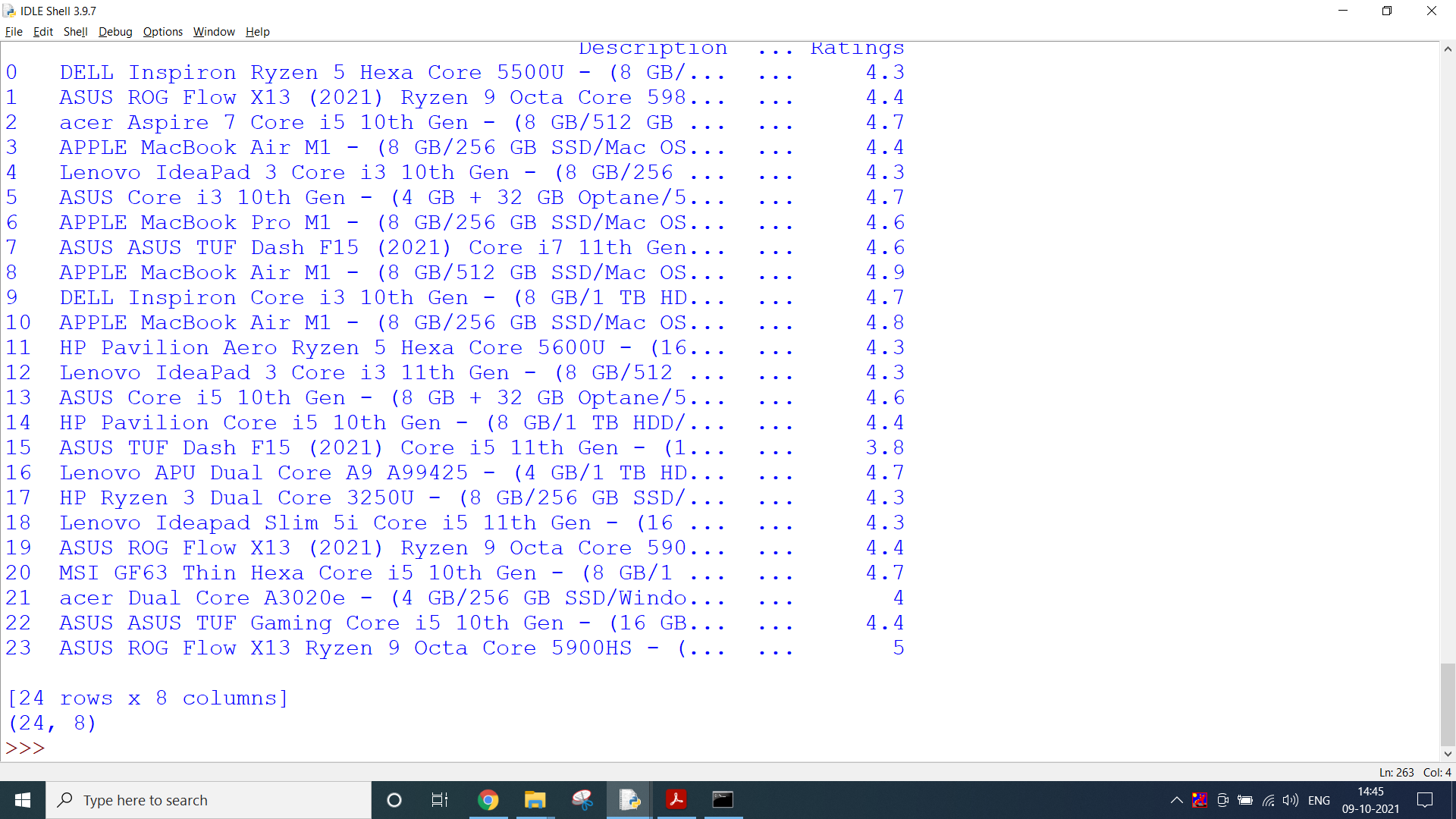
dataset.to\_csv('laptops.csv',index=False)

df = pd.read\_csv('laptops.csv')

print(df.shape)

