

A (debatably) Brief Introduction to Python



CHAPTER 1: PRIMITIVES AND VARIABLES (AND FUNCTIONS, WHY NOT?)

Okay, so let's get into some *real* programming, shall we? First item of business: primitives. These are the most basic data types in python, and all other programming languages. They include:

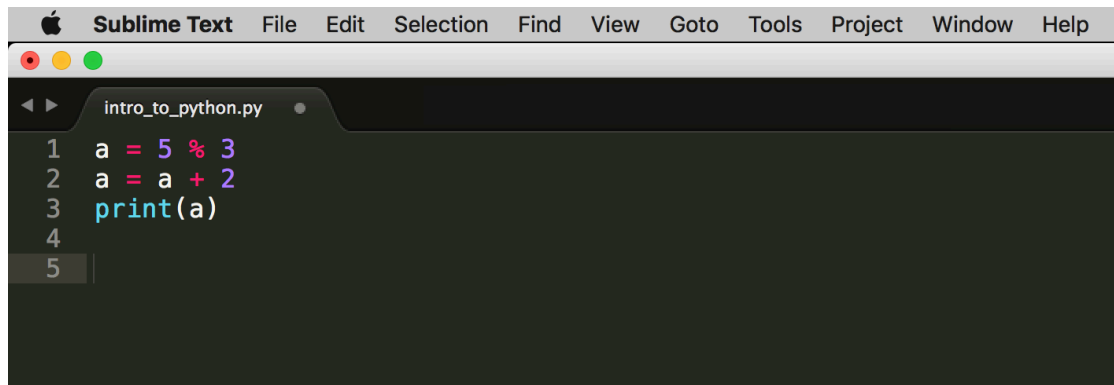
- Integers (ints)
 - o These are pretty self explanatory. They're integers. There's no decimal at the end of them.
- Floats
 - o These are a little more interesting – numbers with decimals at the end of them.
- Strings
 - o Strings are just words, starting and ending with “. They're a little different from the rest of the primitives, in that they have a whole host of built-in functions associated with them, which is great. As a result, however, I'm dedicating chapter 2 to them so that you can have a nice understanding of them. Also, strings most certainly are NOT primitives in other languages like C and Java (for complicated reasons).
- Booleans
 - o Booleans are, in effect, truth values. There are only 2 possible values for booleans, and they are **True** and **False**.

Now that we have a basic overview of all the primitive types, let's find out how to do stuff with them, starting with integers and floats. The biggest thing you can do with integers and floats is math. The math operators in python include:

- **+**, which adds numbers together.
- **-**, which calculates the difference between two numbers.
- **/**, which divides two numbers.
- *****, which multiplies two numbers.
- ******, which is the exponent sign. This is a little iffy to explain like the others, so let me give an example to clear things up. **4**5** will raise 4 to the fifth power.
- **%**, which is the modulo sign. It calculates the remainder when the number on the left is divided by the number on the right.

There are more advanced math functions available in the **math** library, but we won't be needing those until the machine learning segment of this course.

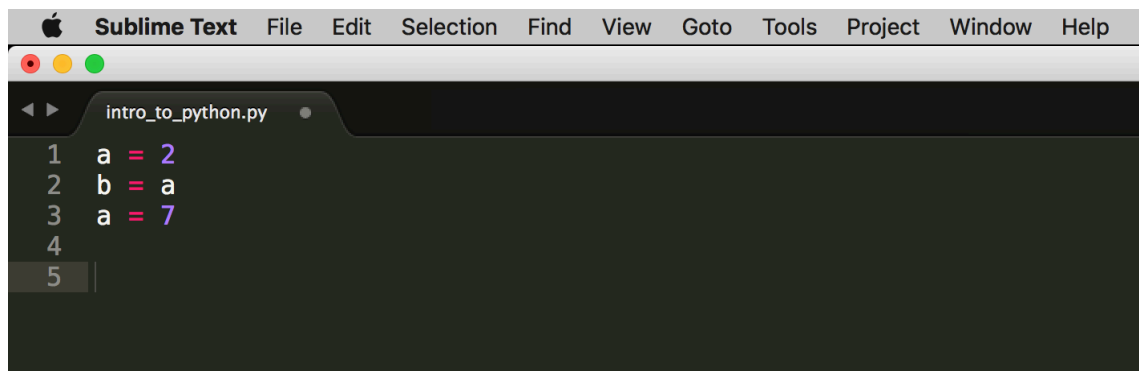
Great. We can now do stuff with data, but how to we keep a hold of their values for future use? Simple. Put them in variables. Hell, you can even put variables inside other variables! Below is an example of how to use them:



```
1 a = 5 % 3
2 a = a + 2
3 print(a)
4
5
```

Try and figure out what `a` is. `5 % 3` is 2, and `2 + 2` is 4. Ergo, `a` is 4. I demonstrated two other key concepts in this code snippet. 1) You can declare (define) a variable as many times as you want, and you can put anything inside them. 2) You can put variables inside other variables.

