01-Intro

Unknown Author

August 15, 2013

0.1 Python, IPython, and the basics + Codecademy Exercise 1

Loosely Based on Lecture Materials By: Milad Fatenejad, Katy Huff, Joshua R. Smith, Tommy Guy, Will Trimble, and Many More

0.2 Introduction

You've just done some basic programming in python using Codecademy. Now, we are going to use an environment called IPython notebook. You can click on anything in this notebook and edit it, but...

Don't panic!

You can't break anything! You can always get another copy of any notebook from the workshop on the Python FUNdamentals repository.

Reference materials:

For reference material, please start with the README, also at the Python FUNdamentals repository. Hopefully, this is all starting to sound pretty simple.

1 Getting Started

IPython has more useful features for interactive use than the standard python interpreter and Codecademy, but it works in the same basic way: you type things and then execute them. We'll use it from here on out.

Unlike code academy, there is no button to run your code! Instead, **you run code using Shift-Enter**. This also moves you to the next box (or "cell") for code below the one you just ran (but this may change in the future).

Try to **run the following code using Shift-Enter** now!

```
In [18]: print "Look Ma, I'm programming!"
```

```
Look Ma, I'm programming!
```

And how about this?

```
In [19]: print "Or am I
```

```
      File "<ipython-input-19-254da1df7681>", line 1
    print "Or am I
                 ^
SyntaxError: EOL while scanning string literal
```

As you can see, IPython offers proper error reporting. But don't worry - Python is just trying to help fix something it can't understand!

1.1 One more detail: Clearing IPython

IPython remembers everything it executed, **even if it's not currently displayed in the notebook**.

To clear everything from IPython use Kernel->Restart in the menu.

```
In [20]: mystring = "And three shall be the count."

         print mystring
```

```
And three shall be the count.
```

Now use Kernel->Restart in the menu!

```
In [21]: print mystring
```

```
And three shall be the count.
```

Note that the error message contains a recap of the input that caused the error (with an arrow, no less!) It is objecting that **mystring** is not defined, since we just reset it.

2 Codecademy Exercise 1

All programming languages have variables, and python is no different. To create a variable, just name it and set it with the equals sign. One important caveat: variable names can only contain letters, numbers, and the underscore character. Let's set a variable.

```
In [22]: experiment = "current vs. voltage"
```

```
In [23]: print experiment
```

```
current vs. voltage
```

```
In [24]: voltage = 2
```

```
In [25]: current = 0.5
```

In [26]:
```
print voltage, current
```

```
2 0.5
```

2.1 Types and Dynamic Typing

Like most programming languages, things in python are typed. The type refers to the type of data. We've already defined three different types of data in experiment, voltage, and current. The types are string, integer, and float. You can inspect the type of a variable by using the type command.

It's OK if you don't fully understand this part, and you shouldn't need to use `type` at all in your Codecademy exercises. But it's important stuff to really master Python!

In [27]:
```
type("current vs. voltage")
```

Out [27]:
```
str
```

In [28]:
```
type(2)
```

Out [28]:
```
int
```

In [29]:
```
type(0.5)
```

Out [29]:
```
float
```

In [30]:
```
type(experiment)
```

Out [30]:
```
str
```

In [31]:
```
type(voltage)
```

Out [31]:
```
int
```

In [32]:
```
type(current)
```

```
Out [32]:
          float
```

Python is a dynamically typed language (unlike, say, C++). If you know what that means, you may be feeling some fear and loathing right now. If you don't know what dynamic typing means, the next stuff may seem esoteric and pedantic. Its actually important, but its importance may not be clear to you until long after this class is over.

Dynamic typing means that you don't have to declare the type of a variable when you define it; python just figures it out based on how you are setting the variable. Lets say you set a variable. Sometime later you can just change the type of data assigned to a variable and python is perfectly happy about that. Since it won't be obvious until (possibly much) later why that's important, I'll let you marinate on that idea for a second.

