# Experiment 1 – Introduction to Python - I

**Objective:** To provide basic knowledge about the python language: Expressions and statements, Operators, Variables, Lists, Strings, Functions,

**Time Required** : 3 hrs

**Programming Language** : Python

**Software Required** : Anaconda/Google Colab

# Learning the basics of Python Language

**Introduction**

Python is a general-purpose programming language used in just about any kind of software you can think of. You can use it to build websites, artificial intelligence, servers, business software, and more.

Python is a portable, cross-platform language —you can write and execute Python code on any operating system with a Python interpreter.

**Expressions and statements**:

A computer program written in Python is built from statements and expressions. Each statement performs some action. For example, the built-in print statement writes to the screen:

A screenshot of a computer

Description automatically generated

The single quotation marks in our print statement indicate that what we are printing is a string – a sequence of letters or other symbols. If the string is to contain a single quotation mark, we must use the double quotation mark key instead otherwise it will generate an error.

A screenshot of a computer

Description automatically generated

Each expression performs some calculation, yielding a value. For example, we can calculate the result of a simple mathematical expression using whole numbers (or integers):

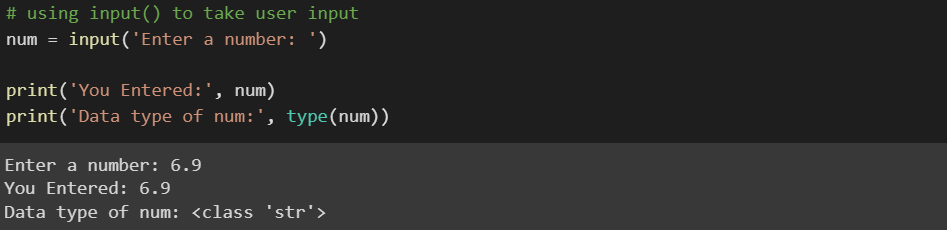
Graphical user interface, text, application, email

Description automatically generated

When Python has calculated the result of this expression, it prints it to the screen, even though we have not used print. All our programs will be built from such statements and expressions.

**Python Input**

While programming, we might want to take the input from the user. In Python, we can use the input() function.



In the above example, we have used the input() function to take input from the user and stored the user input in the num variable.

It is important to note that the entered value 10 is a string, not a number. So, type(num) returns <class 'str'>.

To convert user input into a number we can use int() or float() functions as:

Graphical user interface

Description automatically generated with medium confidence

**Creating Variables:**

In programming, a variable is a container (storage area) to hold data. It has no command for declaring a variable. A variable is created the moment you first assign a value to it. It allows you to assign values to multiple variables in one line:

Text

Description automatically generated

Variables do not need to be declared with any particular type and can even change type after they have been set. (**Note:** variable names are case sensitive)

If we want to assign the same value to multiple variables at once, we can do this as:

A picture containing background pattern

Description automatically generated

You can get the data type of a variable with the type() function.

A picture containing graphical user interface

Description automatically generated

**Python Literals**

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, 'Hello, World!', 12, 23.0, 'C', etc.

