**Lab # 1 Basic Programming in python**

**OBJECTIVES**

In this lab you will learn:

1. Usage of the Python interpreter
2. Basic arithmetic expressions
3. Representing and operating the numbers
4. Usage of print statement

Using the Python interpreter

1. Open PyCharm Interpreter.
2. Create new Project from the file menu, Name the project and then Select **Python 3.7 interpreter**
3. A new window will open and here you will do python coding and execute program by clicking on **Run** icon

**WHAT IS PYTHON?**

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.

It is used for:

* web development (server-side),
* software development,
* mathematics,
* system scripting.

**What can Python do?**

* Python can be used on a server to create web applications.
* Python can be used alongside software to create workflows.
* Python can connect to database systems. It can also read and modify files.
* Python can be used to handle big data and perform complex mathematics.
* Python can be used for rapid prototyping, or for production-ready software development.

**Why Python?**

* Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
* Python has a simple syntax similar to the English language.
* Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
* Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
* Python can be treated in a procedural way, an object-oriented way or a functional way.

**Good to know**

* The most recent major version of Python is Python 3, which we shall be using in this tutorial. However, Python 2, although not being updated with anything other than security updates, is still quite popular.
* In this tutorial Python will be written in a text editor. It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

**Python Syntax compared to other programming languages**

* Python was designed for readability, and has some similarities to the English language with influence from mathematics.
* Python uses new lines to complete a command, as opposed to other programming languages which often use semicolons or parentheses.
* Python relies on indentation, using whitespace, to define scope; such as the scope of loops, functions and classes. Other programming languages often use curly-brackets for this purpose.

**Task 1**

**print("Hello, World!")**

The output is Hello World





**Task 2**

We start by typing simple instructions in the interpreter window. The goal is to get comfortable with basic Python instructions. Note the cursor blinking. This means Python is ready for your instructions. Type into the interpreter window:

**X = 2**

**Y = 3**

**Z = X + Y**

**print (Z)**

Hit **run** to execute the instruction and the outcome of this code.

Here you instructed the computer to define a variables named X & Y**,** and store the Sum of values stored in X & Y into Z.

**Naming the Variables**

Variables are containers for storing data values. Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

You may choose any name (legal in Python) for your variables, assign a value to it and use this value later. Some rules about legal names in Python

• Names can contain numbers and letters.

• Names cannot contain spaces.

• The underscore character \_ is legal (e.g., my\_count)

• The first character has to be a letter.

• Upper and lower case letters are different (e.g., “X” is not the same as “x”).

• Words that are part of the Python language cannot be used.

If you try to give a variable an illegal name, you get a syntax error.

**Task 3**

**myvar = "John"**

**my\_var = "John"**

**\_my\_var = "John"**

**myVar = "John"**

**MYVAR = "John"**

**myvar2 = "John"**

**print(myvar)**

**print(my\_var)**

**print(\_my\_var)**

**print(myVar)**

**print(MYVAR)**

**print(myvar2)**

**Task 4**

**oranges = 3\*2 - 5\*6**

**print (oranges)**

**Task 5**

**Y=2**

**X=Y\*\*3**

**print (X)**

It raises the value of Y (which is 2) to the power of 3 (in others words 23), which we know as exponentiation. In Python, \*\* represents exponentiation.

**Task 6**

Declare the following variable

**a=13**

**x=5.0**

**y=13.0**

**m="Mary"**

**n="Nancy"**

**print( a )**

**print( a, x, m, n )**

b=16

**print( a + b )**

This line retrieves the values of a and b from memory, adds them together, and prints the result on the output window. Neither a nor b are altered in the process. Alternately, we could have created a brand new variable and printed it. The result will be the same.

Enter the following two lines to verify this:

**c = a + b**

**print( c )**

The only difference is that this version adds a new “slot” called c to the memory map above.

Besides addition, the other main math operators are ‒, \* (multiplication), / (division), \*\*

(exponents, which can also be performed using the function pow(x,y) for xy), % (modulo) . Parentheses () may be used to force the execution of some operations before others. Parentheses have the highest precedence and are followed by multiplication, division, addition and subtraction. That is, the expression a=b+c\*d will multiply c by d before b is added. To force the addition first, use parentheses: a=(b+c)\*d

Continuing with the other math operators, type:

**print( y/x )**

**print( b/a )**

The result is also 2.6 even though both variables are integers (an integer, of course, can’t contain a fractional portion). In essence, Python *promotes* the variables to floats in order to maintain precision, producing a floating point answer. Now try:

**print( b/x )**

In this case the answer is again 2.6. This is because in a mixed calculation between a float and an integer, the integer is again promoted to a float in the calculation in order to maintain the precision of the floating point variable.

**print( b%a )**

The modulo operator produces the remainder of the divide, that is, a goes into b two whole times with 3 left over.

**>>>print( x/y )**

**0.38461538461538464**

To limit this to fewer digits, the round() function may be used. The first argument is the value of expression to be rounded and the second is the number of digits after the decimal point. Now enter this:

**>>>print( round(x/y,3) )**

**0.385**

**Task 7**

**The Python math module**

We can do many interesting things with variables. If we want to use mathematical functions, we need to import them first (as done below):

**Import math**

**a = 4**

**b = 3**

**c = math.sqrt(a\*\*2 + b\*\*2)**

**print (c)**

In order to use the **sqrt** function, we needed to **import** it from the **math** module. This is what the

statement **math import** does.

We will import functions from modules as we need them. Here is a list of other commonly used function which have to be imported from the math module

|  |  |
| --- | --- |
| *Function* | *returns* |

|  |  |
| --- | --- |
| **sin**(*x*) | the sine of *x*, in radians |
| **cos**(*x*) | the cosine of *x*, in radians |
| **tan**(*x*) | the tangent of *x*, in radians |
| **acos**(*x*) | the arc cosine of *x*, in radians |
| **asin**(*x*) | the arc sine of *x*, in radians |
| **atan**(*x*) | the arc tangent of *x*, in radians |
| **log**(*x*) | the natural logarithm of *x* |
| **log10**(*x*) | the base-10 logarithm of *x* |
| **pi** | an approximation of the mathematical constant *pi* |

A function needs to be imported only once in a session or in a program. If one wants to import all

functions in the math module, write **math import**

Start a new interpreter session to see what happens if one forgets to import a function used:

**>>> sqrt(44)**

**Traceback (most recent call last):**

**File "<pyshell#0>", line 1, in <module>**

**sqrt(44)**

**NameError: name 'sqrt' is not defined**

**Task 8**

You can input numbers in Python using scientific notation as you would on a calculator. For example,

5e10 is equal to 5×1010

**A= 5e10**

**print (A)**

You can print the specific statement that will elaborate more precisely, For example

**A= 5e10**

**print ("A=", A)**

**LAB ASSIGNMENT 1**

In the interpreter window, set the values of two variables:

total = 55

count = 6

a). What does the following print statement generate?

**print(total, count)**

b). Write a print statement that prints the word “and” (without quotes) between the numbers. (print needs to use the variables)

**print** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

c). Write the print statement generating the following output (the print statement needs to use the variables)

**print** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**The 6 elements sum up to 55**

d). What does the following print statement generate?

**print (“the sum of total and count is”, total+count)**

**LAB ASSIGNMENT 2**

Write a program to perform Mathematical Operations like +, -, X & /.

Your program will take the input numbers and required operation from the user via user inputs and perform the required operation on those numbers.

Your program will show the variable name first then its stored value, and then the answer of specific operation.