

**PYTHON LAB FILE**

**Submitted to: Dr. Varun Sapra**

**Submitted by: Deepanshu Rawat [500097692]**

**Batch: B46**

**EXPERIMENT -1: Programming Sequential Logic**

**Objective: To code sequential logic in Python language**

Q1. Given an integer n, perform the following conditional actions:

· If n is odd, print Weird

· If n is even and in the inclusive range of 2 to 5, print Not Weird

· If n is even and in the inclusive range of 6 to 20, print Weird

· If n is even and greater than 20, print Not Weird

Q2.WAP to read an integer ‘n’ from STDIN. For all non-negative integers i<n, print i^2 on a separate line.

Q3. WAP to read an integer from STDIN. Without using any string methods, print the following on a single line:123…n

Solution:

“if” condition: “if” statement is one of the decision-making statements. The purpose of “if” statement is told whether the following block of code will execute or not. The condition is checked whether true or false. If “true” then the following block of code is executed otherwise not.

Syntax for “if” condition:

if <condition>:

//giving an indentation is necessary

//block of code to be executed

//other statements

“for” loop: “for” loop is one of the iterative statements. The purpose of “for” loop is to execute the same block of code over and over till the initial condition fails.

Syntax for “for” loop:

for <variable> in range (<starting limit>, <end limit>, <increment/decrement>):

//giving an indentation is necessary

//block of code to be executed

//other statements

“while” loop: “while” loop is one the iterative statement. The purpose of “while” loop is to execute the same block of code over and over till the initial condition fails.

Syntax for “while” loop:

while <condition>:

//giving an indentation is necessary

//block of code to be executed

//increment/decrement statement if any

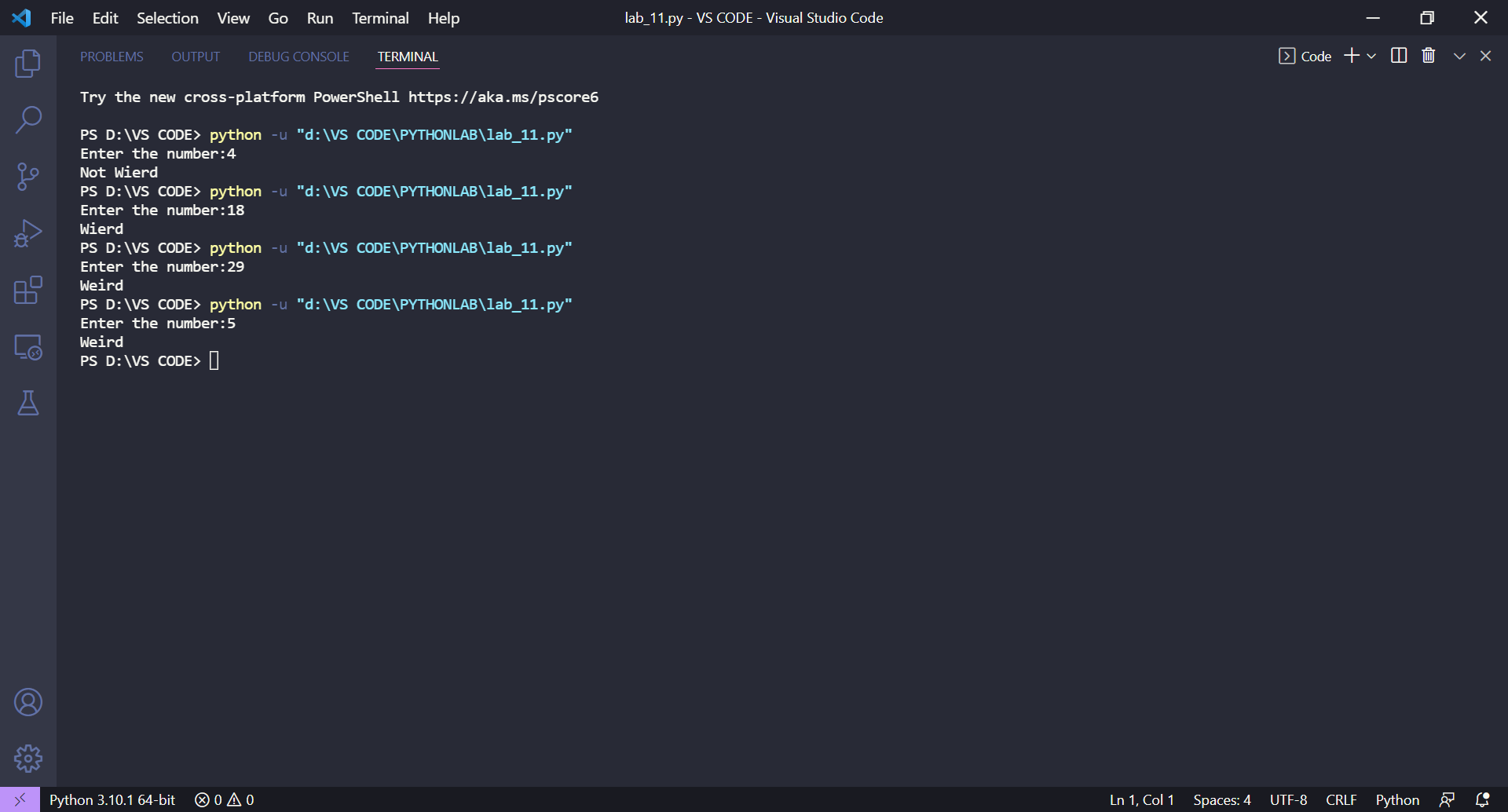
//other statements

Solution Prog:1

Code:

n = int(input("Enter the number:"))  
if n % 2 != 0:  
 print("Weird")  
else:  
 if 2 <= n <= 5:  
 print("Not Wierd")  
 if 6 <= n <= 20:  
 print("Wierd")  
 if n > 20:  
 print("Not Wierd")

Output:

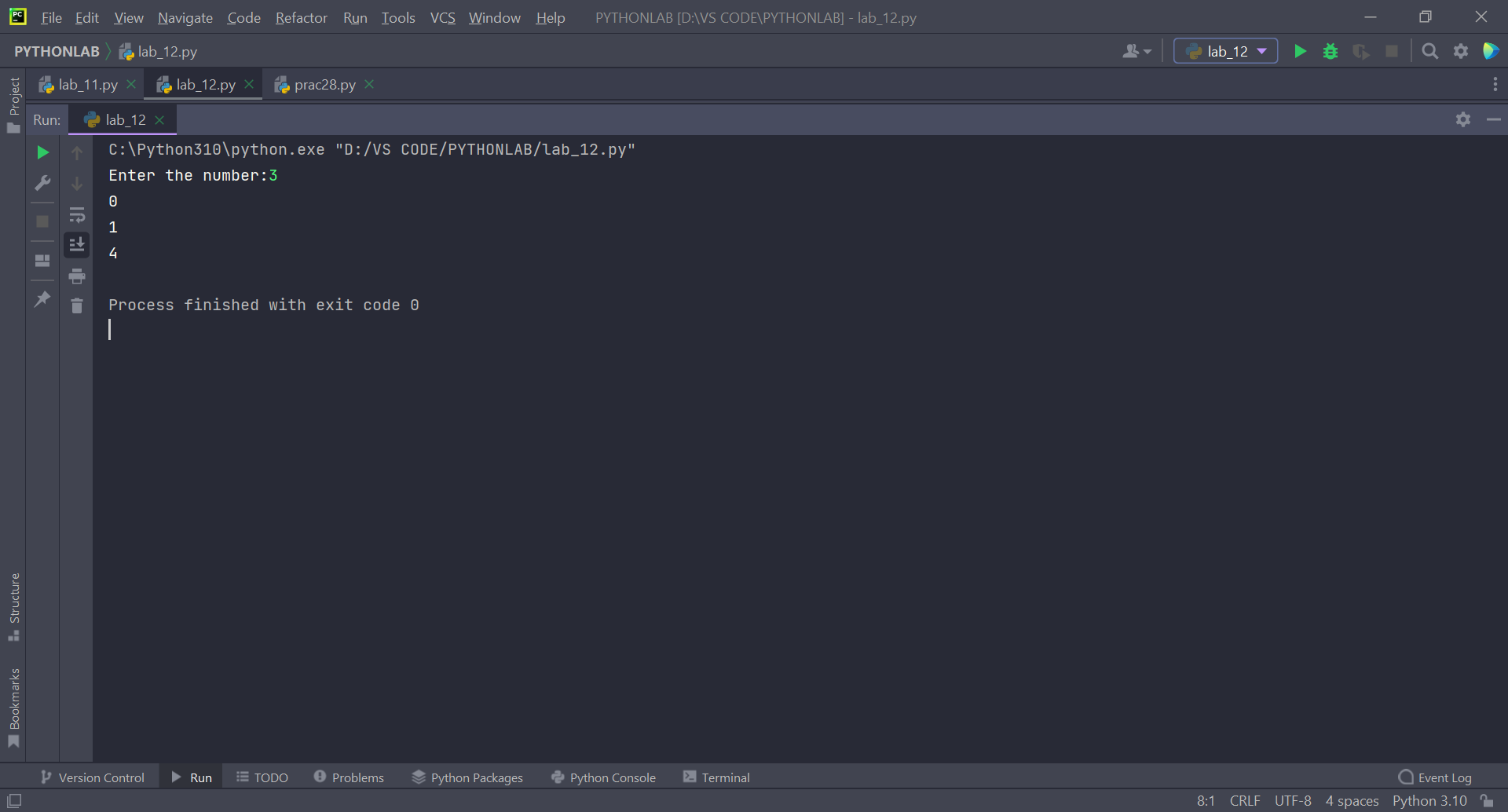


Solution Prog:2

Code:

n = int(input("Enter the number:"))  
for i in range(0, n):  
 print(i \* i)

Output:

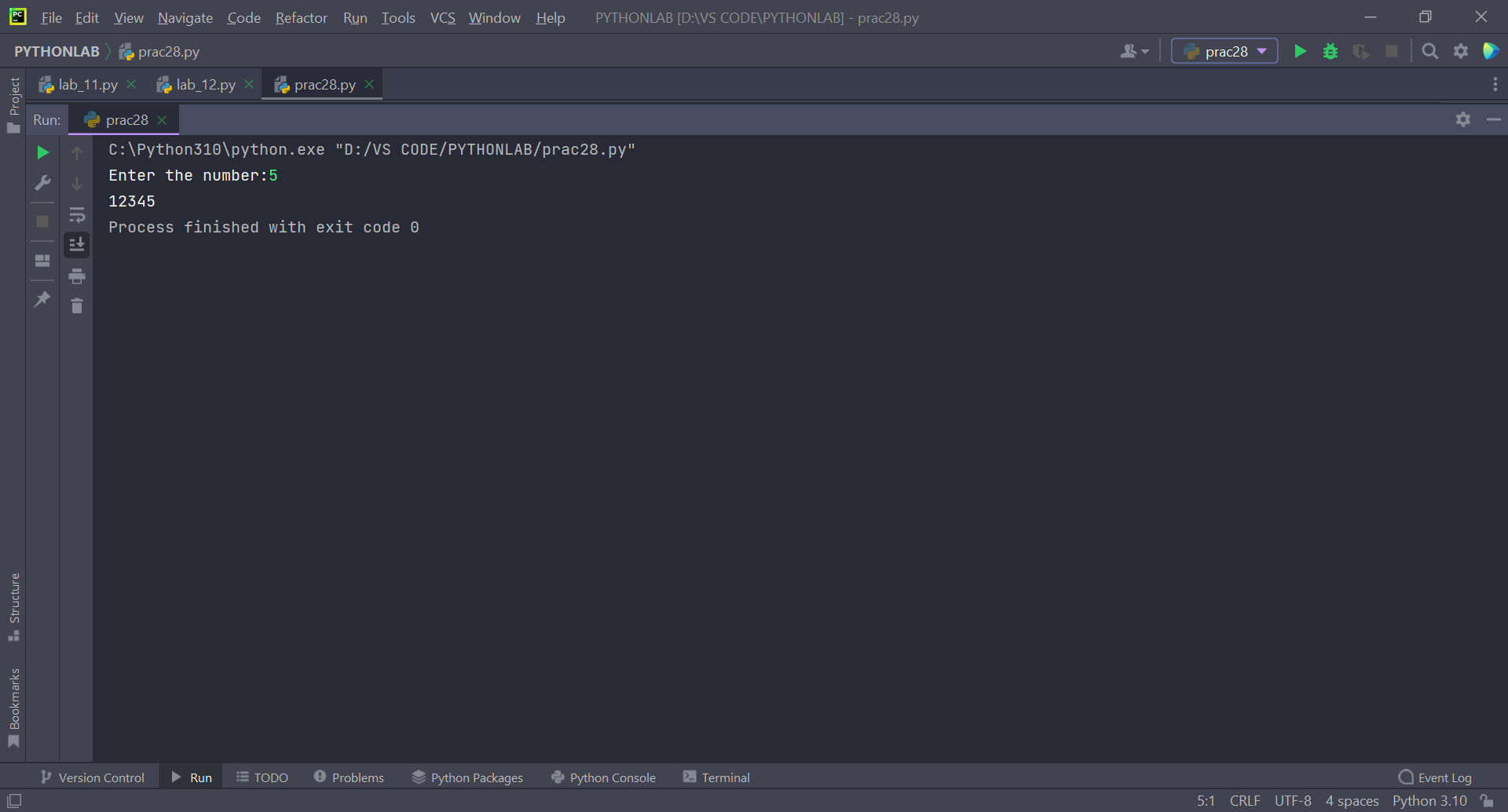


Solution Prog:3

Code:

n = int(input("Enter the number:"))  
for i in range(1, n + 1):  
 print(i, end=' ')

Output:



**EXPERIMENT 2 & 3**

**Objective: To understand the concept of list, manipulating list elements, tuples, and dictionaries in python.**

Q1. WAP to read the record of *n* students in a dictionary containing key/value pairs of names: [marks]. Print the average of the marks obtained by the particular student correct to 2 decimal places.

Q2. WAP to input a list of scores for N students in a list data type. Find the score of the runner-up and print the output.

Q3. Rupal has a huge collection of country stamps. She decided to count the total number of distinct country stamps in her collection. She asked for your help. You pick the stamps one by one from a stack of country stamps. Find the total number of distinct country stamps using a suitable data type.

Solution:

List: List are those data types which store multiple items in a single variable. Lists are mutable, the data stays in an ordered form and allows duplicate items.

Implementation: lst=[<value1>, <value2>…. <valueN>] #Sqaurebrackets for Lists

Tuples: Tuples are those data types which store multiple items in a single variable. They are immutable, the data stays in an ordered form and allows duplicate items.

Implementation: a=(<value1>, <value2>…., <valueN>) #Roundbrackets for Tuples

Dictionary: Dictionary are those data types with store values on the basis of “key: value pairs”. They are mutable, the data stays in an ordered form and does not allow duplicate values.

Implementation: d={“<key1>”: <value-pair1>, “<key2>” :<value-pair2>………, “<keyN>” :<value-pairN>}

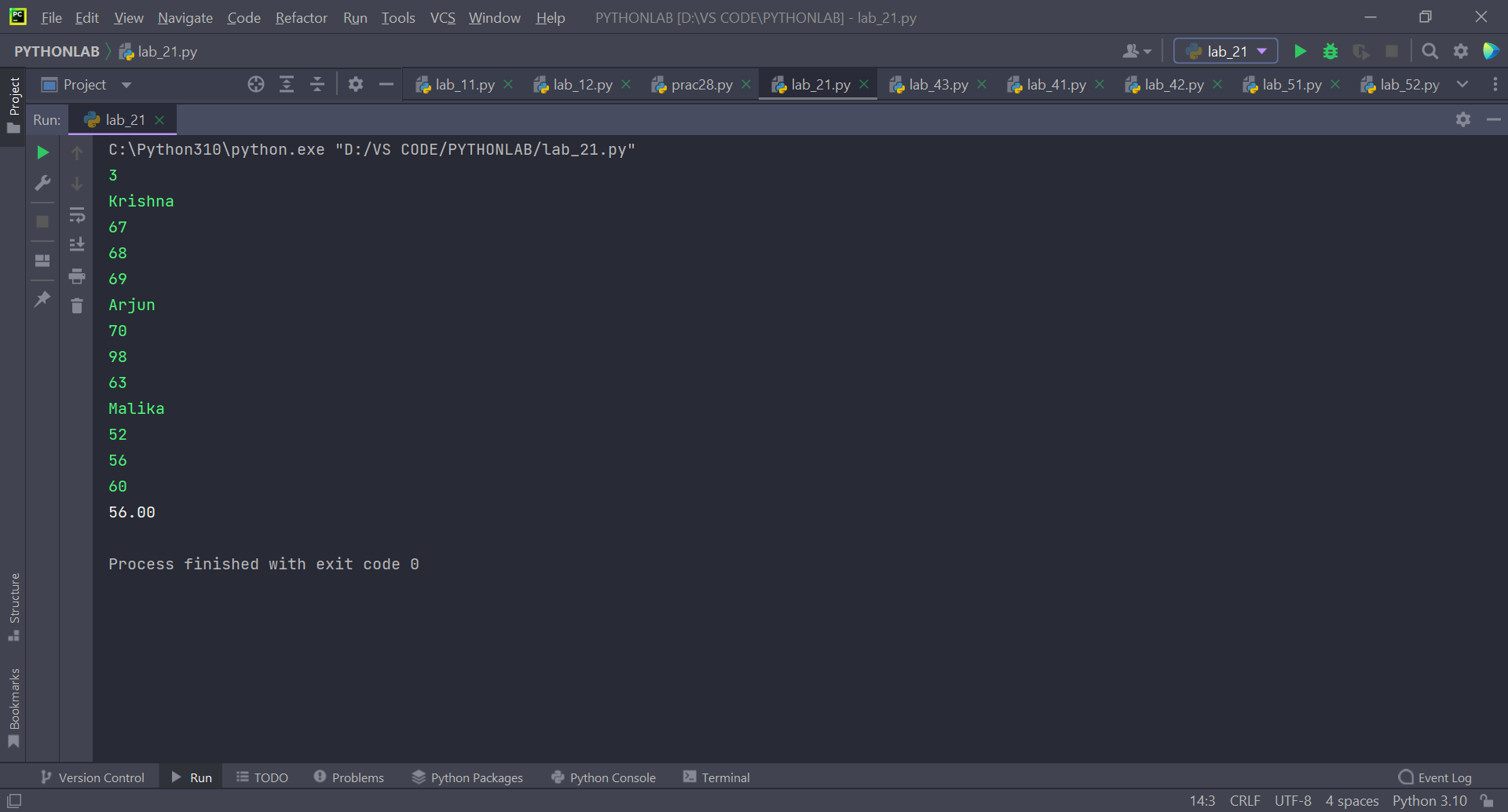
#Curlybrackets for Dictionary

Solution Prog 1:

Code:

dict1 = {}  
list1 = []  
n = int(input())  
i = 0  
name = ""  
for i in range(n):  
 name = input()  
 for j in range(n):  
 num = int(input())  
 list1.append(num)  
 dict1[name] = list1  
 list1 = []  
list2 = dict1[name]  
print("%.2f" % (sum(list2) / n))

Output:

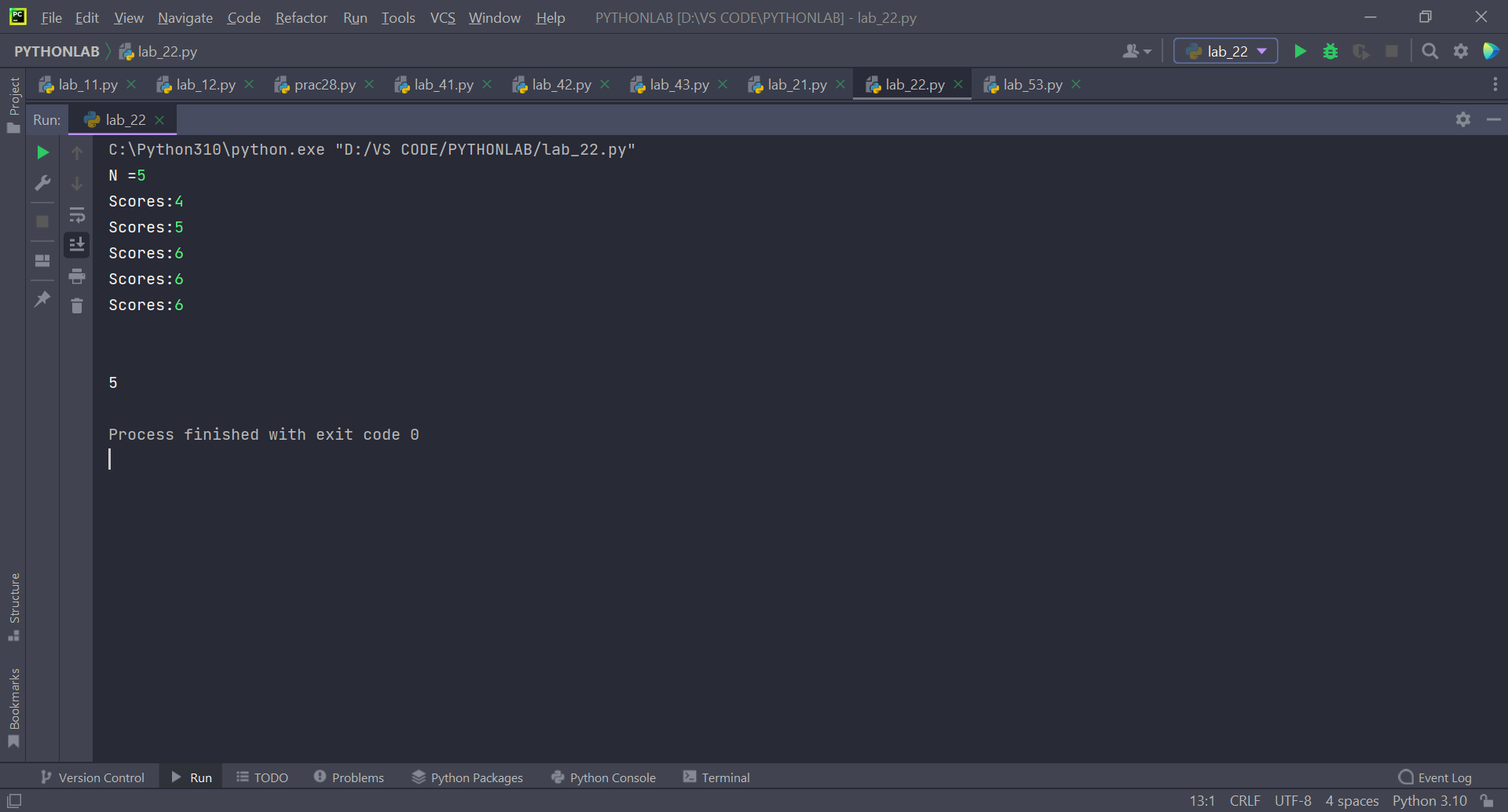


Solution Prog 2:

Code:

n = int(input("N ="))  
list1 = []  
maximum = 0  
for i in range(0, n):  
 element = int(input("Scores:"))  
 list1.append(element)  
 if element > maximum:  
 maximum = element  
list1.sort()  
for i in range(n - 1, -1, -1):  
 if maximum != list1[i]:  
 maximum = list1[i]  
 break  
print("\n")  
print(maximum)

Output:

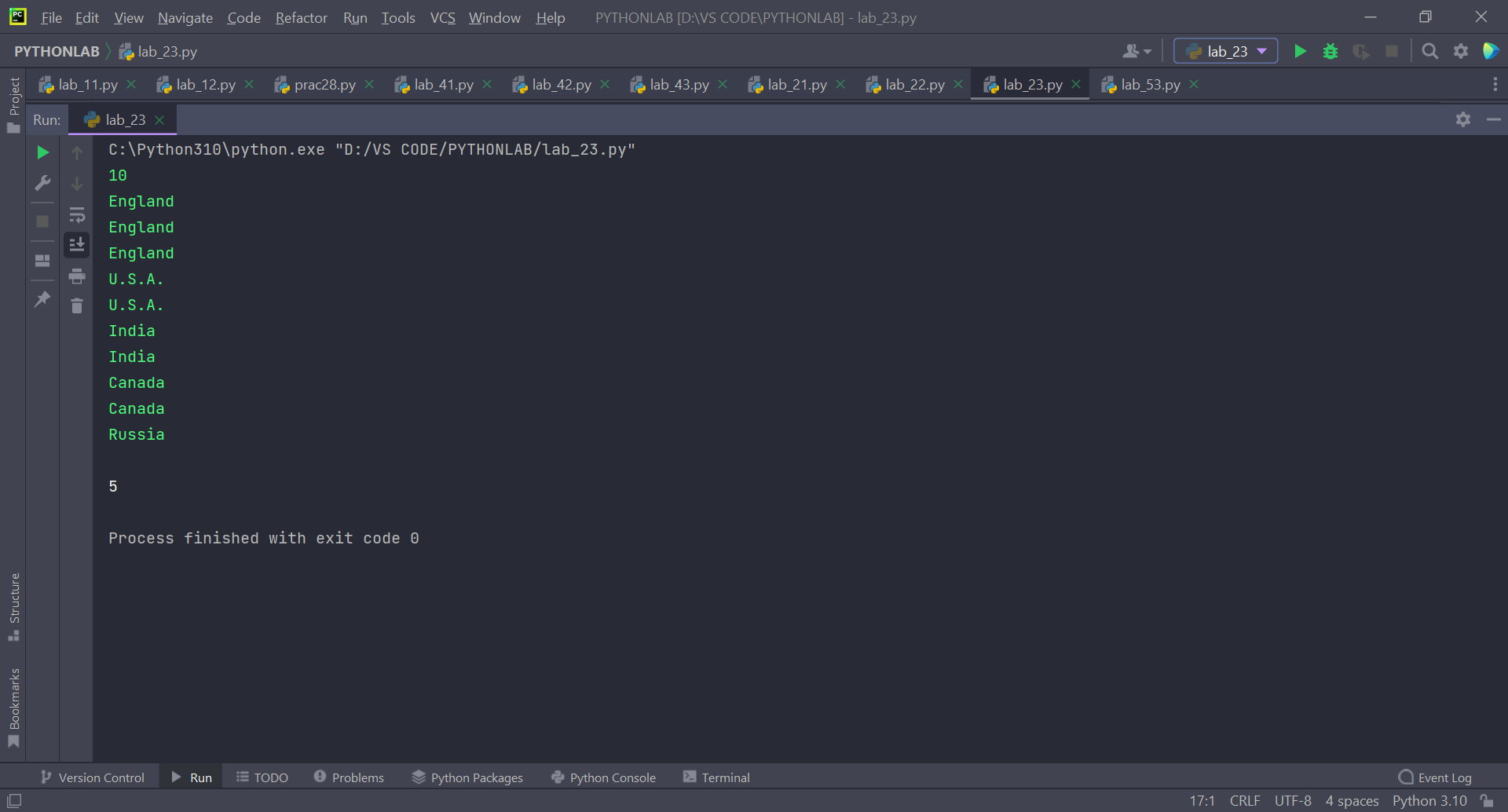


Solution Prog 3:

Code:

set1 = set()  
for i in range(int(input())):  
 set1.add(input())  
print()  
print(len(set1))

Output:



**EXPERIMENT -4: To understand the concept of strings in python.**

**Objective: To understand the concept of strings, and related functions in it.**

List of Lab Activities:

Q1. WAP to enter a string and a substring. You have to print the number of times that the substring occurs in the given string. String traversal will take place from left to right, not from right to left.

Q2. WAP to input the first name, middle and last name of a person. Your task is to print the initials of the first and middle name separated by a dot (.)

The last name should be followed by a dot and a space where the first letter is capital.

Q3. Given a string containing both upper- and lower-case alphabets. Write a Python program to count the number of occurrences of each alphabet (case insensitive) and display the same.

Solution:

Strings: Strings is a progression of characters. They are enclosed either by single quotes, or double quotes. Strings in python are immutable, characters stored in them stays in an ordered form.

Example: a = “Hello World!” or a = ‘Hello World!’ #Both are same

A few Inbuilt String Functions:

1. partition (): It is used to search for a particular string and split the original string into three parts and store all three values in a tuple.

Example: str = “My name is Zephyrus”

a = str.partition(“is”)

print(a) #Output-> (‘My name’ , ‘is’, ‘Zephyrus’)

1. endswith(): It checks if a string ends with a particular value or not. . It can be used with a range (starting point and ending point). Unlike find (), it returns false if value not found.

Syntax: <string\_name>. endswith(<value>)

1. strip (): It is used to trim the spaces from both sides of the given string. Syntax: <string\_name>. strip (<value to be removed[optional]>)
2. split (): It is used to split a string into a list. When can change the elements of the list on the basis of the max repetition of a particular value .

Syntax: <string\_name>. split(<value\_seperator>, <max\_split>)

1. index (): It is used to return the first occurrence of a specified value. It can be used with a range (starting point and ending point). Unlike find (), it returns an exception when no element is found.

Example: str = “My name is Zephyrus”

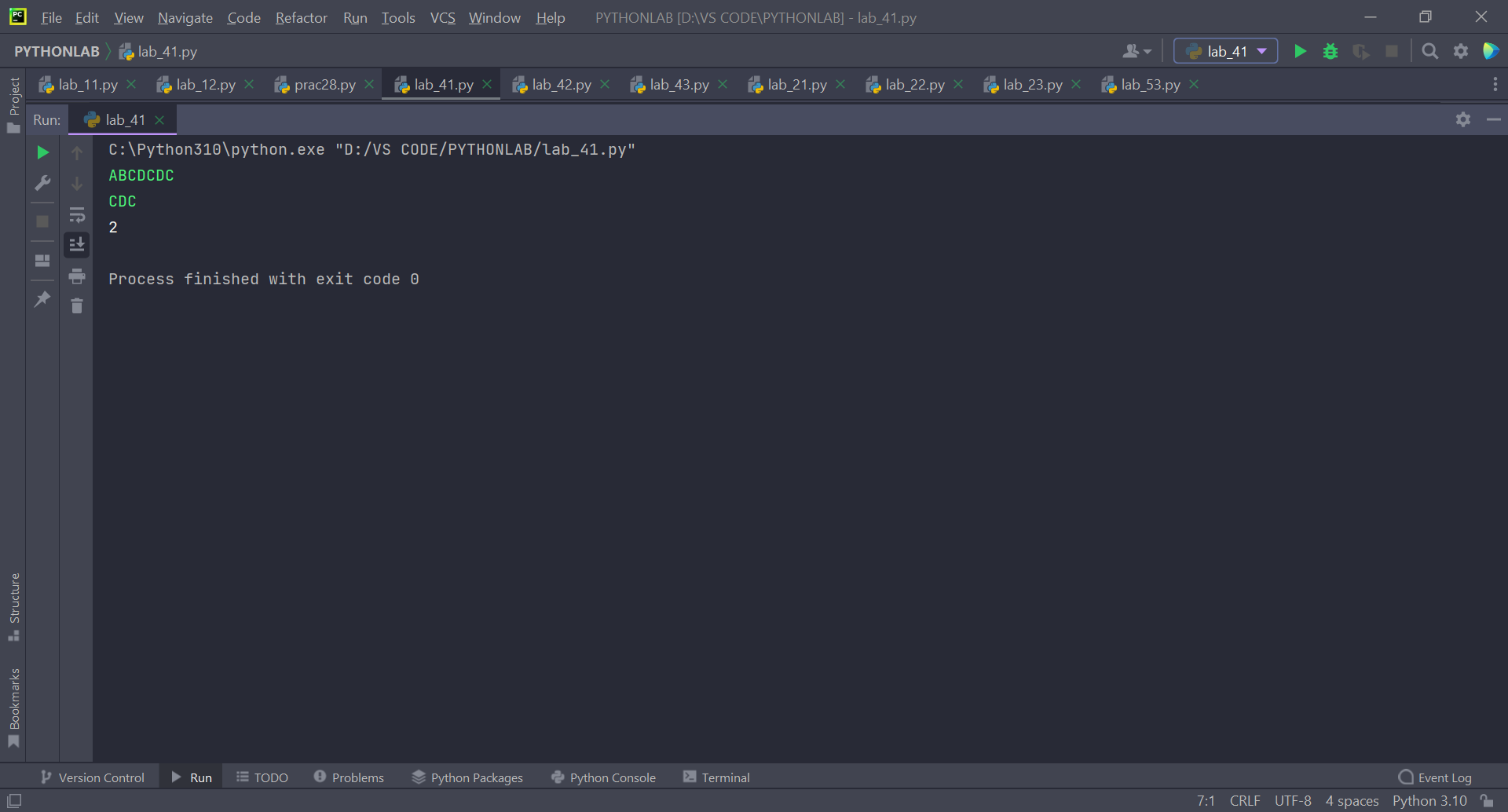
a = str.index(‘a’) print(a) #Output-> 4

Solution Prog 1:

Code:

str1 = input()  
sub\_str = input()  
c = 0  
a = len(sub\_str)  
for i in range(0, len(str1)):  
 if str1[i:a] == sub\_str:  
 c = c + 1  
 a = a + 1  
 if a > len(str1):  
 break  
print(c)

Output:

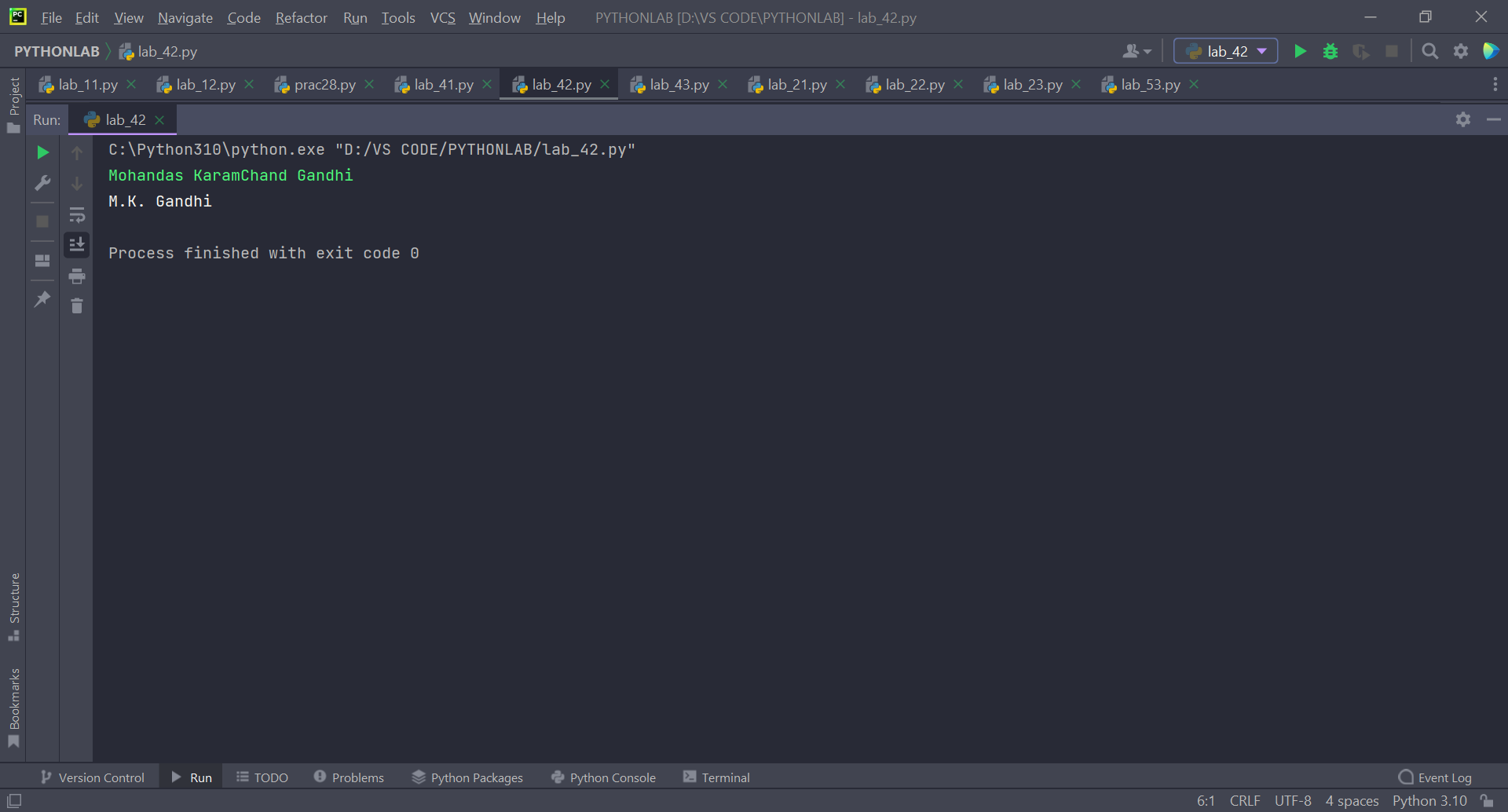


Solution Prog 2:

Code:

str1 = input()  
s = str1.split()  
new\_str = ""  
for i in range(0, len(s) - 1):  
 str1 = s[i]  
 new\_str = new\_str + (str1[0].upper() + '.')  
new\_str = new\_str + ' ' + s[len(s) - 1].capitalize()  
print(new\_str)

Output:

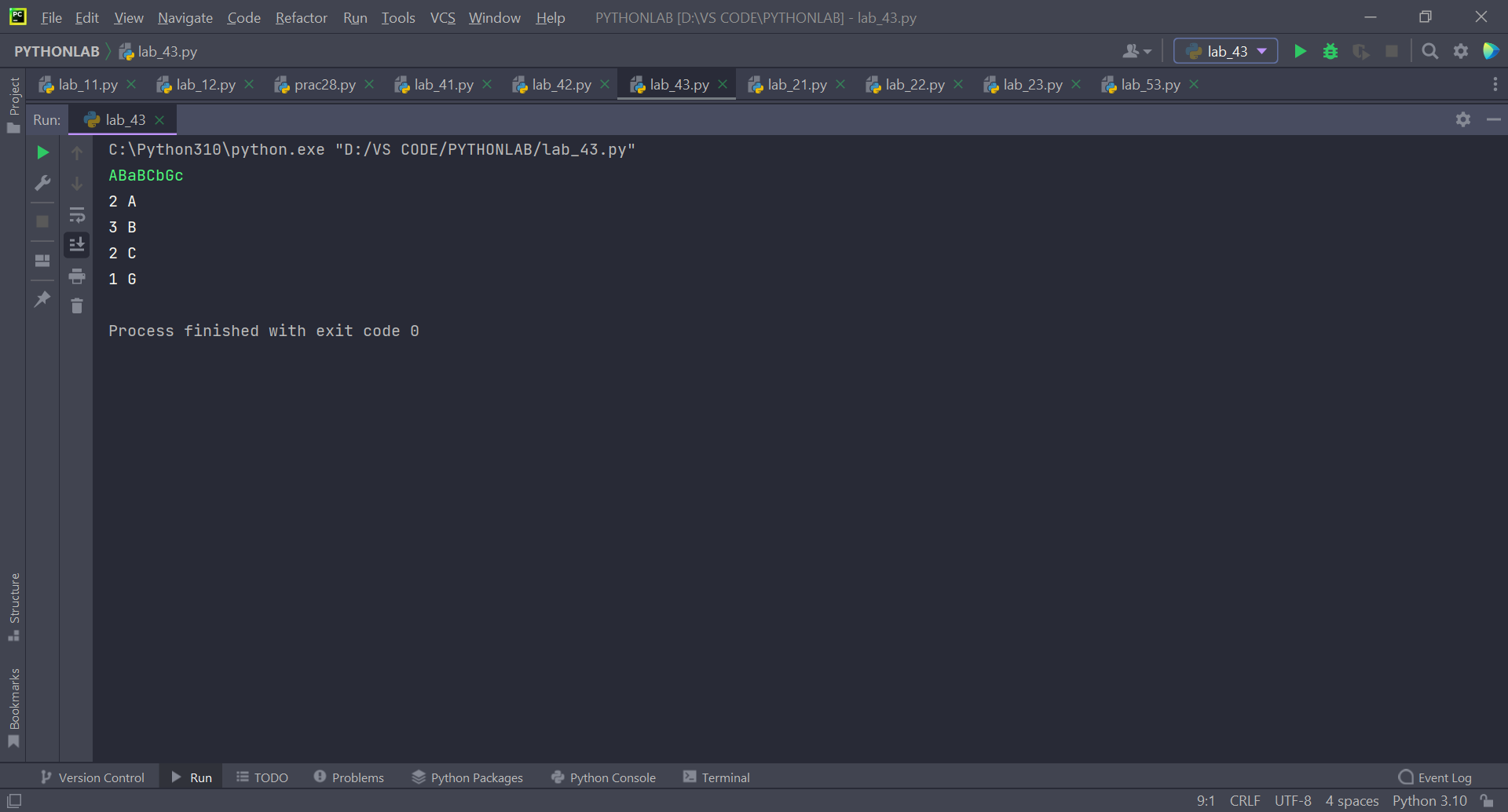


Solution Prog 3:

Code:

str1 = input()  
str1 = str1.upper()  
individual = set(str1)  
individual = sorted(individual)  
for i in individual:  
 c = str1.count(i)  
 print("%d %c" % (c, i))

Output:



**EXPERIMENT -5&6: To understand the concept of functions in python.**

**Objective: To understand the concept of Functions and Looping with Recursion.**

List of Lab Activities:

Q1. Using functions, re-write and execute Python program to:

a). Add natural numbers upto n where n is taken as an input from user.

b). Print Fibonacci series till nth term (Take input from user).

Q2. At an airport, a traveler is allowed entry into the flight only if he clears the following checks:

1. Baggage Check

2. Immigration Check

3. Security Check

The logic for the check methods is given below:

check\_baggage (baggage\_weight)

 returns True if baggage\_weight is greater than or equal to 0 and less than or equal to 40. Otherwise returns False.

check\_immigration (expiry\_year)

 returns True if expiry\_year is greater than or equal to 2030 and less than or equal to 2050. Otherwise returns False.

check\_security(noc\_status)

 returns True if noc\_status is 'valid' or 'VALID', for all other values return False.

traveler()

 Initialize the traveler Id and traveler name and invoke the functions check\_baggage(), check\_immigration() and check\_security() by passing required arguments.

 If all values of check\_baggage(), check\_immigration() and check\_security() are true,

- display traveler\_id and traveler\_name

- display "Allow Traveler to fly!"

Otherwise,

- display traveler\_id and traveler\_name

- display "Detain Traveler for Re-checking!

Invoke the traveler() function. Modify the values of different variables in traveler() function and observe the output.

Refer the table below for values of arguments.



Q3. Write a Python program to find the maximum and minimum values in a given list of tuples using lambda function.

Original list with tuples: [('V', 62), ('VI', 68), ('VII', 72), ('VIII', 70), ('IX', 74), ('X', 65)] **Output-**

Maximum and minimum values of the said list of tuples: (74, 62)

Solution:

Function: Functions are a block of code or statements, which helps us to break down our code into smaller units making it easier to code and to understand. Functions are only executed when they are called.

We can pass arguments and can have return value form a function.

Implementation: def <function\_name>(<argument/s>):

<statements>

return <variable> (optional)

Recursion: When a function calls itself over and over again is called Recursion. This process happens till the control reaches a base condition.

Lambda Function: These are unspecified functions which can have many arguments but a single expression.

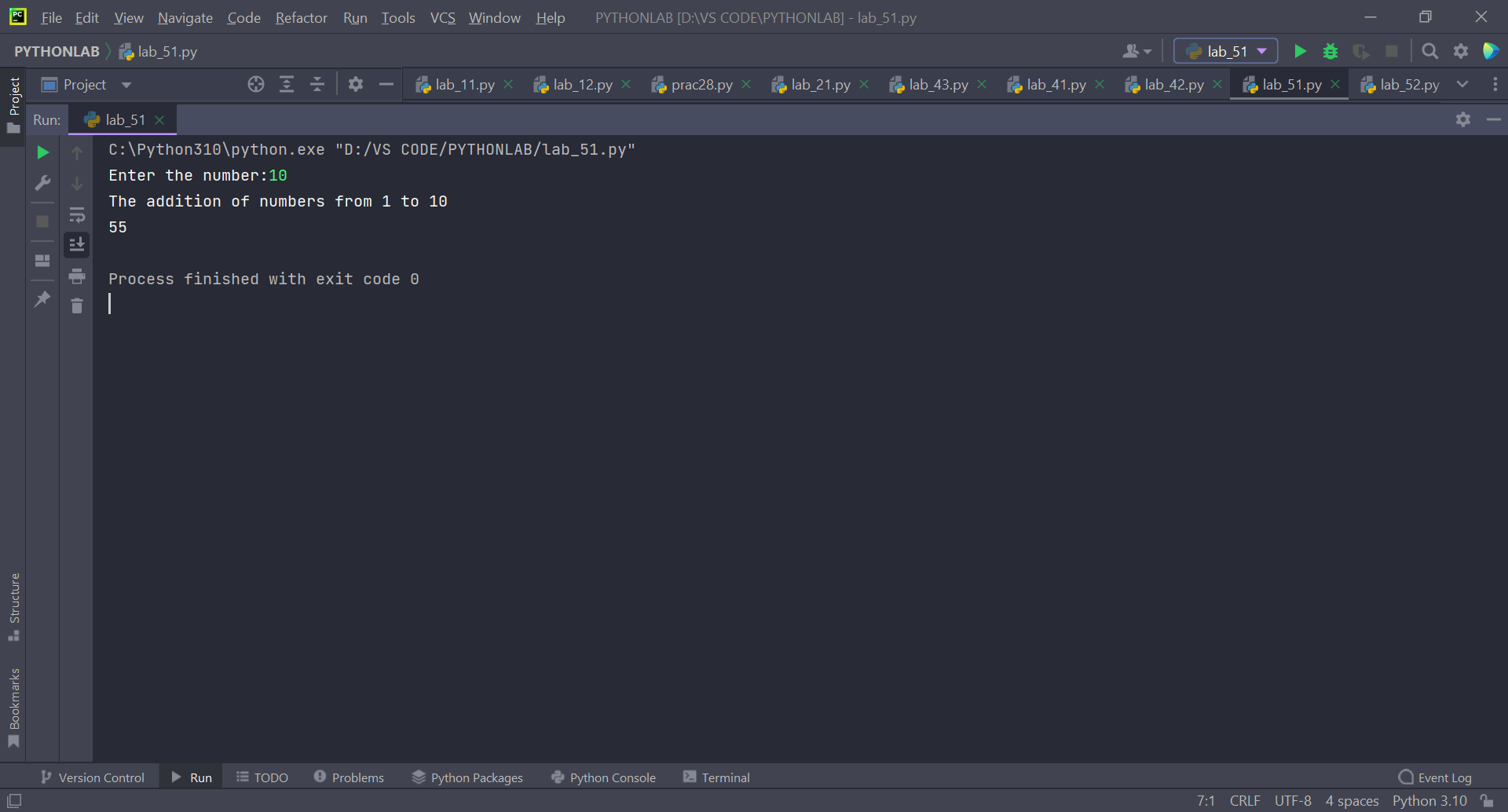
Syntax: lambda<argument/s>: <expression>

Solution Prog 1 a):

Code:

def add(*x*):  
 if *x* == 0:  
 return 0  
 return *x* + add(*x* - 1)  
  
  
n = int(input("Enter the number:"))  
print("The addition of numbers from 1 to %d" % n)  
print(add(n))

Output for 1 a):

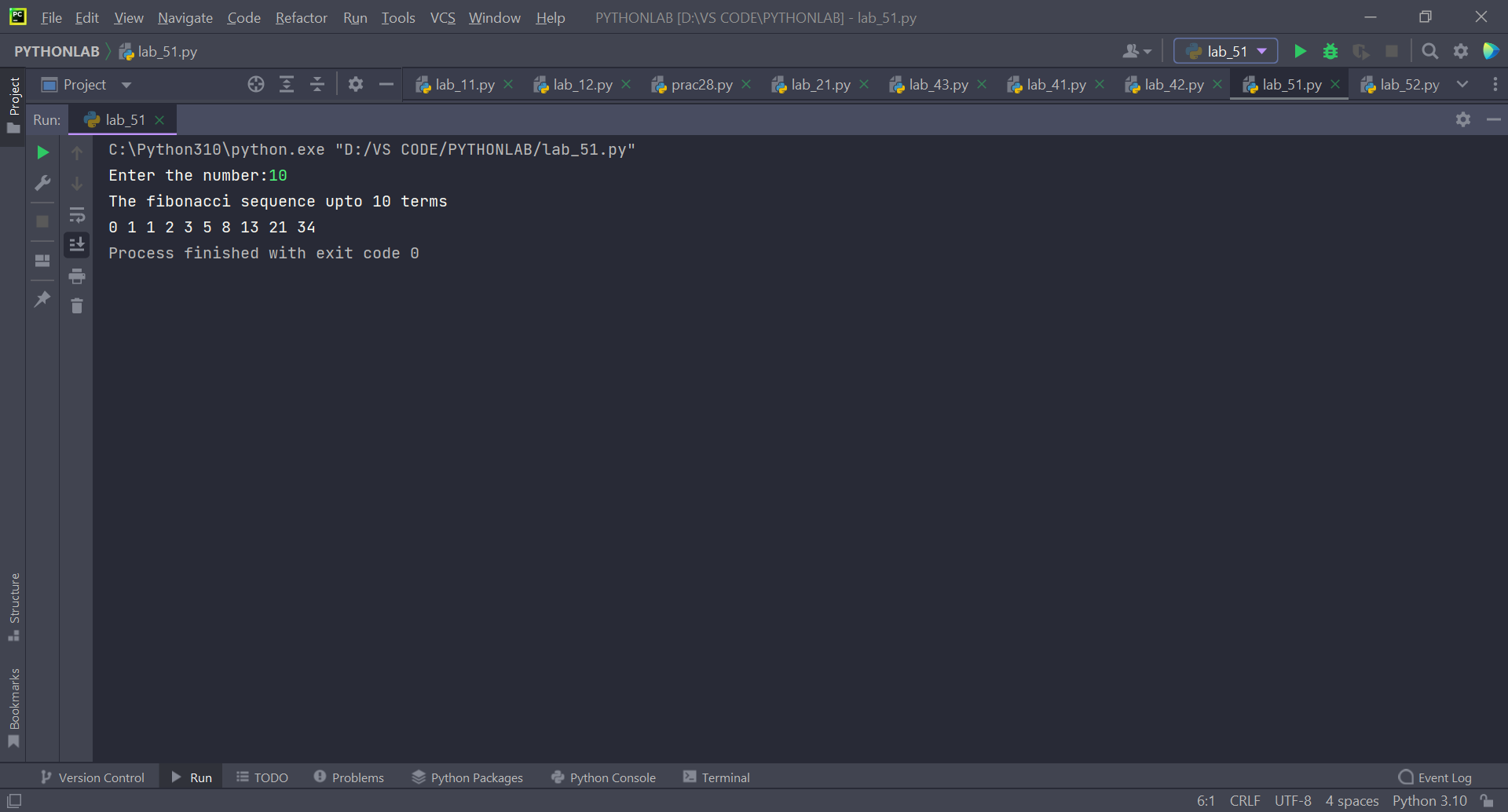


Solution Prog 1 b):

Code:

def fibo(*x*):  
 b = 1  
 c = 0  
 for i in range(1, *x* + 1):  
 print(c, end=" ")  
 i += 1  
 a = b  
 b = c  
 c = a + b  
  
  
n = int(input("Enter the number:"))  
print("The fibonacci sequence upto %d terms" % n)  
fibo(n)

