Python Crash Course

Jordan Boyd-Graber

October 14, 2008

Python is a very easy language to learn, and it has a wonderful collection of libraries that seem to do everything (as implied by the XKCD comic). In this document, we're going to run through the basics of Python and then run through an introduction to NLTK written by Nitin Madnani.

1 How to Interact with Python

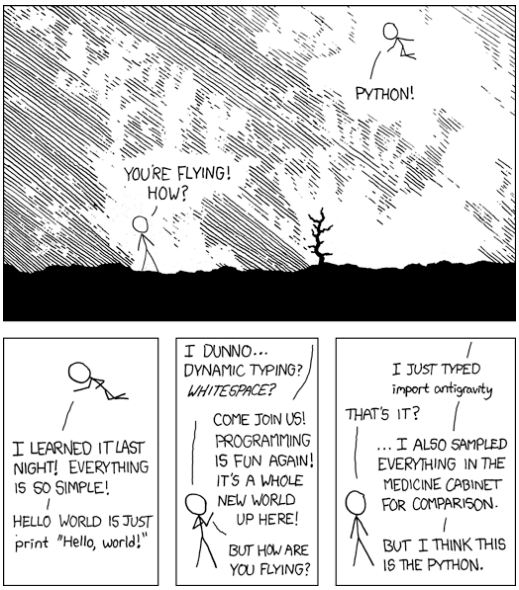
If Python is installed on your computer (and it's in your path), go to the command line and type "python". (Windows users: you might have to specify the full path, as in "c:\Python25\python.exe".)

```
dynamic-oit-vapornet-b-1226:~ jbg$ python
Python 2.5.2 (r252:60911, Feb 22 2008, 07:57:53)
[GCC 4.0.1 (Apple Computer, Inc. build 5363)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello, world!"
Hello, world!
```

Once you see the Python prompt, you can start typing. Alternatively, you can specify a Python program as the input to the Python program, and it run everything written in the Python program. When you type anything on this prompt, Python immediately interprets it and spits out the resulting value. This allows us to write programs really quickly and get instant feedback.

2 Datatypes

The primary data types you need to know in Python are integers, floats, strings, lists, and dictionaries.



int Integers are counting numbers, positive and negative. Note that when you divide one integer by another, you always get an integer. This is often a problem when you're computing probabilities.

```
>>> 22/7
3
```

float Floats are numbers that can be expressed as a fraction.

```
>>> float(22) / float(7)
3.1428571428571428
```

Note that I could have written this as 22.0 / 7.0 and still gotten the right answer, as 22.0 cannot be an integer. The operator "float" allows me to convert from an integer to a float. In general, using the name of a data type as an operator on a data value, allows you to convert one type of data into another. This is also how functions are called.

string We're going to work quite a bit with strings. A string is simply a bunch of characters. Whatever you can type on your keyboard. The nice thing about python is that every string object has a wide range of functions built in:

```
>>> s = " I am the very model of a modern Major-General "
>>> s.strip()
'I am the very model of a modern Major-General'
>>> s.find("am")
3
>>> s.replace("modern Major-General", "Gilbert caricature")
' I am the very model of a Gilbert caricature '
```

Notice that we also used assignment in order to make "s" mean the string that we specified. Instead of writing that whole long string each time, we can just write "s" instead.

list A list is an *ordered* collection of data (of any type). A string is very much like a list with extra functions thrown in. Both lists and strings can access using the accessor []. If we turn a string into a list, we just get a list of all the characters in the string, but there are better ways to turn a string into a list (especially via NLTK).

```
>>> l = range(10)
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> l[4]
4
>>> list(s)
[' ', 'I', ' ', 'a', 'm', ' ', 't', 'h', 'e', ' ', 'v', 'e', 'r', 'y', ' ', 'm', 'o', 'd', 'e', 'l', ' ', 'o'
>>> s.split(" ")
['', 'I', 'am', 'the', 'very', 'model', 'of', 'a', 'modern', 'Major-General', '']
>>> "am" in s.split()
True
>>> "Mikado" in s.split()
False
>>> filter(lambda x: x != "", s.split(" "))
['I', 'am', 'the', 'very', 'model', 'of', 'a', 'modern', 'Major-General']
```

