1. Introduction to Python

Python is a widely used general-purpose, high level programming language. It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation. It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code. Python is a programming language that lets you work quickly and integrate systems more efficiently.

Python is:

- **Interpreted:** it can execute at runtime, and changes in a program are instantly perceptible. To be very technical, Python has a compiler. The difference when compared to Java or C++ is how transparent and automatic it is. With Python, we don't have to worry about the compilation step as it's done in real-time. The tradeoff is that interpreted languages are usually slower than compiled ones.

- **Semantically Dynamic:** you don't have to specify types for variables and there is nothing that makes you do it.

- **Object-Oriented:** everything in Python is an object. But you can choose to write code in an object-oriented, procedural, or even functional way.

- **High level:** you don't have to deal with low-level machine details.
  This lab is a course on the introduction of Python language. The lab allows you to learn the core of the language in a matter of hours instead of weeks.

1.1. Use of Python

Python has been growing a lot recently partly because of its many uses in the following areas:

- **System Scripting**: it's a great tool to automate everyday repetitive tasks.

- **Data Analysis**: it is a great language to experiment with and has tons of libraries and tools to handle data, create models, visualize results and even deploy solutions. This is used in areas like Finance, E-commerce, and Research.
- **Web Development**: frameworks like Django and Flask allow the development of web applications, API's, and websites.
- **Machine Learning**: Tensorflow and Pytorch are some of the libraries that allow scientists and the industry to develop and deploy Artificial Intelligence solutions in Image Recognition, Health, Self-driving cars, and many other fields.

1.2. Syntax

1.2.1. Semicolons

Python doesn't use semicolons to finish lines. A new line is enough to tell the interpreter that a new command is beginning.

The print () method will display something.

In this example, we have two commands that will display the messages inside the single quotes.

```
print('First command')
print('Second command')
```

But the following is wrong due to the semicolons in the end:

```
print('First command');
print('Second command');
```

1.2.2. Indentation

Many languages use curly-brackets to define scope.

Python's interpreter uses only indentation to define when a scope ends and another one starts.

This means you have to be aware of white spaces at the beginning of each line -- they have meaning and might break your code if misplaced.

This definition of a function works:

```
def my_function():
    print('First command')
```

This doesn't work because the indentation of the second line is missing and will throw an error:

```
def my_function():
print('First command')
```

1.2.3. Case Sensitivity and Variables

This Python is case sensitive. So the variables name and Name are not the same thing and store different values.

```
name = 'Renan'
Name = 'Moura'
```

As you can see, variables are easily created by just assigning values to them using the = symbol.

This means name stores 'Renan' and Name stores 'Moura'.

1.2.4. Comments

Finally, to comment something in your code, use the hash mark #.

The commented part does not influence the program flow.

```
# this function prints something
def my_function():
    print('First command')
```

1.2.5. Types

To store data in Python you need to use a variable. And every variable has its type depending on the value of the data stored. Python has dynamic typing, which means you don't have to explicitly declare the type of your variable -- but if you want to, you can. Lists, Tuples, Sets, and Dictionaries are all data types and have dedicated sections later on with more details, but we'll look at them briefly here.

1.2.6. User Input

If you need to interact with a user when running your program in the command line (for example, to ask for a piece of information), you can use the `input ()` function.

```
country = input("What is your country? ") #user enters 'Brazil'

print(country)
```

1.2.7. Operators

In a programming language, operators are special symbols that you can apply to your variables and values in order to perform operations such as arithmetic/mathematical and comparison. Python has lots of operators that you can apply to your variables and I will demonstrate the most used ones.

1.2.7.1. Arithmetic Operators

Arithmetic operators are the most common type of operators and also the most recognizable ones. They allow you to perform simple mathematical operations. They are:

- +: Addition

  -: Subtraction

  *: Multiplication

  /: Division

  **: Exponentiation

  //: Floor Division, rounds down the result of a division

  %: Modulus, gives you the remainder of a division

Let's see a program that shows how each of them is used.

```
print('Addition:', 5 + 2)
print('Subtraction:', 5 - 2)
print('Multiplication:', 5 * 2)
print('Division:', 5 / 2)
print('Floor Division:', 5 // 2)
print('Exponentiation:', 5 ** 2)
print('Modulus:', 5 % 2)
```

1.2.7.2. Concatenation

Concatenation is when you have two or more strings and you want to join them into one. This is useful when you have information in multiple variables and want to combine them.