**OUTPUT:**





**CODE:**

*(ii)*

import requests

import pandas as pd

# URL of the website which you want to scrape

req = requests.get(

"https://api.covidtracking.com/v1/us/daily.json")

json\_data = req.json() # Get the content

type(json\_data)

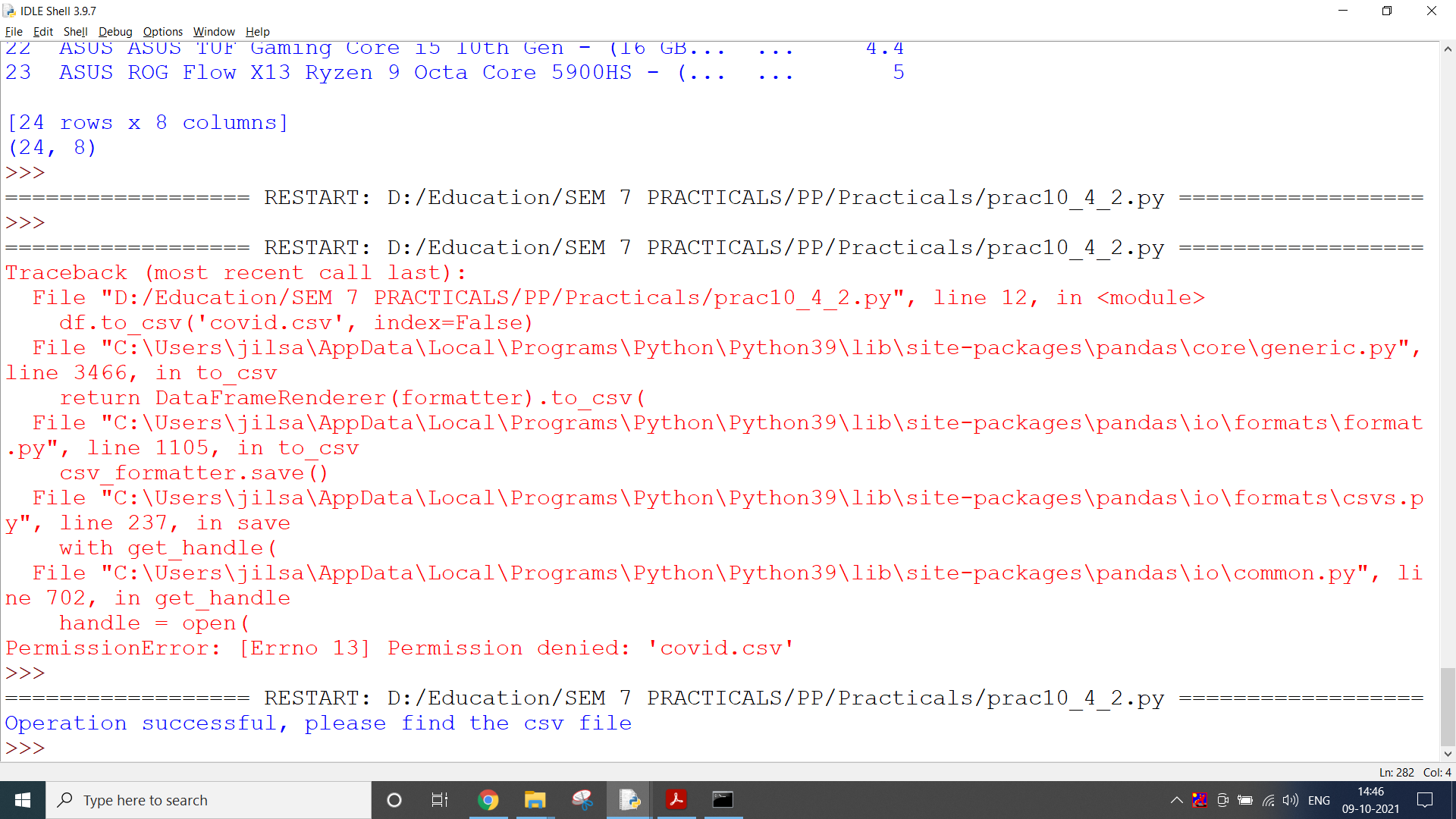
df = pd.json\_normalize(json\_data)