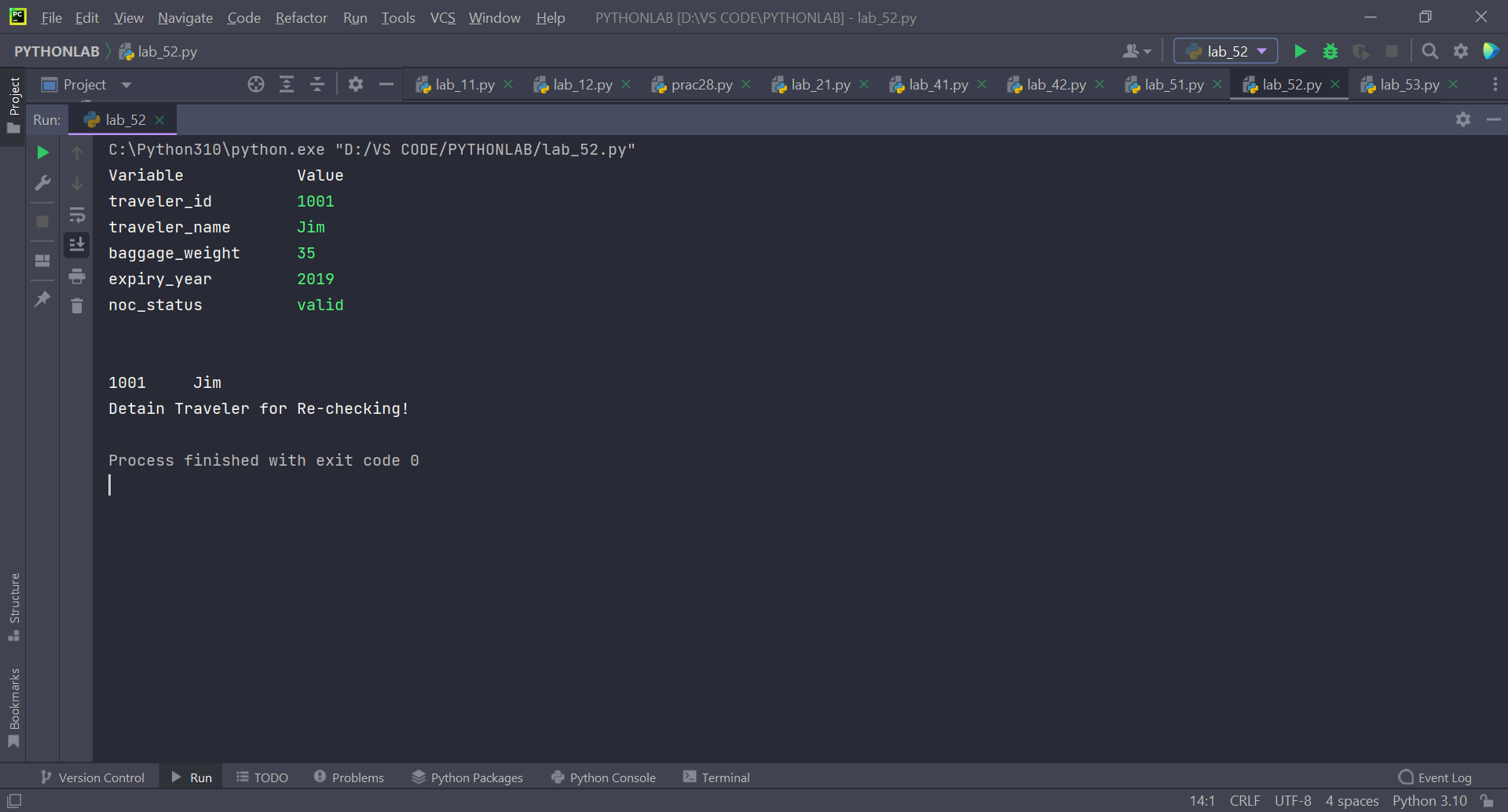
Output for 1 b):



Solution Prog 2:

Code:

def check\_baggage(*baggage\_weight*):  
 if 0 <= *baggage\_weight* <= 40:  
 return True  
 return False  
  
  
def check\_immigration(*expiry\_date*):  
 if 2030 <= *expiry\_date* <= 2050:  
 return True  
 return False  
  
  
def check\_security(*noc\_status*):  
 str1 = "valid"  
 str2 = "VALID"  
 if (*noc\_status* == str1) or (*noc\_status* == str2):  
 return True  
 return False  
  
  
def traveler():  
 print("Variable \t Value")  
 traveler\_id = int(input("traveler\_id \t"))  
 traveler\_name = input("traveler\_name \t")  
 bag = int(input("baggage\_weight \t"))  
 exp = int(input("expiry\_year \t"))  
 noc = input("noc\_status \t")  
 print("\n")  
 if (check\_baggage(bag) is True) and (check\_immigration(exp) is True) and (check\_security(noc) is True):  
 print(traveler\_id, "\t", traveler\_name)  
 print("Allow Traveler to fly!")  
 else:  
 print(traveler\_id, "\t", traveler\_name)  
 print("Detain Traveler for Re-checking!")  
  
  
traveler()

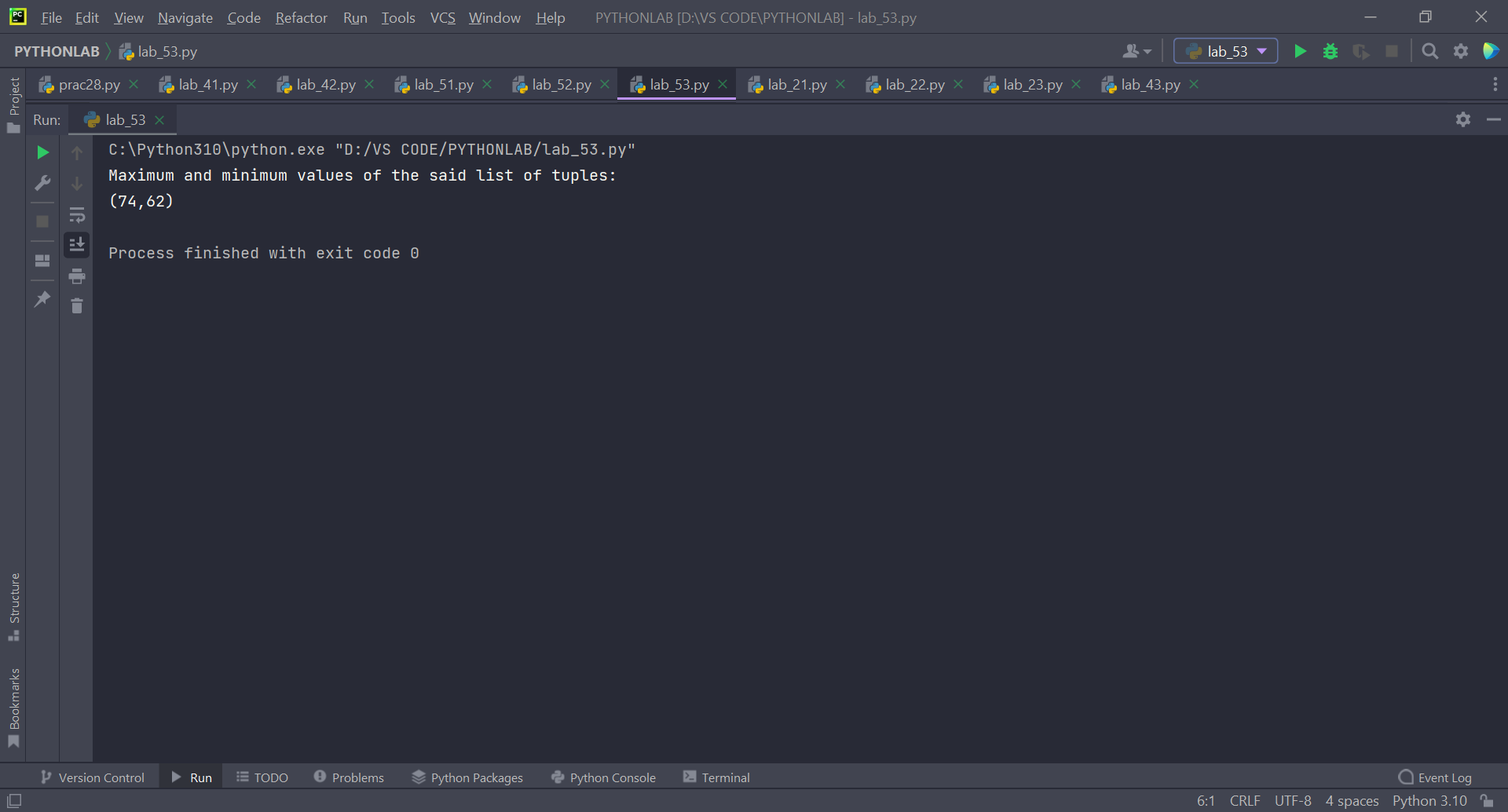
Output:

Solution Prog 3:

Code:

data = [('V', 62), ('VI', 68), ('VII', 72), ('VIII', 70), ('IX', 74), ('X', 65)]  
maximum = max(data, key=lambda a: a[1])[1]  
minimum = min(data, key=lambda b: b[1])[1]  
print("Maximum and minimum values of the said list of tuples:")  
print("(%d,%d)" % (maximum, minimum))

Output:



**Experiment: 7 & 8**

**File Handling in python**

**Objective: To understand the concept of Files, and Operations on Files.**

**File Handling:**

Persistent data is information that is kept in files. It signifies that it is eternal in nature. Python allows us to read and store data to external text files on secondary storage media indefinitely.

We must first open a file before we can begin working with it. It must be closed after performing the desired activity in order to free up resources that are associated to the file.

The process of handling data by software, which includes IO activities, is known as file handling. These files are kept on a hard drive in a directory. Any time an operation is performed, the file is opened and made available for writing and reading in RAM.

**Why to use file handling in python:**

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory.

All projects need the contribution to process and result to show information. Furthermore, everything needs a document as name stockpiling compartments on PCs that are overseen by OS. However, factors give us a method for putting away information while the program runs, assuming we need out information to continue even after the end of the program, we need to save it to a file.

List of Lab Activities:

Q1. Write a Python program to:

1. read a file.

2. add backslash (\) before every double quote in the file contents.

3. write it to another file in the same folder.

4. print the contents of both the files.

**For example:**

If the first file is 'TestFile1.txt' with text as:

Jack said, "Hello Pune".

The output of the file 'TestFile2.txt' should be:

Jack said,\"Hello Pune\".

Q2. Consider a file 'rhyme.txt' in D Drive with following text:



Write a Python program to count the words in the file using a dictionary (use space as a delimiter). Find unique words and the count of their occurrences (ignoring case). Write the output in another file "words.txt" at the same location.

Q3. Assume a file city.txt with details of 5 cities in given format (cityname population(in lakhs) area(in sq KM) ):

Example:

Dehradun 5.78 308.20

Delhi 190 1484

……………

Open file city.txt and read to:

a. Display details of all cities

b. Display city names with population more than 10Lakhs

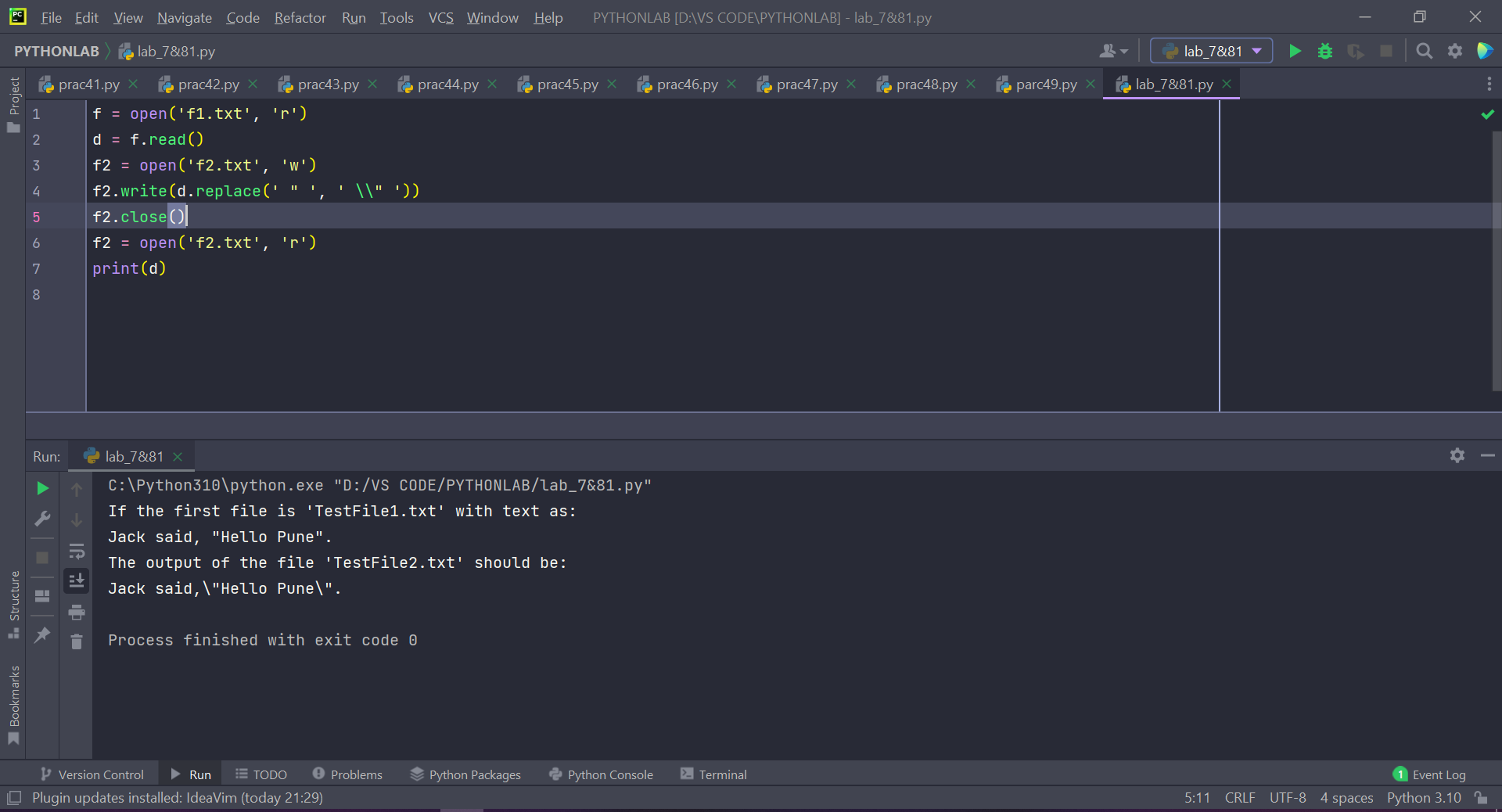
c. Display sum of areas of all cities

**Solution to Prog 1:**

**Code:**

f = open('f1.txt', 'r')  
d = f.read()  
f2 = open('f2.txt', 'w')  
f2.write(d.replace(' " ', ' \\" '))  
f2.close()  
f2 = open('f2.txt', 'r')  
print(d)

**Output:**

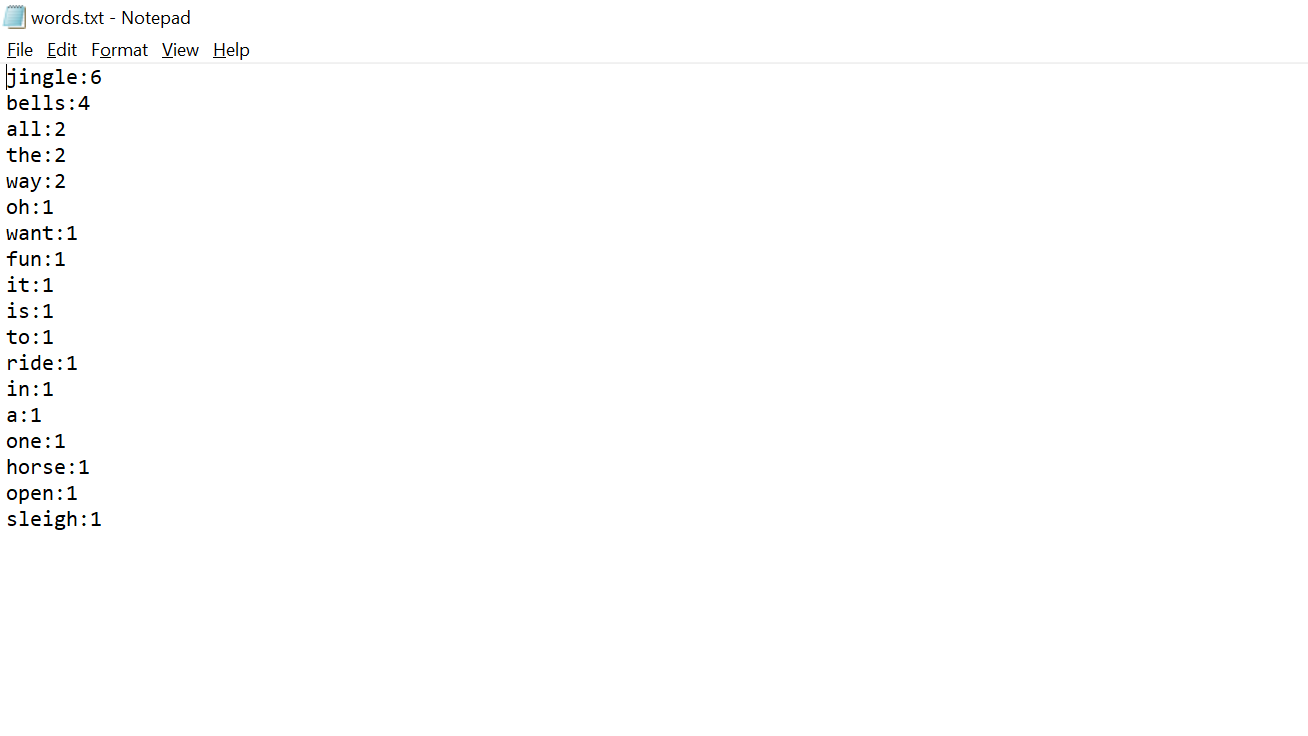
****

**Solution for Prog 2:**

**Code:**

f = open('jingle.txt', 'r+')  
l = f.read().lower()  
l = l.split()  
cont = {}  
for i in l:  
 if i not in cont:  
 cont[i] = 1  
 else:  
 cont[i] += 1  
  
  
f2 = open('words.txt', 'w+')  
for key, value in cont.items():  
 f2.write('%s:%s\n' % (key, value))