For instance, in this next example I combine two variables that contain my first name and my last name respectively to have my full name.

The + operator is used to concatenate.

```
first_name = 'Renan '
last_name = 'Moura'

print(first_name + last_name)
```

1.2.7.3. Comparison Operators

Use comparison operators to compare two values. These operators return either True or False. They are:

- ==: Equal

  !=: Not equal

  >: Greater than

  <: Less than

  >=: Greater than or equal to

  <=: Less than or equal to

Let's see a program that shows how each of them is used.

```
print('Equal:', 5 == 2)
print('Not equal:', 5 != 2)
print('Greater than:', 5 > 2)
print('Less than:', 5 < 2)
print('Greater than or equal to:', 5 >= 2)
print('Less than or equal to:', 5 <= 2)
```

1.2.7.4. Logical Operators

Logical operators are used to combine statements applying Boolean algebra. They are:

and: True only when both statements are true

or: False only when both x and y are false

not: The not operator simply inverts the input, True becomes False and vice versa.

Let's see a program that shows how each one is used.

```
x = 5
y = 2

print(x == 5 and y > 3)

print(x == 5 or y > 3)

print(not (x == 5))
```

1.2.8. Conditionals

Conditionals are one of the cornerstones of any programming language. They allow you to control the program flow according to specific conditions you can check.

1.2.8.1. if statement

The way you implement a conditional is through the `if` statement. The general form of an `if` statement is:

```
if expression:
    statement
```

The `expression` contains some logic that returns a Boolean, and the `statement` is executed only if the return is `True`. A simple example:

```
bob_age = 32
sarah_age = 29

if bob_age > sarah_age:
    print('Bob is older than Sarah')
```

1.2.8.2. if else or elif statement

In our last example, the program only does something if the condition returns `True`.

But we also want it to do something if it returns `False` or even check a second or third condition if the first one wasn't met.

In this example, we swapped Bob's and Sarah's age. The first condition will return `False` since Sarah is older now, and then the program will print the phrase after the `else` instead.

```
bob_age = 29
sarah_age = 32

if bob_age > sarah_age:
    print('Bob is older than Sarah')
else:
    print('Bob is younger than Sarah')
```

Now, consider the example below with the `elif`.

```
bob_age = 32
sarah_age = 32

if bob_age > sarah_age:
    print('Bob is older than Sarah')
elif bob_age == sarah_age:
    print('Bob and Sarah have the same age')
else:
    print('Bob is younger than Sarah')
```

1.3. Objective of the Experiment

After completing this lab, the students should be able to:

- Understand Python and its syntax in detail.