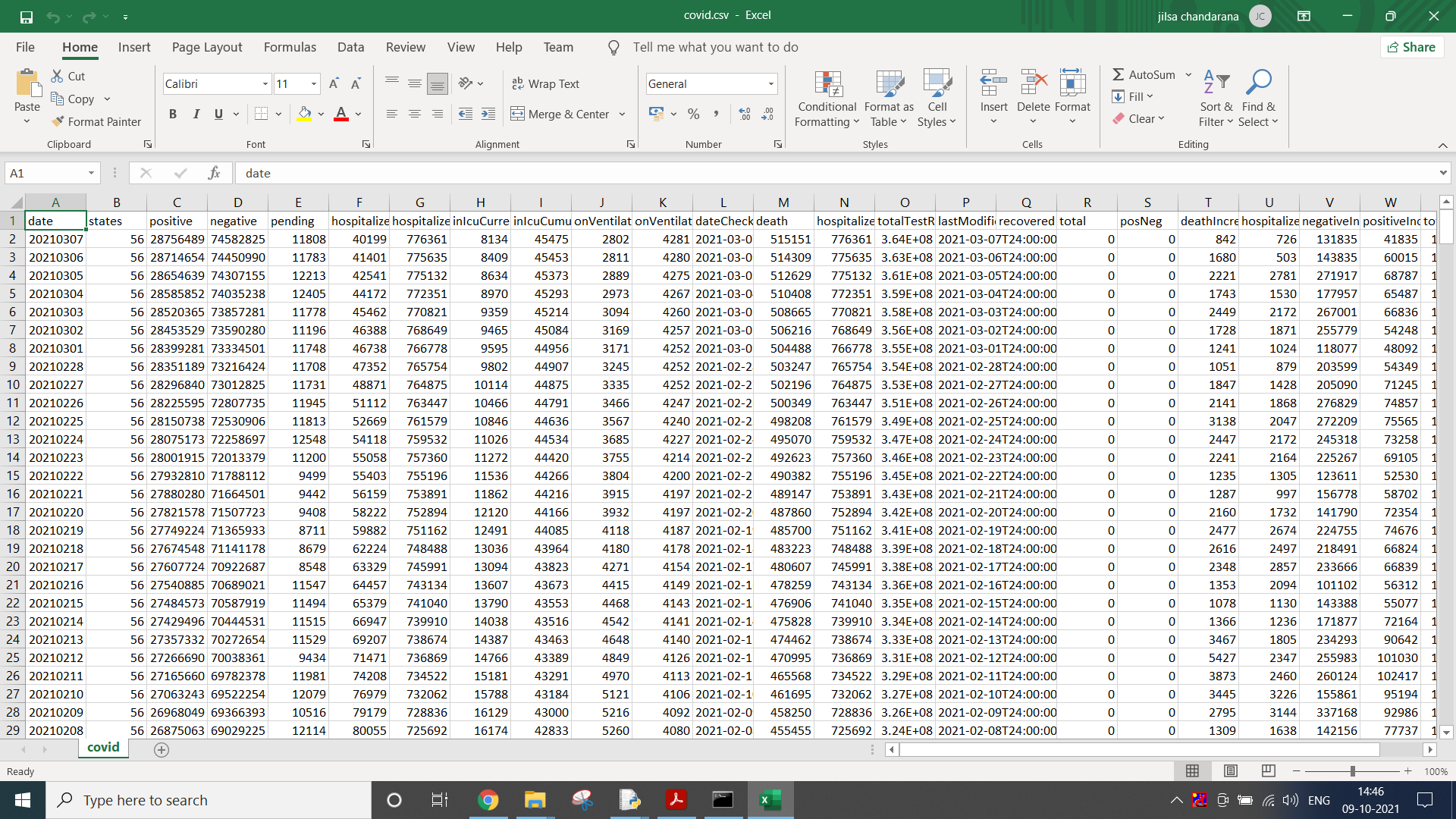
# save to csv

df.to\_csv('covid.csv', index=False)

print("Operation successful, please find the csv file")

**OUTPUT:**





**CONCLUSION:**

In this practical, we scrapped the data from website and stored it in a CSV file.