**Output:**

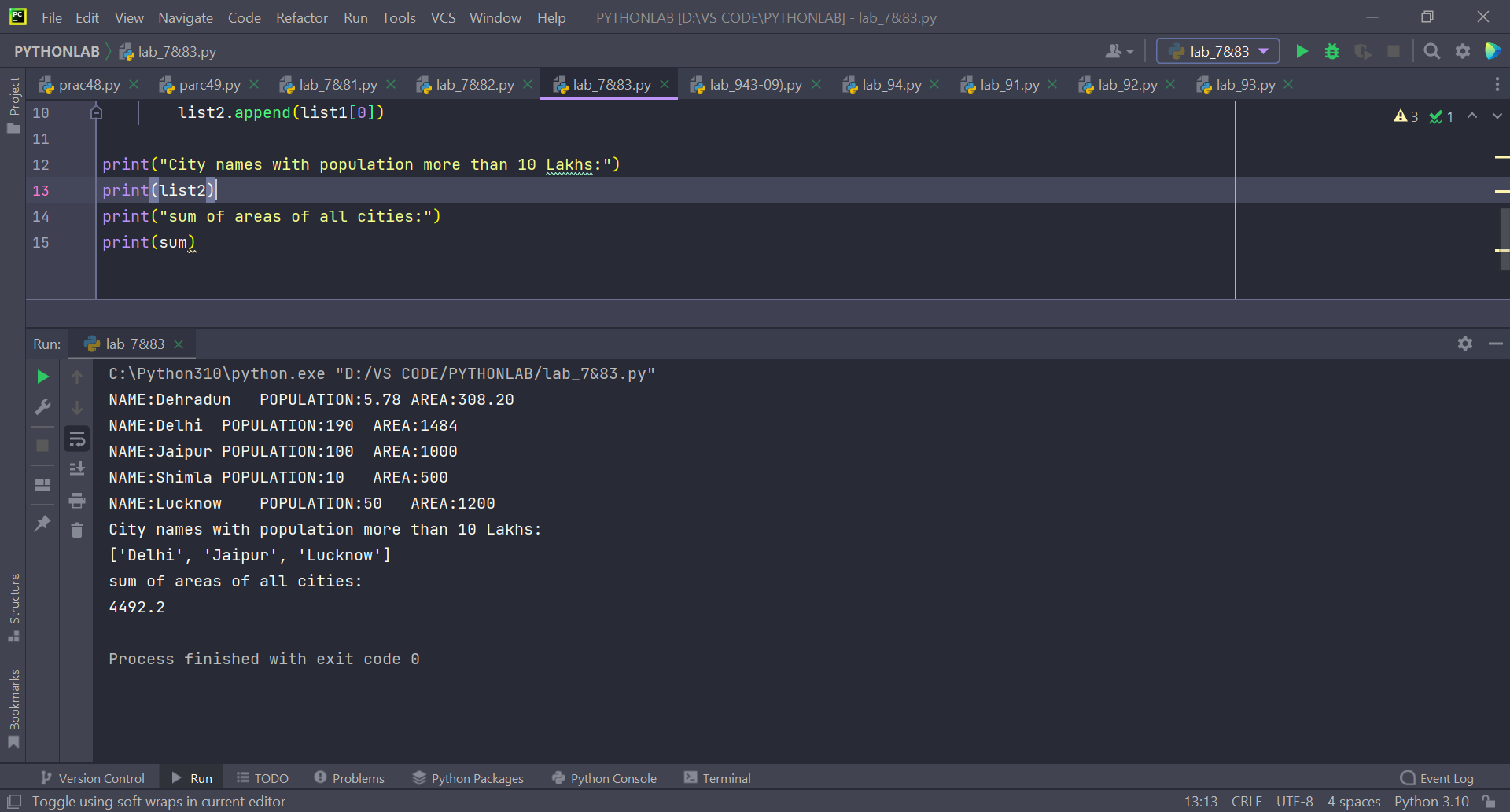
****

**Solution for prog 3:**

**Code:**

f = open("city.txt", "r+")  
y = f.readlines()  
list2 = []  
sum = 0  
for i in y:  
 list1 = i.split()  
 print('NAME:{}\tPOPULATION:{}\tAREA:{}'.format(list1[0], list1[1], list1[2]))  
 sum = sum + float(list1[2])  
 if float(list1[1]) > 10:  
 list2.append(list1[0])  
  
print("City names with population more than 10 Lakhs:")  
print(list2)  
print("sum of areas of all cities:")  
print(sum)

**Output:**

****

**Experiment: 9**

**Exception Handling**

**Objective: To understand the concept of Exceptions and Exception handling mechanism in python.**

**Exception Handling:**

A program's exception can be characterized as an unexpected condition that causes the program's flow to be disrupted.

When an exception occurs, the program's execution is halted, and the remaining code is not executed. As a result, run-time failures that are unable to manage the Python script are an exception. A Python object that describes an error is called an exception.

Python provides a mechanism for handling exceptions so that the code can continue to run uninterrupted. The interpreter does not execute all of the code that exists after the exception if we do not manage it.  Python includes a number of built-in exceptions that allow our application to execute uninterrupted and produce the desired results

**Some import exception types:**

Some of the common built-in exceptions in Python programming along with the error that cause them are listed below:

Exception Cause of Error

AssertionError Raised when an assert statement fails.

AttributeError Raised when attribute assignment or reference fails.

EOFError Raised when the input() function hits end-of-file condition.

FloatingPointError Raised when a floating point operation fails.

GeneratorExit Raise when a generator's close() method is called.

ImportError Raised when the imported module is not found.

IndexError Raised when the index of a sequence is out of range.

**Explain try, except and finally :**

Python Exception Handling Mechanism

Exception handling is managed by the following 5 keywords:

1. try

2. except

3. finally

Python Try Statement

A try statement includes keyword try, followed by a colon (:) and a suite of code in which exceptions may occur. It has one or more clauses.

During the execution of the try statement, if no exceptions occurred then, the interpreter ignores the exception handlers for that specific try statement.

In case, if any exception occurs in a try suite, the try suite expires and program control transfers to the matching except handler following the try suite.

Syntax:

try:

statement(s)

Python Except Block

If the exception does occur, the program flow is transferred to the except: block. The statements in the except: block are meant to handle the cause of the exception appropriately.

You can mention a specific type of exception in front of the except keyword.

The subsequent block will be executed only if the specified exception occurs.

There may be multiple except clauses with different exception types in a single try block.

If the type of exception doesn't match any of the except blocks, it will remain unhandled and the program will terminate.

Syntax:

except:

# optional block

# Handling of exception (if required)

Finally Keyword in Python

Python provides a keyword finally, which is always executed after the try and except blocks. The final block always executes after normal termination of try block or after try block terminates due to some exception.

Syntax:

try:

# Some Code....

except:

# optional block

# Handling of exception (if required)

finally:

# Some code .....(always executed)

Q1. Input two values from user where the first line contains N, the number of test cases.

The next N lines contain the space separated values of a and b. Perform integer division and print a/b. Print the error code in the case of *ZeroDivisionError* or *ValueError*.

**Sample input**

1 0

2 $

3 1

**Sample Output**

Error Code: integer division or modulo by zero

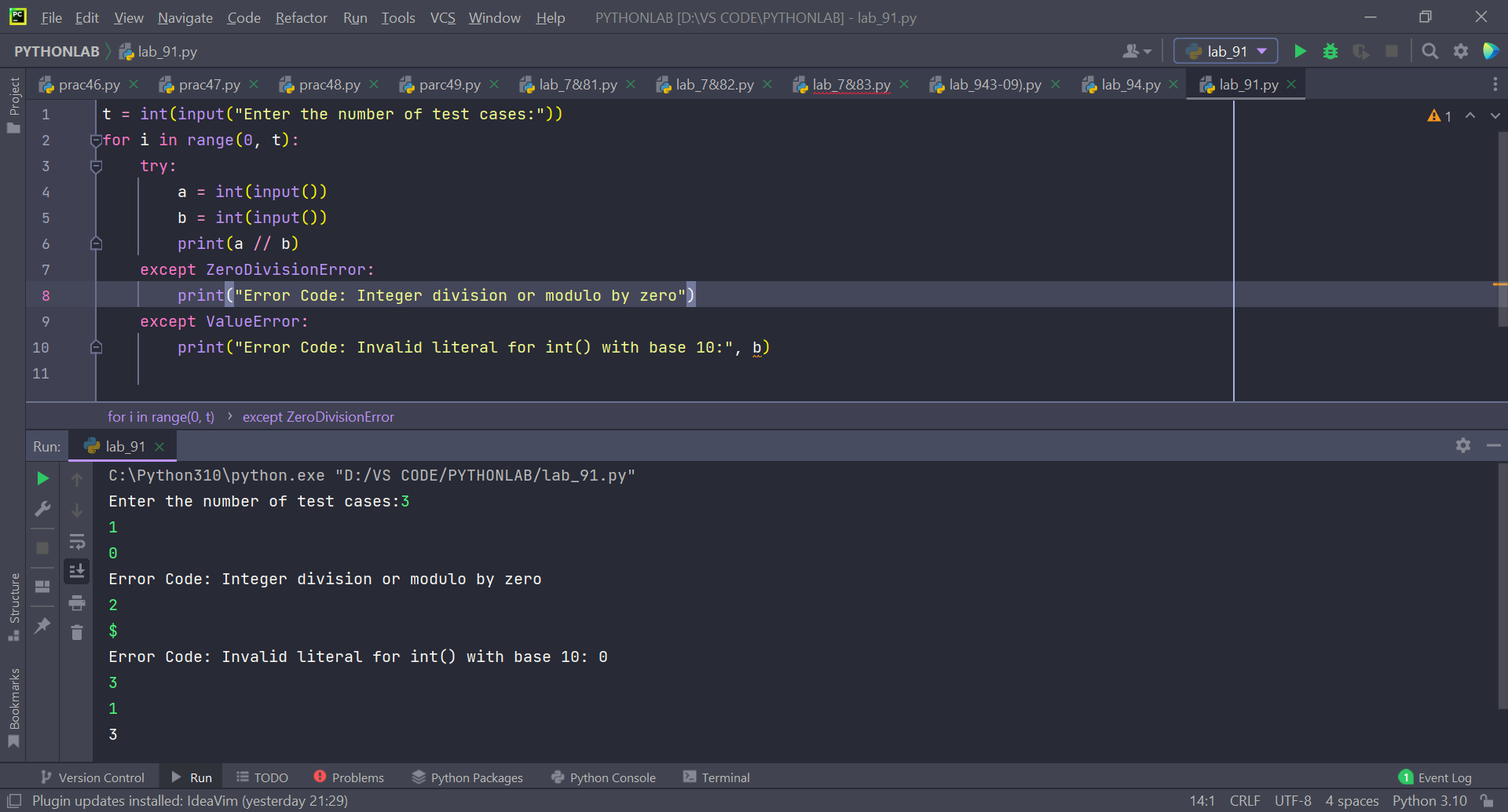
Error Code: invalid literal for int() with base 10: '$'

3

**Code:**

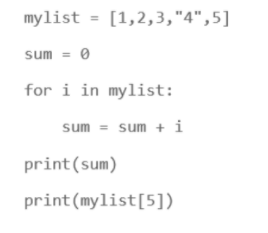
t = int(input("Enter the number of test cases:"))  
for i in range(0, t):  
 try:  
 a = int(input())  
 b = int(input())  
 print(a // b)  
 except ZeroDivisionError:  
 print("Error Code: Integer division or modulo by zero")  
 except ValueError:  
 print("Error Code: Invalid literal for int() with base 10:", b)

**Output:**



Q2. Assume the following Python code-

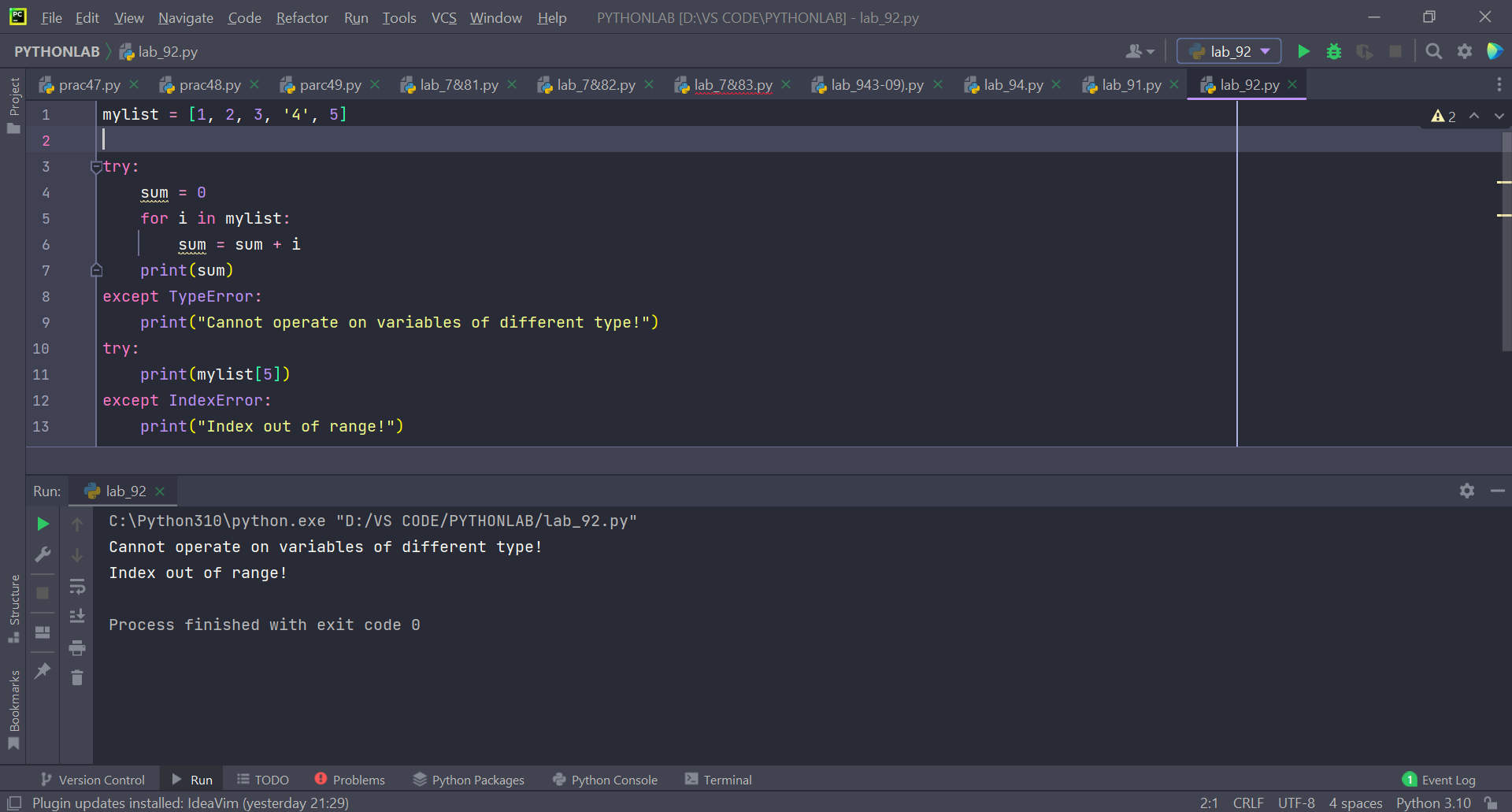
Rewrite the code to handle the exceptions raised. Print appropriate error messages wherever applicable.



**Code:**

mylist = [1, 2, 3, '4', 5]  
  
try:  
 sum = 0  
 for i in mylist:  
 sum = sum + i  
 print(sum)  
except TypeError:  
 print("Cannot operate on variables of different type!")  
try:  
 print(mylist[5])  
except IndexError:  
 print("Index out of range!")

**Output:**



Q3. You have already created a Python program to implement the following in ﬁle handling section:

1. read a ﬁle.

2. add backslash (\) before every double quote in the ﬁle contents.

3. write it to another ﬁle in the same folder.

4. print the contents of both the ﬁles.

For example:

If the ﬁrst ﬁle is 'TestFile1.txt' with text as:

Jack said, "Hello Pune".