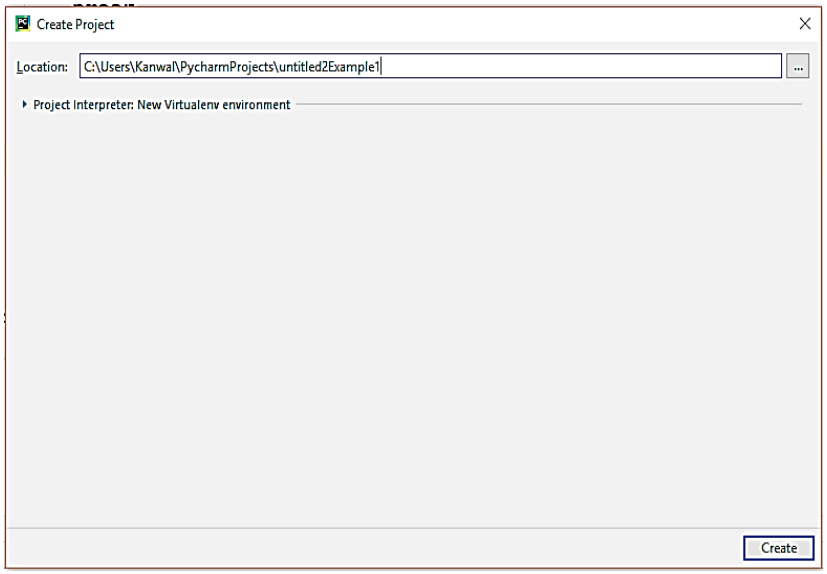
1.4. Practice Task

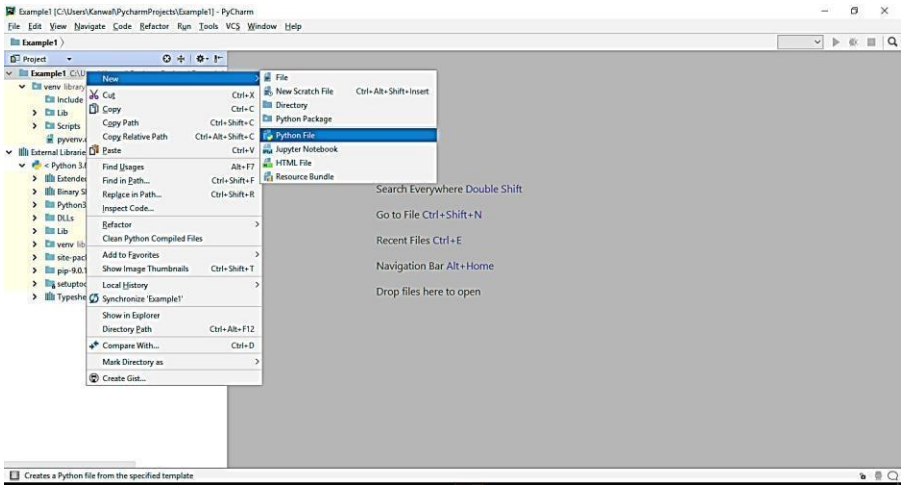
1. Go to the URL https://www.jetbrains.com/pycharm/download/?section=windows#section=windows . To download Pycharm python IDE and install it.
2. For Python installation, go to the URL

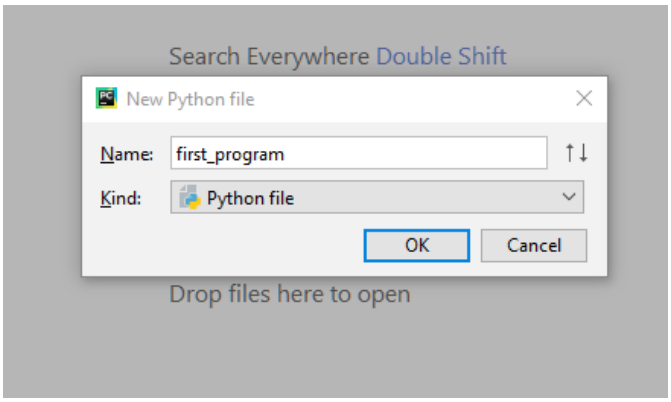
https://www.python.org/downloads/release/python-373/ and install **Windows x86-64 executable installer.**

3. Open Pycharm and create new project named **Example1** by clicking on **file** in menu bar and then on **new project.**

