

Numbers

August 20, 2017

1 Numbers and more in Python!

In this lecture, we will learn about numbers in Python and how to use them.

We'll learn about the following topics:

- 1.) Types of Numbers in Python
- 2.) Basic Arithmetic
- 3.) Differences between Python 2 vs 3 in division
- 4.) Object Assignment in Python

1.1 Types of numbers

Python has various “types” of numbers (numeric literals). We'll mainly focus on integers and floating point numbers.

Integers are just whole numbers, positive or negative. For example: 2 and -2 are examples of integers.

Floating point numbers in Python are notable because they have a decimal point in them, or use an exponential (e) to define the number. For example 2.0 and -2.1 are examples of floating point numbers. 4E2 (4 times 10 to the power of 2) is also an example of a floating point number in Python.

Throughout this course we will be mainly working with integers or simple float number types.

Here is a table of the two main types we will spend most of our time working with some examples:

Numbers in Python

Examples

Number “Type”

1,2,-5,1000

Integers

1.2,-0.5,2e2,3E2

Floating-point numbers

Now let's start with some basic arithmetic.

1.1.1 Basic Arithmetic

```
In [1]: # Addition
        2+1
```

```
Out[1]: 3
```

```
In [2]: # Subtraction
        2-1
```

```
Out[2]: 1
```

```
In [3]: # Multiplication
        2*2
```

```
Out[3]: 4
```

```
In [4]: # Division
        3/2
```

```
Out[4]: 1
```

1.1.2 Python 3 Alert!

Whoa! What just happened? Last time I checked, 3 divided by 2 is equal 1.5 not 1!

The reason we get this result is because we are using Python 2. In Python 2, the / symbol performs what is known as “classic” division, this means that the decimal points are truncated (cut off). In Python 3 however, a single / performs “true” division. So you would get 1.5 if you had inputted 3/2 in Python 3.

So what do we do if we are using Python 2 to avoid this?

There are two options:

Specify one of the numbers to be a float:

```
In [11]: # Specifying one of the numbers as a float
         3.0/2
```

```
Out[11]: 1.5
```

```
In [12]: # Works for either number
         3/2.0
```

```
Out[12]: 1.5
```

We could also “cast” the type using a function that basically turns integers into floats. This function, unsurprisingly, is called float().

```
In [14]: # We can use this float() function to cast integers as floats:
         float(3)/2
```

```
Out[14]: 1.5
```

We will go over functions in much more detail later on in this course, so don’t worry if you are confused by the syntax here. Consider this a sneak preview.

One more “sneak preview” we can use to deal with classic division in Python 2 is importing from a module called **future**.

This is a module in Python 2 that has Python 3 functions, this basically allows you to import Python 3 functions into Python 2. We will go over imports and modules later in the course, so don’t worry about fully understanding the import statement right now!

```
In [15]: from __future__ import division
         3/2
```

```
Out[15]: 1.5
```

When you import division from the **future** you won't need to worry about classic division occurring anymore anywhere in your code!

1.1.3 Arithmetic continued

```
In [16]: # Powers
         2**3
```

```
Out[16]: 8
```

```
In [17]: # Can also do roots this way
         4**0.5
```

```
Out[17]: 2.0
```

```
In [18]: # Order of Operations followed in Python
         2 + 10 * 10 + 3
```

```
Out[18]: 105
```

```
In [19]: # Can use parenthesis to specify orders
         (2+10) * (10+3)
```

```
Out[19]: 156
```

1.2 Variable Assignments

Now that we've seen how to use numbers in Python as a calculator let's see how we can assign names and create variables.

We use a single equals sign to assign labels to variables. Let's see a few examples of how we can do this.

```
In [37]: # Let's create an object called "a" and assign it the number 5
         a = 5
```

Now if I call *a* in my Python script, Python will treat it as the number 5.

```
In [38]: # Adding the objects
         a+a
```

```
Out[38]: 10
```

What happens on reassignment? Will Python let us write it over?

```
In [39]: # Reassignment
         a = 10
```

```
In [40]: # Check
a
```

```
Out[40]: 10
```

Yes! Python allows you to write over assigned variable names. We can also use the variables themselves when doing the reassignment. Here is an example of what I mean:

```
In [41]: # Check
a
```

```
Out[41]: 10
```

```
In [42]: # Use A to redefine A
a = a + a
```

```
In [43]: # Check
a
```

```
Out[43]: 20
```

The names you use when creating these labels need to follow a few rules:

1. Names can not start with a number.
2. There can be no spaces in the name, use `_` instead.
3. Can't use any of these symbols : `'", <>/?|\()!@#$$%^&*~+-`
3. It's considered best practice (PEP8) that the names are lowercase.

Using variable names can be a very useful way to keep track of different variables in Python. For example:

```
In [44]: # Use object names to keep better track of what's going on in your code!
my_income = 100

tax_rate = 0.1

my_taxes = my_income*tax_rate
```

```
In [46]: # Show my taxes!
my_taxes
```

```
Out[46]: 10.0
```

So what have we learned? We learned some of the basics of numbers in Python. We also learned how to do arithmetic and use Python as a basic calculator. We then wrapped it up with learning about Variable Assignment in Python.

Up next we'll learn about Strings!