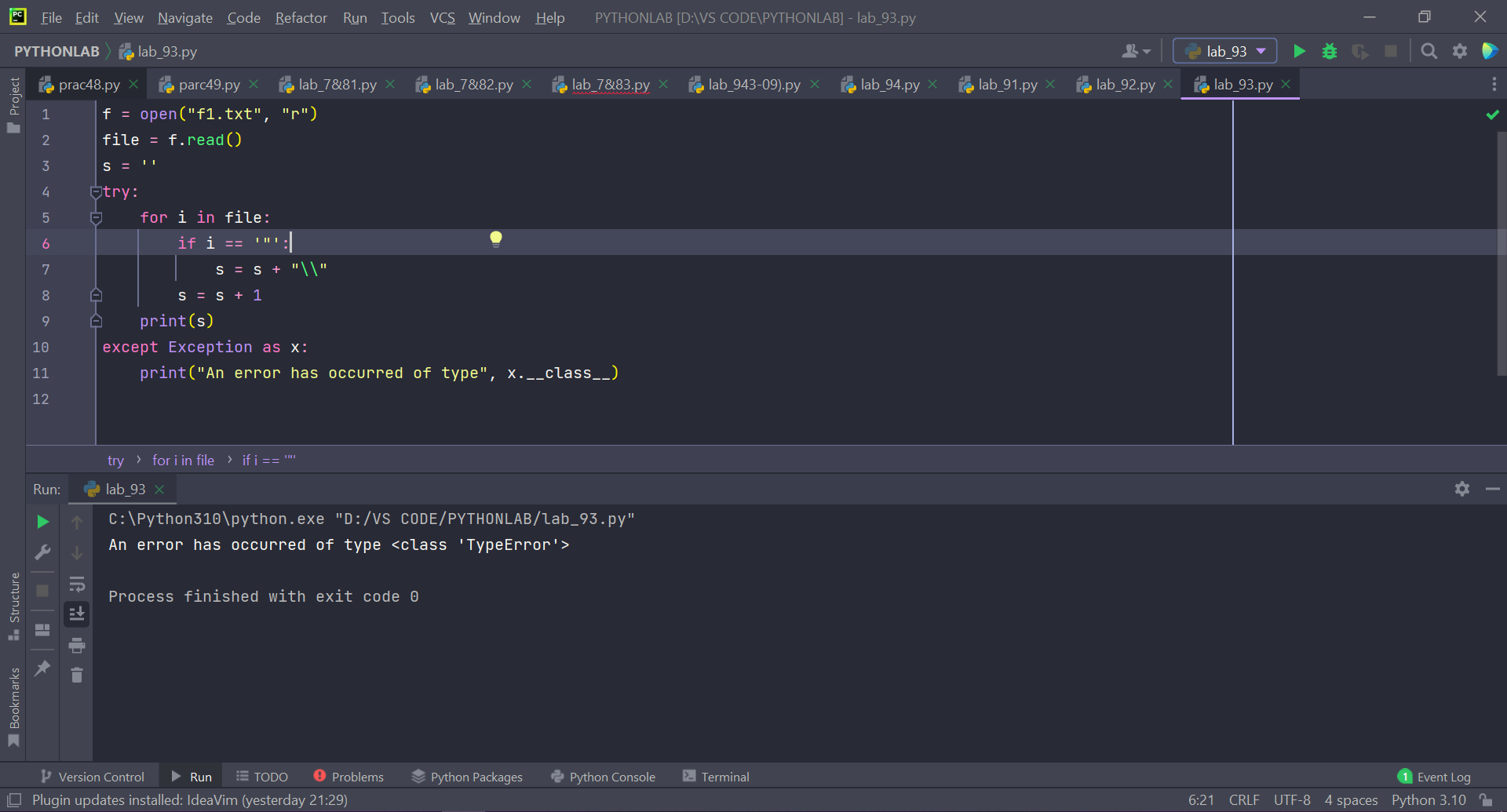
The output of the ﬁle 'TestFile2.txt' should be:

Jack said,\"Hello Pune\".

Modify your code to implement Exception handling. Print appropriate error messages wherever applicable.

**Code:**

f = open("f1.txt", "r")  
file = f.read()  
s = ''  
try:  
 for i in file:  
 if i == '"':  
 s = s + "\\"  
 s = s + 1  
 print(s)  
except Exception as x:  
 print("An error has occurred of type", x.\_\_class\_\_)

**Output:**

**EXPERIMENT-10&11: Object Oriented Programming**

**Objective: To understand the concept of OOPS like polymorphism, inheritance, abstraction and applying them in solving the problems.**

Object and Classes:

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

Class creates a user-defined data structure, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Some points on Python class:

Classes are created by keyword class.

Attributes are the variables that belong

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It’s not an idea anymore, it’s an actual dog, like a dog of breed pug who’s seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.  
An object consists of :

State: It is represented by the attributes of an object. It also reflects the properties of an object.

Behavior: It is represented by the methods of an object. It also reflects the response of an object to other objects.

Identity: It gives a unique name to an object and enables one object to interact with other objects.

List of Lab Activities:

Question 1:

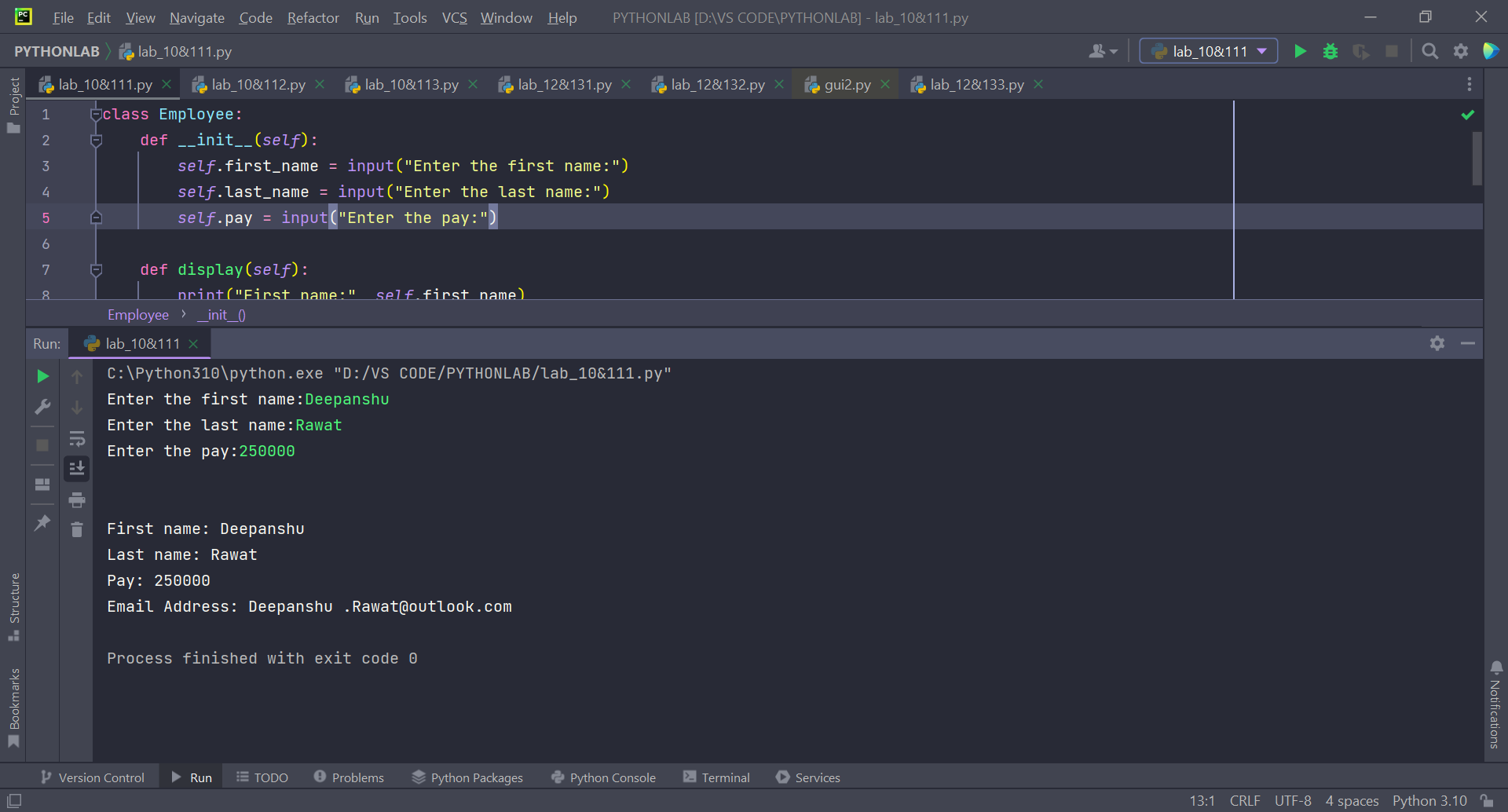


Solution:

Code:

class Employee:  
 def \_\_int\_\_(*self*):  
 *self*.first\_name = input("Enter the first name:")  
 *self*.last\_name = input("Enter the last name:")  
 *self*.pay = input("Enter the pay:")  
  
 def display(*self*):  
 print("First name:", *self*.first\_name)  
 print("Last name:", *self*.last\_name)  
 print("Pay:", *self*.pay)  
 s = *self*.first\_name + "." + *self*.last\_name + "@outlook.com"  
 print("Email Address:", s)  
  
  
obj = Employee()  
print("\n")  
obj.display()

Output:



Perform the following instructions:

a) Create a Vehicle class with max\_speed and mileage as instance attributes. Additionally, create a method named seating\_capacity() using the below syntax:

def seating\_capacity(self, capacity):

return f"The seating capacity of a {self.name} is {capacity} passengers"

b) Create child class ‘Bus’ that will inherit all of the variables and methods of the Vehicle class. Set the seating capacity of the bus to 50 using super().

c) Create a Bus object that will inherit all of the variables and methods of the Vehicle class and display it.

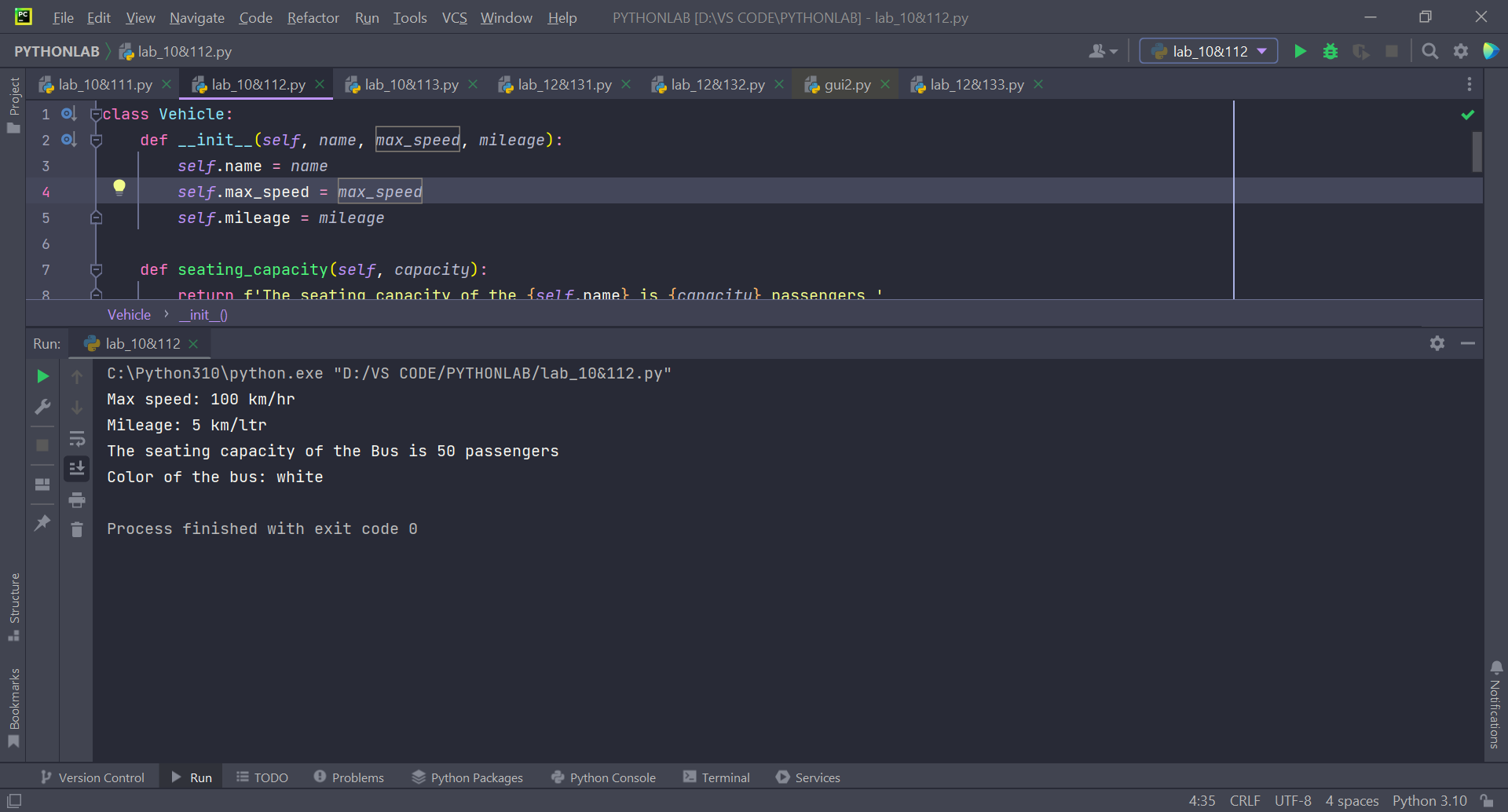
d) Define a class attribute “color” with a default value white. I.e., Every Vehicle should be white.

Solution:

Code:

class Vehicle:  
 def \_\_init\_\_(*self*, *name*, *max\_speed*, *mileage*):  
 *self*.name = *name  
 self*.max\_speed = *max\_speed  
 self*.mileage = *mileage* def seating\_capacity(*self*, *capacity*):  
 return f'The seating capacity of the {*self*.name} is {*capacity*} passengers '  
  
  
class Bus(Vehicle):  
 color = "white"  
  
 def \_\_init\_\_(*self*):  
 super().\_\_init\_\_("Bus", "100 km/hr", "5 km/ltr")  
  
 def display(*self*):  
 print("Max speed:", *self*.max\_speed)  
 print("Mileage:", *self*.mileage)  
 print(super().seating\_capacity(50))  
 print("Color of the bus:", *self*.color)  
  
  
obj = Bus()  
obj.display()

Output:



Question 3:



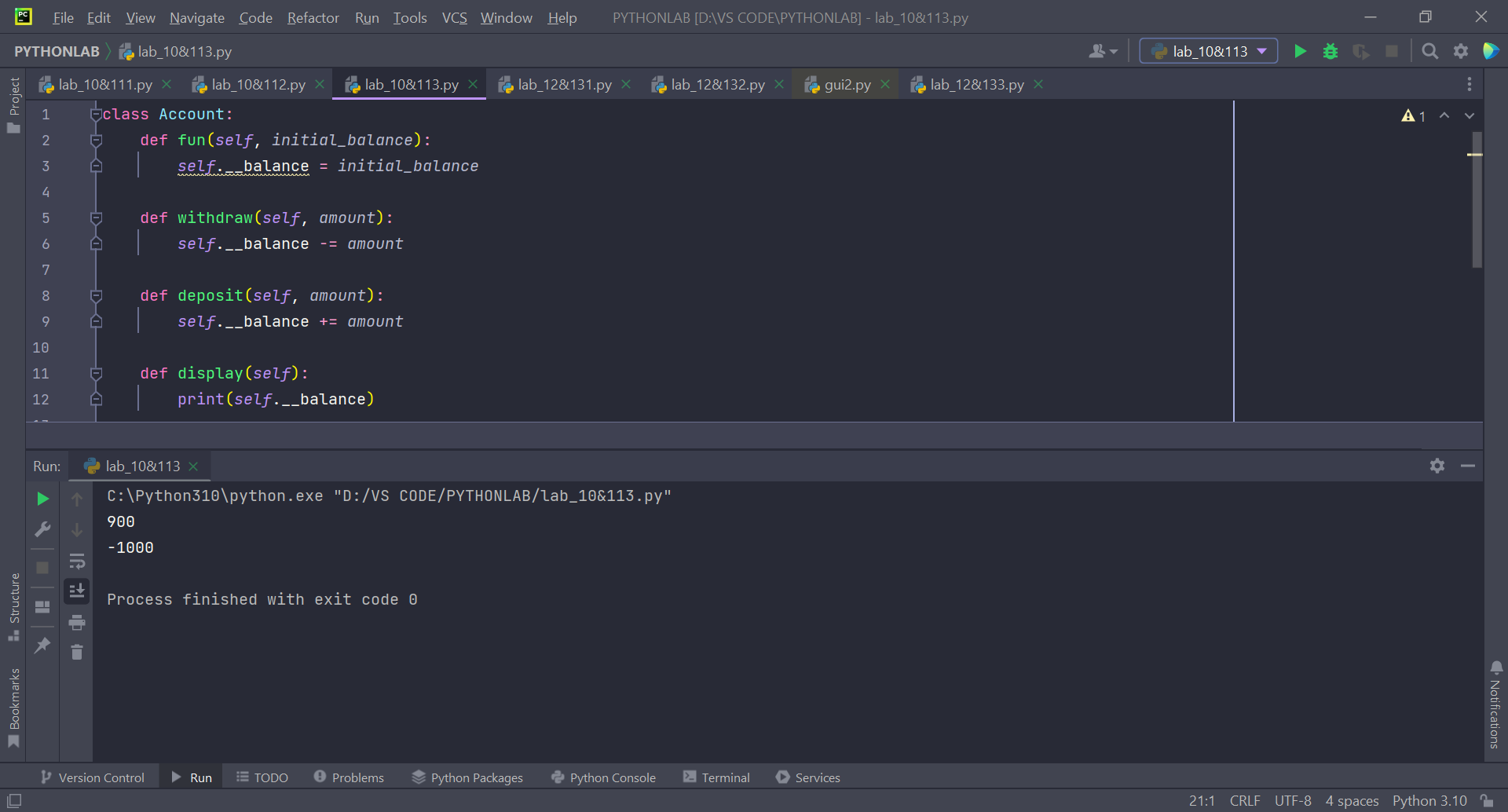


Solution:

Code:

class Account:  
 def fun(*self*, *initial\_balance*):  
 *self*.\_\_balance = *initial\_balance* def withdraw(*self*, *amount*):  
 *self*.\_\_balance -= *amount* def deposit(*self*, *amount*):  
 *self*.\_\_balance += *amount* def display(*self*):  
 print(*self*.\_\_balance)  
  
  
ac = Account()  
ac.fun(1000)  
ac.withdraw(100)  
ac.display()  
ac.balance = -1000  
print(ac.balance)

Output:



**EXPERIMENT-12&13: Tkinter module**

**Objective: Creating GUI applications in python using Tkinter module**

**Tkinter Module:**

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

**mainloop():** There is a method known by the name mainloop() is used when your application is ready to run. mainloop() is an infinite loop used to run the application, wait for an event to occur and process the event as long as the window is not closed.