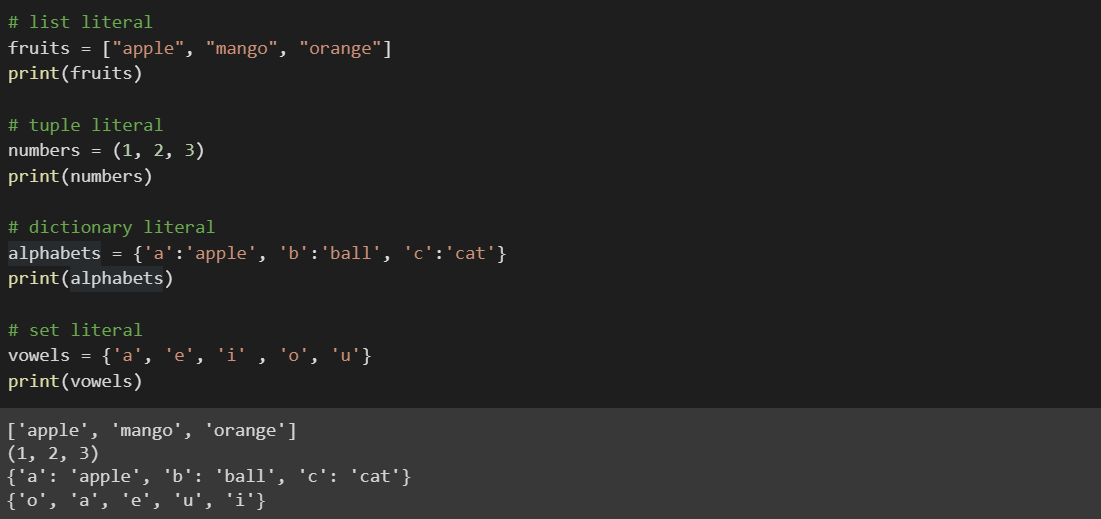
Literals are often used to assign values to variables or constants. For example:



In the above expression, site\_name is a variable, and 'programiz.com' is a literal.

**Literal Collections**

There are four different literal collections List literals, Tuple literals, Dict literals, and Set literals.



In the above example, we created a list of fruits, a tuple of numbers, a dictionary of alphabets having values with keys designated to each value and a set of vowels.

**Python Data Types**

|  |  |  |
| --- | --- | --- |
| Data Types | Classes | Description |
| Numeric | int, float, complex | holds numeric values |
| String | str | holds sequence of characters |
| Sequence | list, tuple, range | holds collection of items |
| Mapping | dict | holds data in key-value pair form |
| Boolean | bool | holds either True or False |
| Set | set, frozeenset | hold collection of unique items |

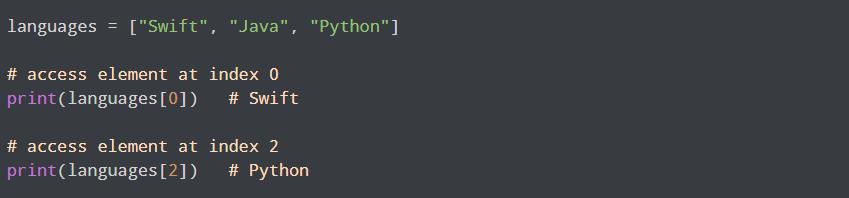
Since everything is an object in Python programming, data types are classes and variables are instances(object) of these classes.

**List Data Type**

List is an ordered collection of similar or different types of items separated by commas and enclosed within brackets [ ]. For example,

**Access List Items:** Here, we have created a list named languages with 3 string values inside it.

To access items from a list, we use the index number (0, 1, 2 ...). For example,



We can measure the length of list by following command:

***len(languages)***

**Concatenate two lists:** Adding two lists by using simple ‘+’ operator creates a new list with everything from the first list, followed by everything from the second list.

We use append() to a list to add a single item to a list and the list itself is updated as a result of the operation.

***languages.append(“C++”)***

The following command prints all words present before the 2nd index in language list.

***languages[:2]***

The following command prints all words present after the 2nd index in language list.

***languages[2:]***

By convention, m:n means elements m…n-1.

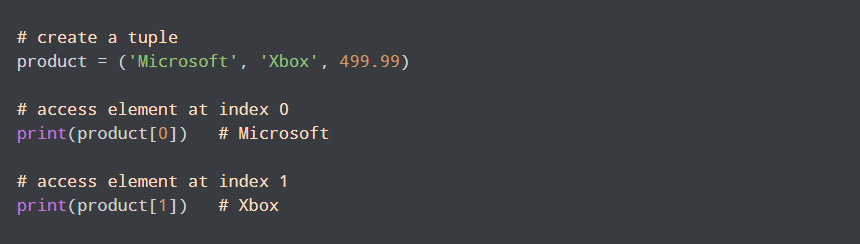
***languages[0:2]***

We can also slice with negative indexes — the same basic rule of starting from the start index and stopping one before the end index applies.

