The Basic Python Tutorial

Uri Arev

Contents

Chapter 1

Basics

Input/Output

Output

In Python 3, unlike C and C++, there is no need to use any I/O Library. To create a simple "Hello, World!" output, we only need to write one line. To do that, we need to create a Python file. We will name it *hello.py*. *"hello"* is the file name, while *".py"* is the file extension.

```
# hello.py
print('Hello, World!')
```

Let's look at the code we wrote. At line one, we wrote "print(*string*)", where *string* is "Hello, World!". The *print()* function can take all kinds of variables, not just strings. Notice that to create a comment, we use '#' characters. If we run the code with the command *python3 hello.py*, the program will print:

```
Hello, World!
```

We can also use *Escape Sequences* in strings. For example:

```
# hello.py
print("Hello,\nWorld")
```

outputs:

```
Hello,
World!
```

Notice that we can use the ' or " symbols for strings.

Input

To get a user's keyboard input, we can use a function called *input()*. This function lets the user enter a string with a keyboard, waits for a *newline* character (also known as $\backslash n$ or *ENTER*) and returns the string. For example:

```
# input.py
print(input("Name:"))
```

Take a look at the code above. In the code, we wrote "input(*string*)" where *string* is "Name:". The argument (*string*) is the prompt; what the user will see before he is allowed to use his keyboard. Let's run the code again:

```
Name:
```

Notice that we can now use the keyboard to type something. And after we press enter, the program will print what the user just typed.

```
Name:Test
Test
```

When we look at the example above, we will see that the user typed "Test". The program then printed "Test". Notice that the *input()* function DOES return the newline too.

Variables

The *type()* Function

The *type()* returns the type of the object passed to it. For example, we can use the *type()* function to print the type of a string:

```
x = "Hello"
print(type(x))
```

and it will return:

```
<class 'str'>
```

or we can check the type of an integer using the following code:

```
x = 127
print(type(x))
```

Strings

The *string* type allows us to store multiple characters. There are two ways to create multiple-line strings. The first one is to use *newlines*. We can do that by adding a '\n' character to the string. Example:

```
print("This is the first line.\nAnd this is the second.")
```

or, we can use three double-quotes to create a multiple-line string.

```
print("""This is the first line.
And this is the second.""")
```

or:

```
print('''This is the first line.
And this is the second.''')
```

Integers & Floats

Integers

The integer type allows storage of integers. For example, we can initialize an integer variable like the following snippet:

```
x = 739
```

Let's look at the line above: We wrote "x = 739". x is the name of the variable, and 739 is the value of it. We can also check the type of x using the *type()* function in the following code:

```
print(type(x))
```

which will print:

```
<class 'int'>
```

Floats

The float type stores decimal numbers, and can be initialized like the following:

```
x = 547.23
```

and its type will be:

```
<class 'float>
```

Lists/Arrays

Lists (aka arrays) can store multiple values including strings, integers (and floats) and even other lists!

We can initialize a simple list with the following snippet:

```
integerArray = [1, 4, 72, 7]
```

and we can access one of the values in the list using square brackets. Example:

```
print(integerArray[0])
```

will output:

```
1
```

notice that when accessing a value in a list, the number *1* points to the first value in the list, *2* points to the second value, and so forth.

We can also create a list with multiple types of values like the following:

```
array = [16, "hello", 712.48]
```

and access it with:

```
print(array[1])
```

or print it all with:

```
print(array)
```

which will output the following text:

```
[16, "hello", 712.48]
```