Syntax:

From tkinter import \*

SUDO=Tk()

SUDO.mainloop()

List of Lab Activities:

Q1. a) import Tkinter package and create a window and set its title

b) set the default window size using geometry function

c) Create a label with “Hello” text in it and set its position on the form.

d) Add a button to the window with “CLICK ME” written on it.

e) change the foreground and background color for the button created above

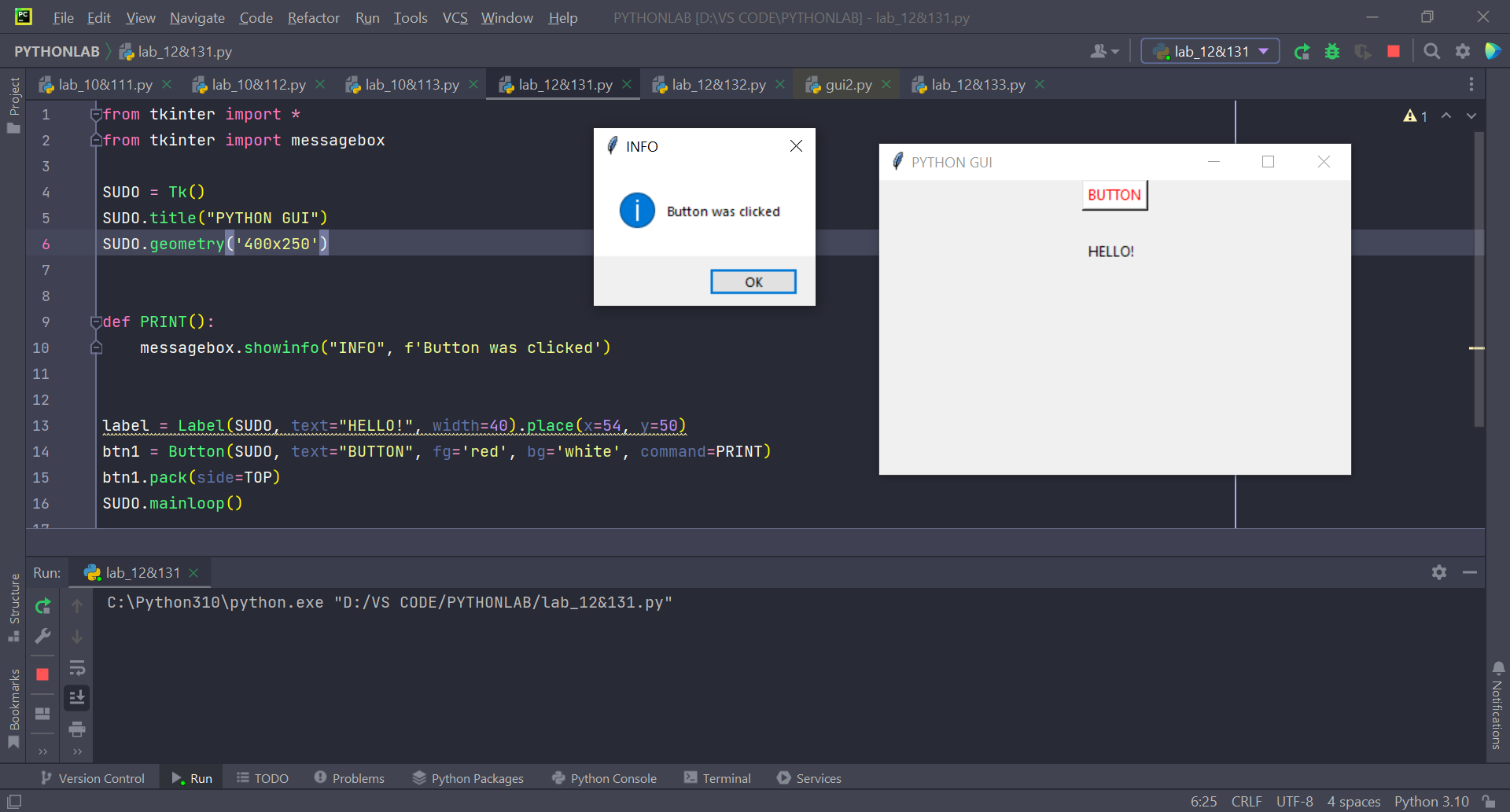
f) Create a function that will be executed when the button is clicked and print “Button was clicked” on clicking the button

Solution:

Code:

from tkinter import \*  
from tkinter import messagebox  
  
SUDO = Tk()  
SUDO.title("PYTHON GUI")  
SUDO.geometry('400x250')  
  
  
def PRINT():  
 messagebox.showinfo("INFO", f'Button was clicked')  
  
  
label = Label(SUDO, text="HELLO!", width=40).place(x=54, y=50)  
btn1 = Button(SUDO, text="BUTTON", fg='red', bg='white', command=PRINT)  
btn1.pack(side=TOP)  
SUDO.mainloop()

Output:



Q2. This is the continuation of Question1, add the given below features in the above program:

a) Take **user name** as input using the Tkinter Entry class

b) Print the entered text (username) on clicking the button.

c) Create three RadioButtons as displayed below



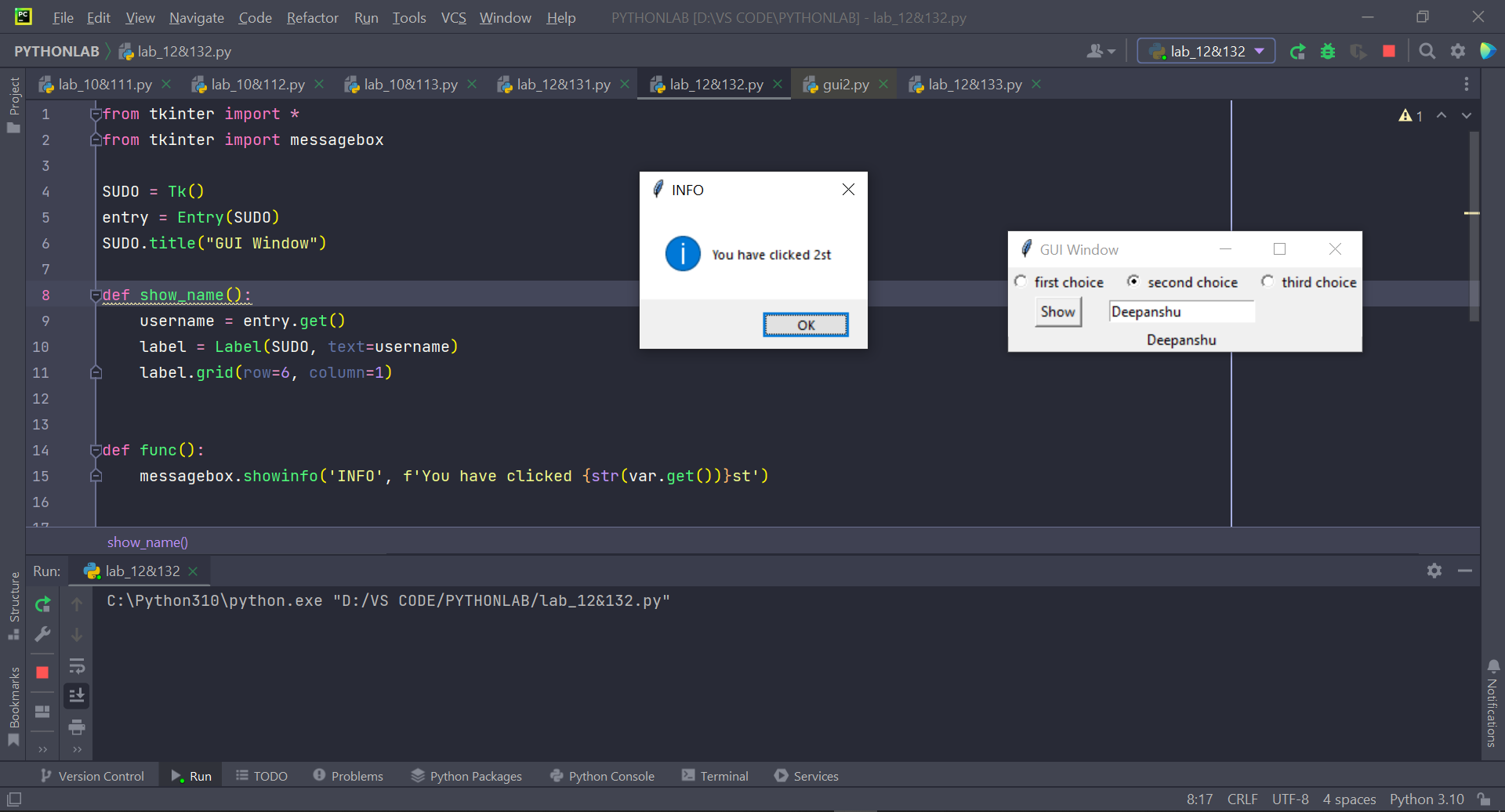
d) Print the currently selected radio button or the radio button value.

Solution:

Code:

from tkinter import \*  
from tkinter import messagebox  
  
SUDO = Tk()  
entry = Entry(SUDO)  
  
  
def show\_name():  
 username = entry.get()  
 label = Label(SUDO, text=username)  
 label.grid(row=6, column=1)  
  
  
def func():  
 messagebox.showinfo('INFO', f'You have clicked {str(var.get())}st')  
  
  
var = IntVar()  
RB1 = Radiobutton(SUDO, text="first choice", value=1, command=func, variable=var)  
RB1.grid(row=1, column=0)  
RB2 = Radiobutton(SUDO, text="second choice", value=2, command=func, variable=var)  
RB2.grid(row=1, column=1)  
RB3 = Radiobutton(SUDO, text="third choice", value=3, command=func, variable=var)  
RB3.grid(row=1, column=2)  
btn1 = Button(SUDO, text="Show", command=show\_name)  
btn1.grid()  
entry.grid(row=2, column=1)  
SUDO.mainloop()

Output:



Q3. Write a program to accept following details from a student using GUI

1. Name of the student (using Textbox)

2. Gender (Using radio button)

3. Qualification (Using List)

4. Marks of three subjects (using Textbox)

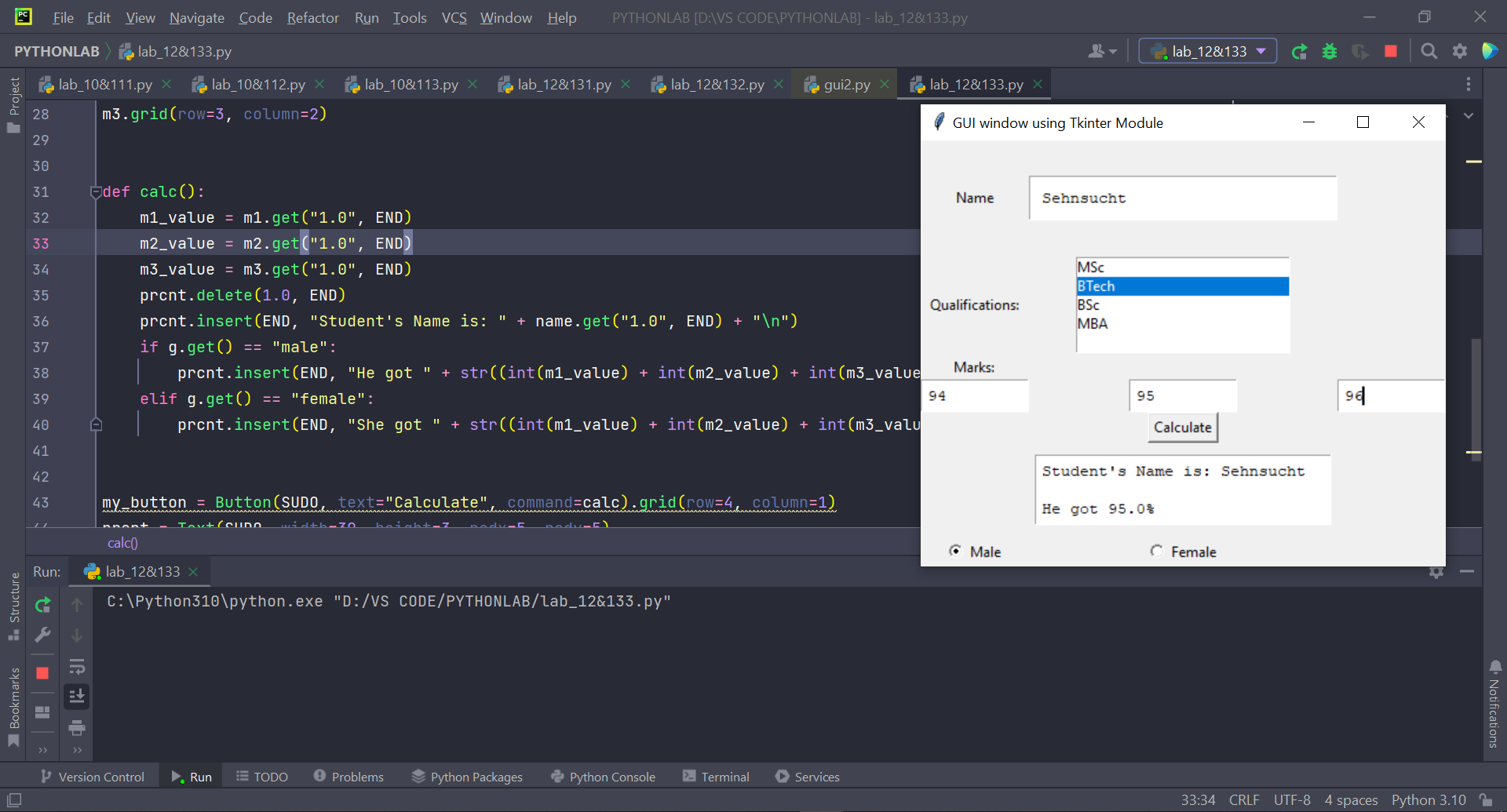
Compute the percentage of the student and display it in a textbox.

Solution:

Code:

from tkinter import \*  
  
SUDO = Tk()  
SUDO.title("Tkinter")  
label\_name = Label(SUDO, text="Name").grid(row=0, column=0)  
name = Text(SUDO, width=30, height=1, padx=10, pady=10)  
name.grid(row=0, column=1, pady=30)  
g = StringVar()  
m = Radiobutton(SUDO, text="Male", variable=g, value="male")  
f = Radiobutton(SUDO, text="Female", variable=g, value="female")  
m.grid(row=6, column=0)  
f.grid(row=6, column=1)  
label1 = Label(SUDO, text="Qualifications:")  
label1.grid(row=1, column=0)  
qualifications = Listbox(SUDO, width=30, height=5)  
qualifications.grid(row=1, column=1)  
qualifications.insert(1, "MSc")  
qualifications.insert(2, "BTech")  
qualifications.insert(3, "BSc")  
qualifications.insert(4, "MBA")  
marks\_Label = Label(SUDO, text="Marks:")  
marks\_Label.grid(row=2, column=0)  
m1 = Text(SUDO, width=10, height=1, padx=5, pady=5)  
m1.grid(row=3, column=0)  
m2 = Text(SUDO, width=10, height=1, padx=5, pady=5)  
m2.grid(row=3, column=1)  
m3 = Text(SUDO, width=10, height=1, padx=5, pady=5)  
m3.grid(row=3, column=2)  
  
  
def calc():  
 m1\_value = m1.get("1.0", END)  
 m2\_value = m2.get("1.0", END)  
 m3\_value = m3.get("1.0", END)  
 prcnt.delete(1.0, END)  
 prcnt.insert(END, "Student's Name is:" + name.get("1.0", END) + "\n")  
 if g.get() == "male":  
 prcnt.insert(END, "He got" + str((int(m1\_value) + int(m2\_value) + int(m3\_value)) / 3))  
 elif g.get() == "female":  
 prcnt.insert(END, "She got" + str((int(m1\_value) + int(m2\_value) + int(m3\_value)) / 3))  
  
  
my\_button = Button(SUDO, text="Calculate", command=calc).grid(row=4, column=1)  
prcnt = Text(SUDO, width=30, height=3, padx=5, pady=5)  
prcnt.grid(row=5, column=1, pady=10)  
SUDO.mainloop()

Output:



**EXPERIMENT-14: Numpy**

**Objective: Understand the usage of numpy module of python.**

**Numpy Module:**

**Numpy**is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python.Besides its obvious scientific uses, Numpy can also be used as an efficient multi-dimensional container of generic data.

**Numpy** Array: NumPy is used to work with arrays. The array object in NumPy is called ndarray.

We can create a NumPy ndarray object by using the array() function.

**Reshaping an array:** this is used to reshape the order of the matrix.

Syntax:

Array\_name.reshape(order of the matrix)

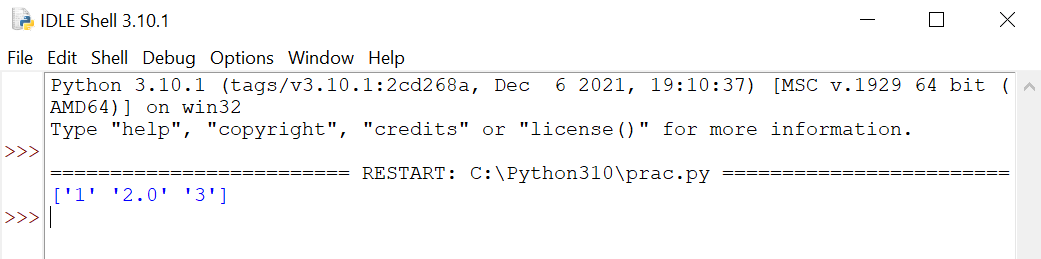
List of Lab Activities:

a. Convert numbers =[1, 2.0, 3] to numpy array and convert all elements to string type.