Here's a tricky example that tends to confuse a ton of beginners:



```
1 a = 2
2 b = a
3 a = 7
4
5
```

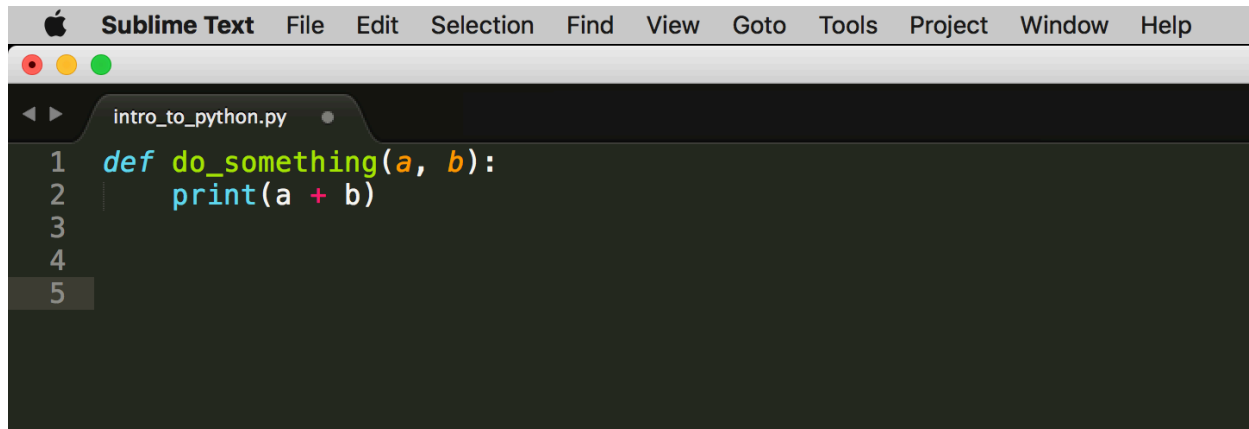
Try and figure out what `b` is. Is it 2? Is it 7? Is it the letter "a"?

It's 2.

Why? Let me explain it this way: in line 2, we put the **current value** of `a` inside of `b`.

Whatever happens to `a` after line 2 doesn't concern `b`. We never reassigned the value of `b`, so that means the value of `b` never changed. It's still 2.

We've seen examples of functions already (`print()` in particular), and we've worked with variables. Let's combine our knowledge of the two to create our own functions. The advantage of creating functions is simple: you don't want to keep repeating large chunks of code at multiple points in a program. What you can do *instead* is wrap that large chunk of code inside a function and call the function (as shown in chapter 0) whenever you need it. Here's a simple example of a function:

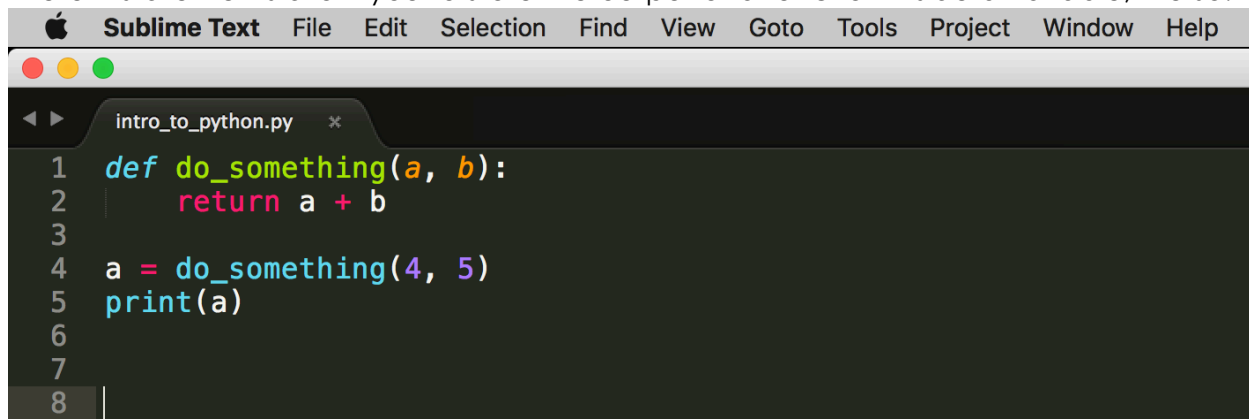


```
Sublime Text  File  Edit  Selection  Find  View  Goto  Tools  Project  Window  Help
intro_to_python.py
1  def do_something(a, b):
2      print(a + b)
3
4
5
```