Here's an example of dynamic typing. What's the type of data assigned to voltage?

```
In [33]: type(voltage)
```

```
Out [33]:
          int
```

Lets assign a value of 2.7 (which is clearly a float) to voltage. What happens to the type?

```
In [34]: voltage = 2.7
```

```
In [35]: type(voltage)
```

```
Out [35]:
          float
```

You can even now assign a string to the variable voltage and python would be happy to comply.

```
In [36]: voltage = "2.7 volts"
```

```
In [37]: type(voltage)
```

```
Out [37]:
          str
```

I'll let you ruminate on the pros and cons of this construction while I change the value of voltage back to an int:

```
In [38]: voltage = 2
```

2.2 Changing types: Coersion

It is possible to coerce (a fancy and slightly menacing way to say "convert") certain types of data to other types. For example, its pretty straightforward to coerce numerical data to strings.

```
In [39]: voltageString = str(voltage)
```

```
In [40]: currentString = str(current)
In [41]: voltageString
```

```
Out [41]:
          '2'
```

```
In [42]: type(voltageString)
```

```
Out [42]:
          str
```

As you might imagine, you can go the other way in certain cases. Lets say you had numerical data in a string.

```
In [43]: resistanceString = "4.0"
```

```
In [44]: resistance = float(resistanceString)
```

```
In [45]: resistance
```

```
Out [45]:
          4.0
```

```
In [46]: type(resistance)
```

```
Out [46]:
          float
```

What would happen if you tried to coerce resistanceString to an int? What about coercing resistance to an int? Consider the following:

```
In [47]: resistanceString = "4.0 ohms"
```

Do you think you can coerce that string to a numerical type?

2.3 GOTCHA: On Being Precise with floats and ints

Again, the following may seem esoteric and pedantic, but it is very important. So bear with me.

Let's say you had some voltage data that looks like the following

"

0

0.5

1

1.5

2

"

Obviously, if you just assigned this data individually to a variable, you'd end up with the following types

"

0 -> int

0.5 -> float

1 -> int

1.5 -> float

2 -> int

"

But what if you wanted all of that data to be floats on its way in? You could assign the variable and then coerce it to type float:

```
In [48]: voltage = float(1)
```

But that's ugly. If you want what is otherwise an integer to be a float, just add a period at the end

```
In [49]: voltage = 1.
```

```
In [50]: type(voltage)
```

```
Out [50]:
          float
```

This point becomes important when we start operating on data in the next section.

2.4 Data Operations

What's the point of data if we aren't going to do something with it? Let's get computing.

```
In [51]: a = 1
```

```
In [52]: b = 2
```

```
In [53]: c = a+b
```

```
In [54]: c
```

```
Out [54]:
          3
```

```
In [55]: type(a), type(b), type(c)
```

```
Out [55]:
          (int, int, int)
```

So we got a value of three for the sum, which also happens to be an integer. Any operation between two integers is another integer. Makes sense.

So what about the case where a is an integer and b is a float?

```
In [56]: a = 1
```

```
In [57]: b = 2.
```

```
In [58]: c = a + b
```

```
In [59]: c
```

```
Out [59]:
         3.0
```

```
In [60]: type(a), type(b), type(c)
```

```
Out [60]:
         (int, float, float)
```

You can do multiplication on numbers as well.

```
In [61]: a = 2
```

```
In [62]: b = 3
```

```
In [63]: c = a * b
```

```
In [64]: c
```

```
Out [64]:
         6
```

```
In [65]: type(a), type(b), type(c)
```

```
Out [65]:
         (int, int, int)
```

Also division.

```
In [66]: a = 1
```

```
In [67]: b = 2
```

```
In [68]: c = a / b
```

```
In [69]: c
```

```
Out [69]:
          0
```

ZING!

This is why type is important. Divding two integers returnes an integer: this operation calculates the quotient and floors the result to get the answer.

If everything was a float, the division is what you would expect.

```
In [70]: a = 1.
```

```
In [71]: b = 2.
```

```
In [72]: c = a / b
```

```
In [73]: c
```

```
Out [73]:
          0.5
```

```
In [74]: type(a), type(b), type(c)
```

```
Out [74]:
          (float, float, float)
```

This is actually “fixed” in python 3, and you can pull this change in like so. For a beginner, I’d recommend *always* putting the following statement at the beginning of your scripts where you’re doing a lot of math! (It’ll make more sense later)

```
In [75]: from __future__ import division
```

Now we’ll get a different kind of answer from division, even if both numbers are integers

```
In [76]: 1 / 2
```

```
Out [76]:
          0.5
```