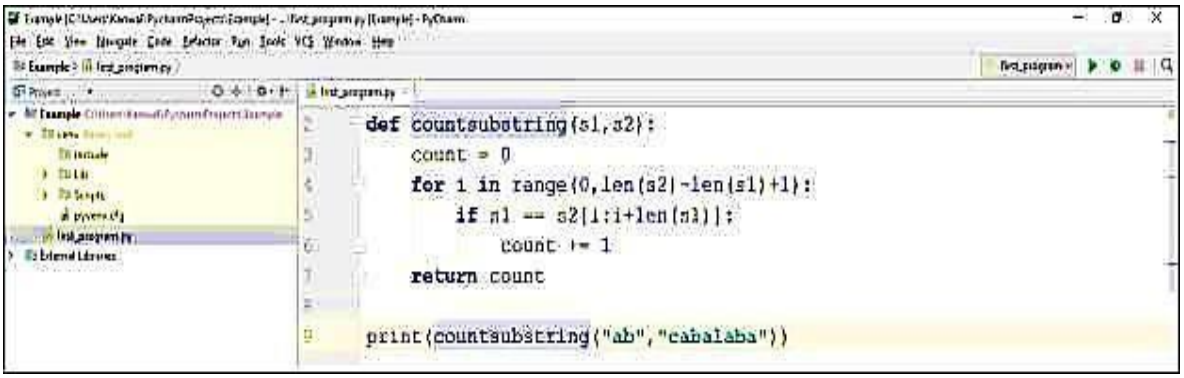
4. After creating new project right click on project name and create a **python file** name first program as shown in figures below.