Code:

import numpy  
  
num = [1, 2.0, 3]  
a = numpy.asarray(num, dtype=str)  
print(a)

Output:

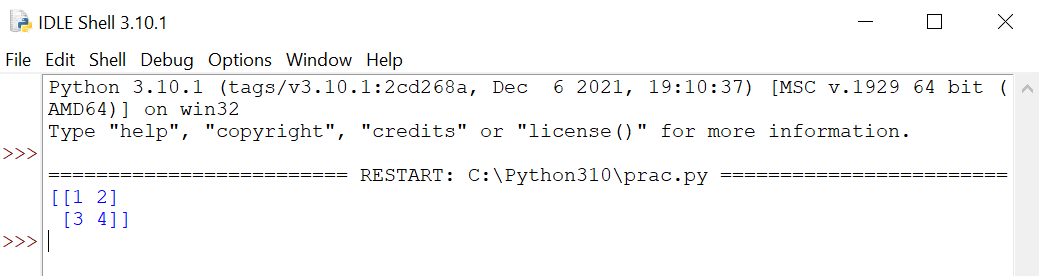


b. Create a 2 D array through list and set dtype as int32

Code:

import numpy as np  
  
lst = [1, 2, 3, 4]  
a = np.asarray(lst, dtype=int)  
b = a.reshape(2, 2)  
print(b)

Output:

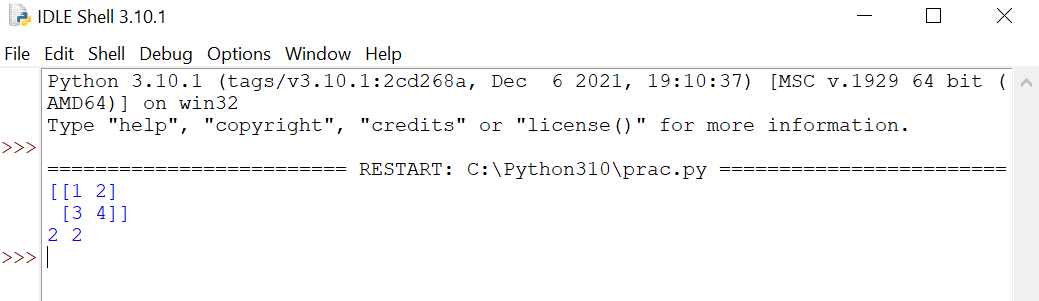


c. Find the rows and columns of the 2d array created in part b

Code:

rows = b.shape[0]  
cols = b.shape[1]  
print(rows, cols)

Output:

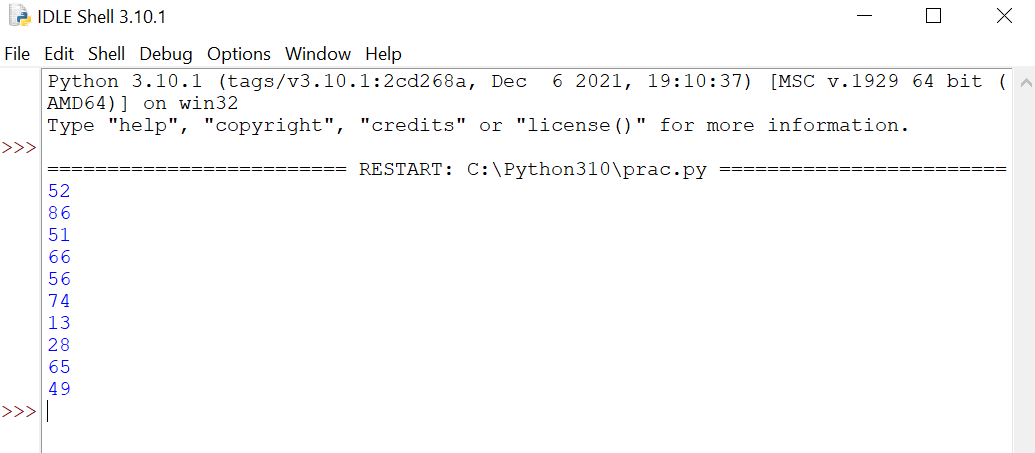


d. Print 10 random numbers between 1 and 100.

Code:

import numpy as np  
  
for i in range(10):  
 print(np.random.randint(1, 100))

Output:



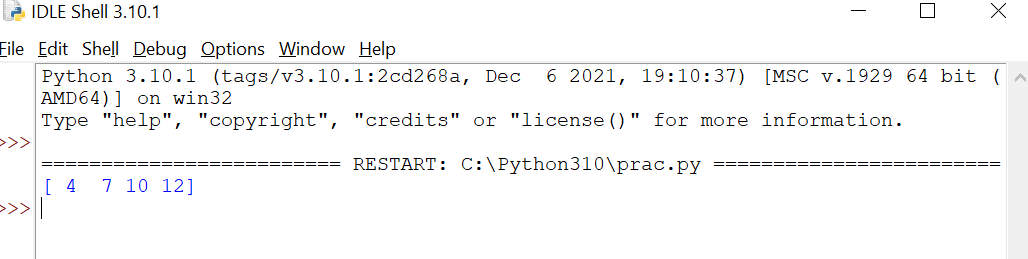
Q2.

a) Write a NumPy program to get help on the add function

Code:

import numpy as np  
  
array1 = np.array([1, 2, 3, 4])  
array2 = np.array([3, 5, 7, 8])  
  
  
def arrSum(*arr1*, *arr2*):  
 print(*arr1* + *arr2*)  
  
  
arrSum(array1, array2)

Output:

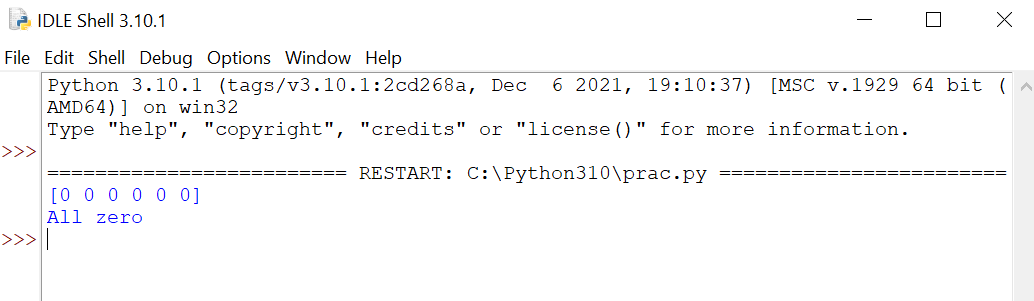


b) Write a NumPy program to test whether none of the elements of a given array is zero

Code:

import numpy as np  
  
arr = np.array([0, 0, 0, 0, 0, 0])  
check = np.all(arr == 0)  
print(arr)  
if (check == True):  
 print("All zero")  
else:  
 print("non zero element present")

Output:

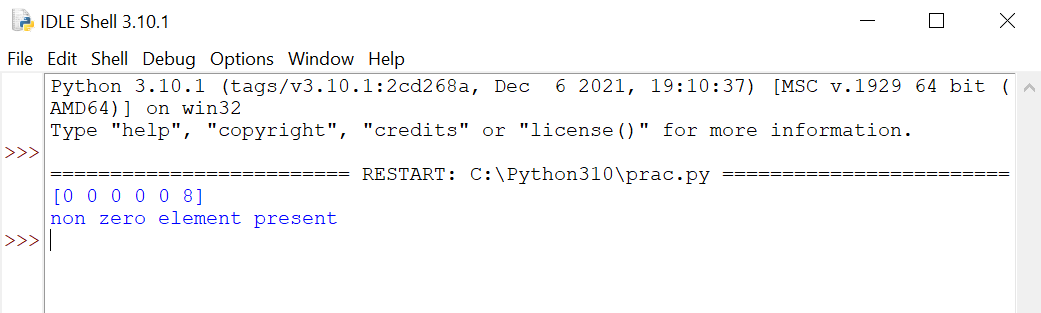


c) Write a NumPy program to test whether any of the elements of a given array is non-zero

Code:

import numpy as np  
  
arr = np.array([0, 0, 0, 0, 0, 8])  
check = np.any(arr != 0)  
print(arr)  
if (check == True):  
 print("non zero element present")  
else:  
 print(" All Zero")

Output:

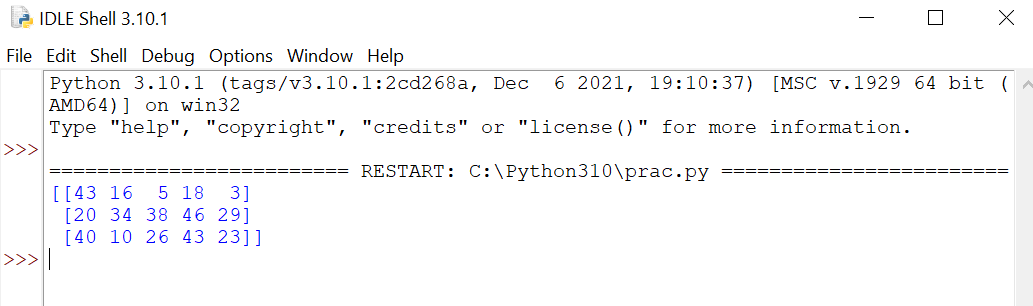


d) Write a NumPy program to generate an array of 15 random numbers from a standard normal distribution

Code:

import numpy as np  
  
arr = np.random.randint(1, 50, 15).reshape((3, 5))  
print(arr)

Output:



**EXPERIMENT-15: Pandas**

**Objective: Understand the usefulness of pandas module in python**

**Concept:**

**Pandas module:**Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.

**Advantages:**

Fast and efficient for manipulating and analysing data.

Data from different file objects can be loaded.

Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data

Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects

Data set merging and joining.

Flexible reshaping and pivoting of data sets

Provides time-series functionality.

Powerful group by functionality for performing split-apply-combine operations on data sets.

List of Lab Activities:

Q1. Refer the given excel file and perform various operations using pandas library:



a. Read the above excel file in python.

b. How do I write this file to a new file “new.csv”?

c. Include column names in this file. Use ‘ticker’, ‘eps’, ‘revenue’, ‘price’, ‘people’ as column names.

d. Convert all not available or n.a. values to NAN and also convert negative revenues to NAN because revenues can never be negative.

e. Fill NAN values using a suitable approach.

f. Write a function to change n.a value appearing in WMT to Sam Walton

Solution:

Code:

import pandas as pd

df = pd.read\_csv("data.csv")  
print(df)

s\_data = df.to\_csv('second.csv', index=True)

df.columns = ['ticker', 'eps', 'revenue', 'price', 'people']  
print('\n', df)

b = df.replace(to\_replace=['n.a.', -1, '-1'], value='NAN')

print('\n', b)

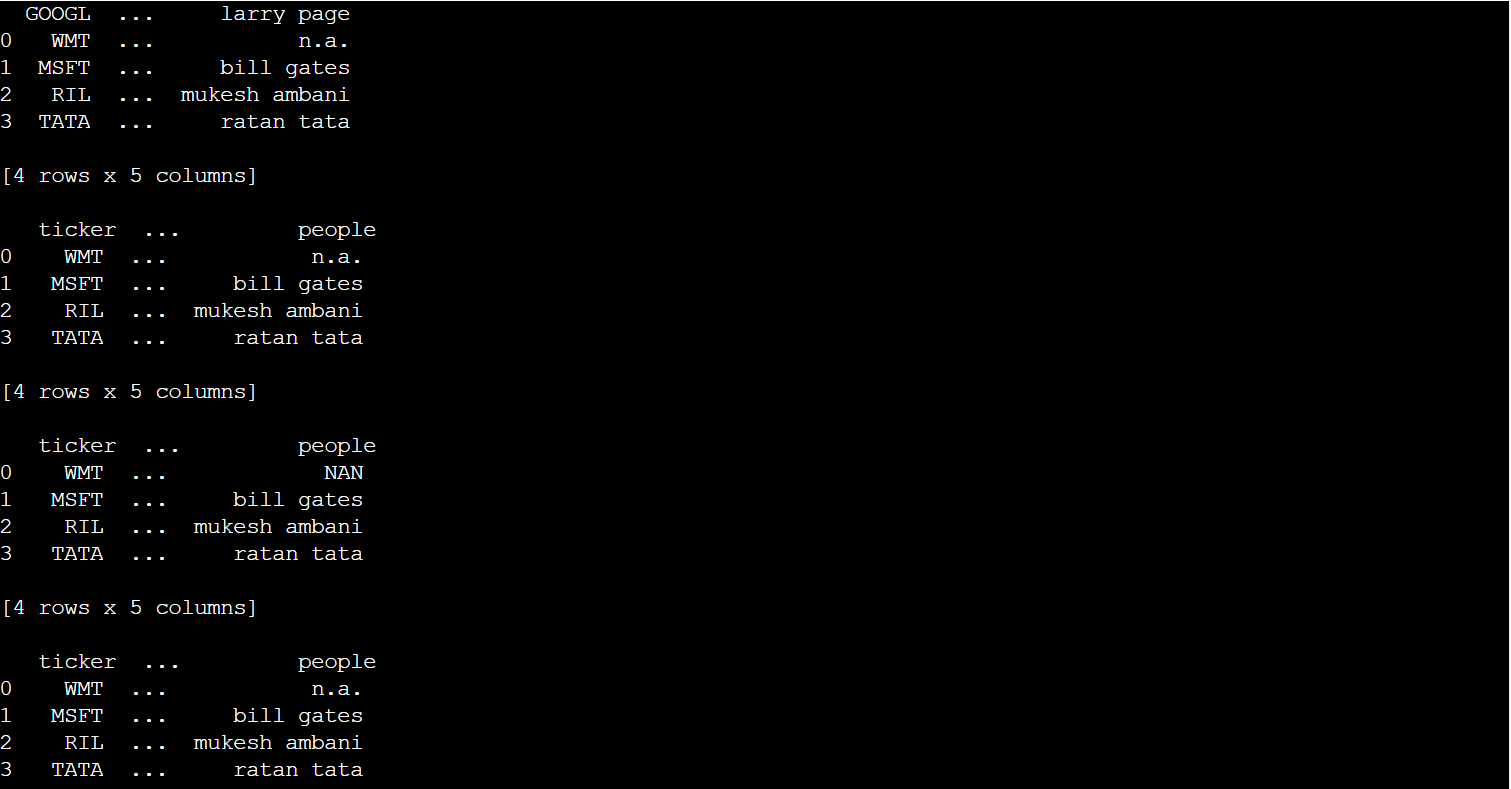
mean = df['revenue'].mean()  
df['revenue'] = df['revenue'].replace(to\_replace=-1, value=mean)

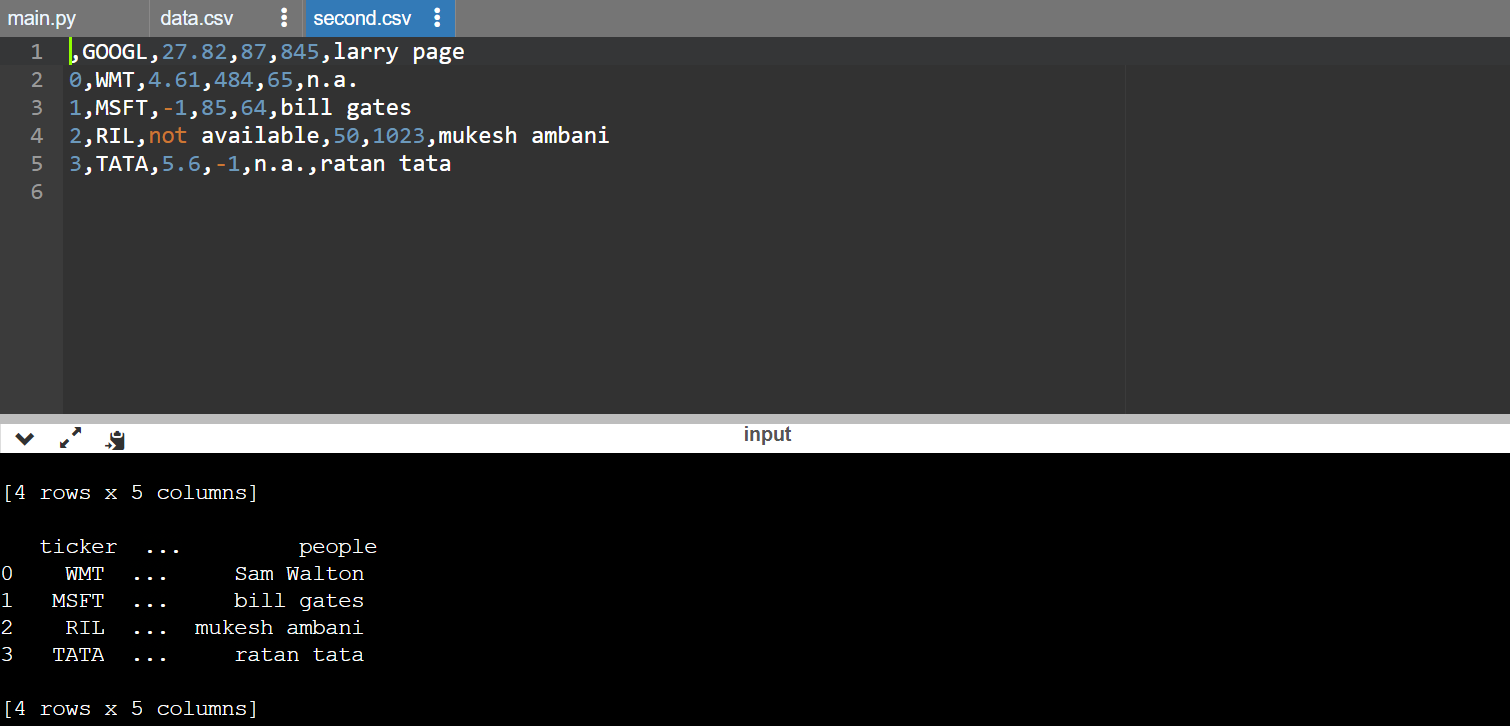
print('\n', df)

df['people'] = df['people'].replace(to\_replace='n.a.', value='Sam Walton')

print('\n', df)

**Output:**

****

****