**Python Tuple Data Type**

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

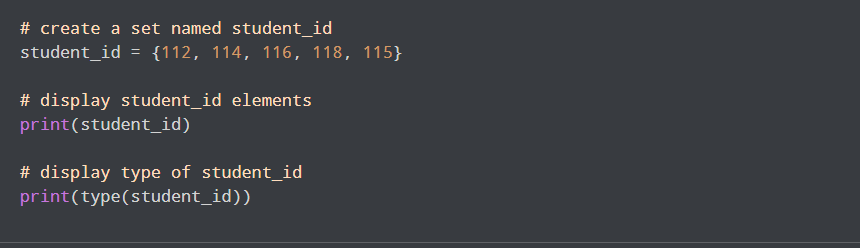
In Python, we use the parentheses () to store items of a tuple. For example,

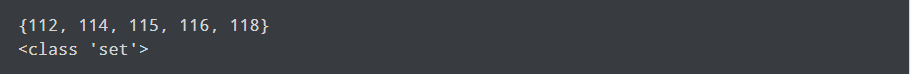


Here, product is a tuple with a string value Xbox and integer value **499.99**.

**Python Set Data Type**

Set is an unordered collection of unique items. Set is defined by values separated by commas inside braces { }. For example,





Here, we have created a set named student\_info with 5 integer values.

Since sets are unordered collections, indexing has no meaning. Hence, the slicing operator [] does not work.

**Dictionary Data Type**

Python dictionary is an ordered collection of items. It stores elements in key/value pairs.

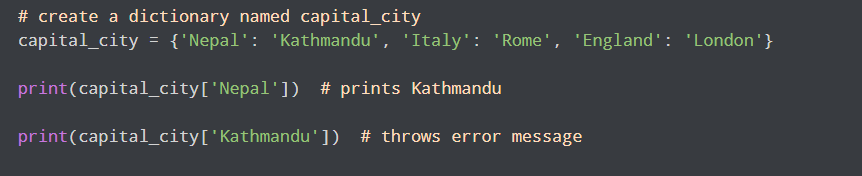
Here, keys are unique identifiers that are associated with each value.

**Access Dictionary Values Using Keys:** We use keys to retrieve the respective value. But not the other way around. For example,

In the below example, we have created a dictionary named capital\_city. Here,

*Keys* are 'Nepal', 'Italy', 'England'

*Values* are 'Kathmandu', 'Rome', 'London'



|  |  |
| --- | --- |
| Methods | Description |
| *list.*append(x) | Add an item to the end of the list. |
| *list.*extend(L) | Extend the list by appending all the items in the given list. |
| *list.*insert(i, x) | Insert an item at a given position. The first argument is the index of the element before which to insert, so a.insert (0,x) inserts at the front of the list, and a.insert(len(a), x) is equivalent to a.append(x). |
| *list.*remove(x) | Remove the first item from the list whose value is x. It is an error if there is no such item. |
| *list.*pop([i]) | Remove the item at the given position in the list, and return it. If no index is specified, a.pop() removes and returns the la the list. (The square brackets around the iin the method signature denote that the parameter is optional, not that you should square brackets at that position. You will see this notation frequently in the Python Library Reference.) |
| list.sort( ) | Sort the items of the list, in place. |
| list.reverse( ) | Reverse the elements of the list, in place. |

Some of the methods we used to access the elements of a list also work with individual words, or strings. For example, we can assign a string to a variable, index a string, and slice a string:

Graphical user interface, text, application, email

Description automatically generated

We can also perform multiplication and addition with strings:

Graphical user interface, text, application, email

Description automatically generated

We can join the words of a list to make a single string, or split a string into a list, as follows:

Graphical user interface, text, application, email

Description automatically generated

**Functions:**

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

**Creating & Calling a Function**

In Python a function is defined using the **def**keyword:

Graphical user interface, text, application, email

Description automatically generated

**Arguments**

Information can be passed into functions as arguments. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

A screenshot of a computer

Description automatically generated

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you have to call the function with 2 arguments, not more, and not less.