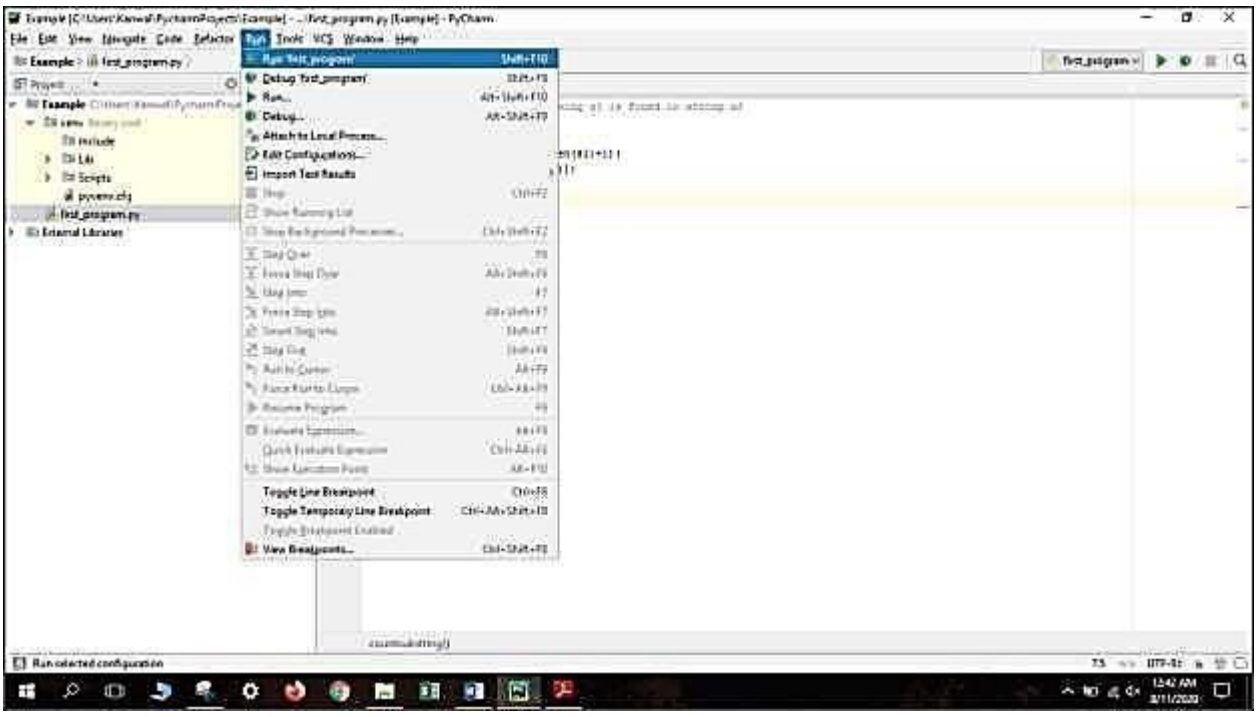




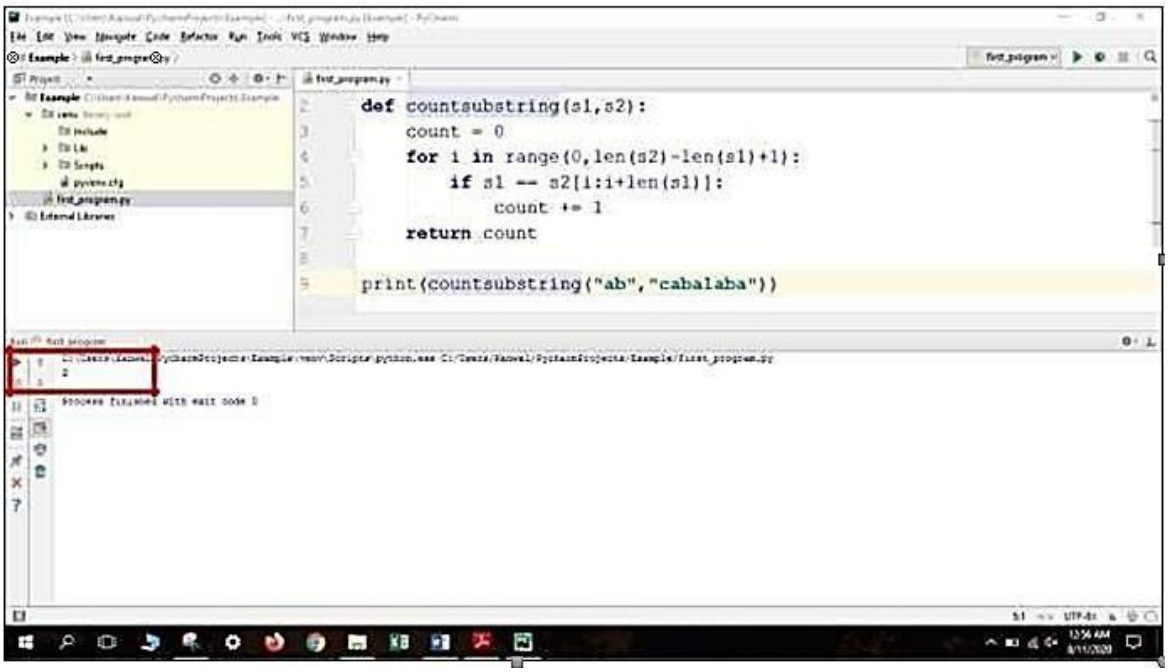
5. When file created write following code



6. Run file by clicking on Run in menu bar



7. After that you can see output in output box. Function returned 2.



1.5. Evaluation Tasks

1.5.1. Task 1 (Marks 10)

Write a python program to print the multiplication table for a number and range given by a user?

For example: the number is 2 and the range is 11 to 13 and the output will be:

2	x	11	=	22
2	x	12	=	24
2	x	13	=	26

1.5.2. Task 2 (Marks 10)

Write a python program using **function** to check whether the positive integer is prime or not. The number will be given by the user that lies between 0 to 1000. For example, for input 5 the output will be displayed as “prime number”. For input 6, the output will be “not a prime number”.

1.5.3. Task 3 (Marks 10)

Write a Python program that accepts a positive integer (n) and computes the sum of squares from 1 to n. For example: n is 3 so the sum will be 14 and for n is 5 the sum will be 55.

1.6. Evaluation using Rubrics

The evaluation criteria for this lab will be based on the completion of the following tasks. Each task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Criteria	Level 0 (0%)	Level 1 (1-50)%	Level 2 (51-65)%	Level 3 (66-80)%	Level 4 (81-100)%	Total Score
Procedural Awareness						
Practical Implementation						
Program Correctness						
Use of Software Tool						
Sub Total Marks						

1.7. Out comes

After completing this lab, students will be able to learn Python syntax and implement different Python concepts in detail.

1.8. Further Readings

1.8.1. Books

Artificial Intelligence: A modern approach 3rd Edition by Stuart Russell & Peter Norvig

1.8.2. Links

https://pynative.com/python-basic-exercise-for-beginners/

https://www.w3resource.com/python-exercises