As shown above, this is how you define a function: `def function_name(parameters):`, followed by whatever you want the function to do. **Note that all of the code inside of the function has to be indented, otherwise the compiler will think that nothing is inside of it.** The parameters, the stuff inside the parentheses, are temporary variables that you need to put into the parentheses when calling the function, separated by commas. Keep in mind that, when passing parameters, order matters. In the case of the above function, the first parameter corresponds to `a`, and the second parameter corresponds to `b`. These variables are initialized every time the function is called, and they *die* after the function code ends, so you won't have access to anything defined inside the function after the code ends. Unless, that is...

If you add a RETURN STATEMENT!!!

Return statements allow you to store the output of a function inside a variable, like so:

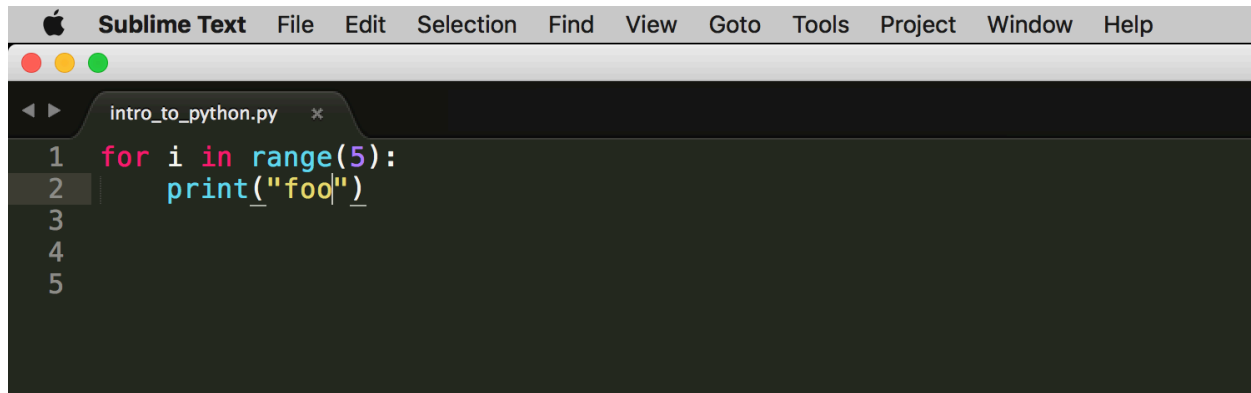


```
Sublime Text  File  Edit  Selection  Find  View  Goto  Tools  Project  Window  Help
intro_to_python.py x
1  def do_something(a, b):
2      return a + b
3
4  a = do_something(4, 5)
5  print(a)
6
7
8
```

In this case, `a` is equal to 9. Here's an interesting case, though. I just defined a variable `a` inside a function, and a variable `a` outside of the function. How are they different? Simple: their **scope** differs. The scope of `a` inside the function is only that function, which

means it only lives inside that function. The scope of the `a` outside of the function is, well, outside of the function. In short, scope determines where you can use a variable.

Finally, here's another way to prevent repeating code in a program. More specifically, this is a way to help you repeat the same few lines of code multiple times in the same place. This is called a **loop**. The body of a loop looks like this:



```
Sublime Text  File  Edit  Selection  Find  View  Goto  Tools  Project  Window  Help
intro_to_python.py
1  for i in range(5):
2      print("foo")
3
4
5
```

Line 1 declares a variable `i` whose scope is that loop. You can use any other variable if you really want. The `range(5)` part means that the line of code below will loop 5 times. You can substitute any number for 5, and any variable for `i`, and it won't make a difference. I'll go more in depth on loops and iterations in chapter 3, when we discuss lists and dictionaries.