This function expects 2 arguments, and gets 2 arguments:

Graphical user interface, text, application, email

Description automatically generated

If you try to call the function with 1 or 3 arguments, you will get an error. Moreover, If we call the function without argument, it uses the default value.

**Return Values**

To let a function, return a value, use the ***return***statement.

A screenshot of a computer

Description automatically generated

**Lab Tasks:**

**Q1:** Write a program to:

1. Define a string with your name and assign it to a variable. Print the contents of this variable in two ways, first by simply typing the variable name and pressing enter, then by using the print statement.
2. Try adding the string to itself using my\_string + my\_string, or multiplying it by a number, e.g., my\_string \* 3. Notice that the strings are joined together without any spaces. How could you fix this?

**Q2.** Write a python program to create a function arithematic\_operation to perform arithmetic operations(+, -, \*, /) using two numbers.

**Q3:** Given two strings, s1, and s2 return a new string made of the first, middle, and last characters each input string.

**Given:**

s1 = "America"

s2 = "Japan"

**Expected Output:**

AJrpan

**Q4**: Write a program to create a function that takes wo strings s1 and s2 and create a new string by appending s2 in the middle of s1.

**Given:**

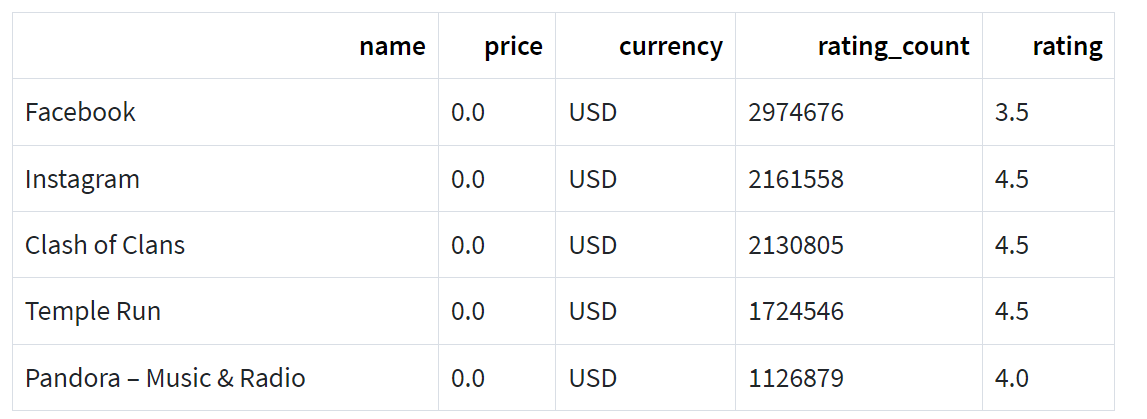
s1 = "Software"

s2 = "Design"

**Expected Output:**

SoftDesignware

**Q5**. Write a program to create five lists, one for each row in our dataset:



Also use list indexing to extract the number of ratings from the five rows and then average them.

**Q6.** Write a program to create the course variable then set the course variable to be an empty list.

1. Now, Add 'Machine Learning', 'Software Construction', and 'Formal Methods' to the users list in that order without reassigning the variable.
2. Delete software construction and display the updated list content.
3. Add the course 'Artificial Intelligence' to course where ' Software Construction' used to be.
4. Slice course to Return 1st and 3rd Elements

**References:**

1. <https://mas-dse.github.io/startup/anaconda-windows-install/#anaconda>
2. <https://www.programiz.com/python-programming/first-program>
3. <https://www.codecademy.com/learn/learn-python-3>
4. <https://www.w3schools.com/python/>