We did a whole lot there. First, the "range" function allows you to generate a list of integers. With one argument, it gives you all non-negative integers less than that argument. The "split" function of a string allows you to break apart a string into a list where the elements were separated by the argument of the split function in the original string. You can also ask if something is in a list (or a string) by using the keyword "in". The "filter" function is pretty advanced, and I won't explain it here, but I'm including it here for anyone from a functional background to know that you can do it in python.

dict Dictionaries are the awesomest thing in Python. This is basically a hash table at the very lowest level of the language. A dictionary stores a mapping between keys and values.

```
>>> d = {}
>>> d[3] = 4
>>> d[3]
4
>>> d[2]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
>>> d[3] = 2
>>> d[3]
2
```

You can have anything that's hashable as a key, and anything as a value. Note that if you put in a value for the same key twice, it will overwrite the contents.

3 Control

This brings us to one of the quirkiest features of Python: syntactic whitespace. You don't tell your program what is part of a loop or a block of your code by using brackets, as in other languages. You use whitespace. Everything inside an if statement has to be at the same level of indentation. You tell Python that you're done by returning to the earlier indentation level.

```
>>> sheeps_clothing = "wool"
>>> if "wolf" in sheeps_clothing:
...     print "RUN"
... elif len(sheeps_clothing) > 4:
...     print "Haircut time"
... else:
...     print "All is well"
...
All is well
```

If blocks are only required to have an "if", and everything else is optional. The end of each control line has a colon. For loops are also pretty straightforward; they iterate over the elements of a list:

```
>>> sum = 0
>>> for i in range(100):
...     sum += i
...
>>> print sum
4950
```

(Remember that range(X) generates a list of all the numbers less than the argument; a more memory efficient version of range for loops is "xrange", but this likely won't be a problem for a while.)

4 Functions you make yourself and functions you borrow

```
>>> import nltk, re
>>> sent = "The quick brown fox jumped over the lazy dog"
>>> nltk.re_show("(fox|dog|marzipan)", sent)
The quick brown {fox} jumped over the lazy {dog}
>>> re.findall("(fox|dog|rat)", sent)
['fox', 'dog']
```

We use the "from X import Y" construction to bring in code that other people have written. Python looks for this code in all of the places listed in the environment variable "PYTHON_PATH"; it will search subdirectories (as specified by the "." so long as there is a "__init__.py" file. Now we'll import more code from NLTK.

```
>>> from nltk.tokenize import WordPunctTokenizer
>>> from nltk.stem import PorterStemmer
>>>
>>> tokenizer = WordPunctTokenizer()
>>> porter = PorterStemmer()
>>>
... tokenize = tokenizer.tokenize
>>> stem = porter.stem
>>>
>>> def tokens(raw_text):
...     tokens = map(stem, tokenize(raw_text))
...     return tokens
...
>>> tokens("Mares eat oats, and does eat oats, and little lambs eat ivy; A kid'll eat ivy too, wouldn't you?")
['Mare', 'eat', 'oat', ',', 'and', 'doe', 'eat', 'oat', ',', 'and', 'littl', 'lamb', 'eat', 'ivi', ';', 'A', 'kid',
```

We create two instances of a tokenizer and a stemmer, and then use their "tokenize" and "stem" functions to build a function of our own.

We create a new function called "tokens" which takes all the tokens in a string and then stems them. "def" is the keyword used to let Python know we're defining a new function. The parentheses tell the function takes a single argument. We then write all the code that we want to run every time the function "tokens" is called. Note that the return keyword ends the execution of a function; if you don't specify a return value, it returns "None" (a special value in Python).

5 Pointers to other material

1. http://nltk.org/doc/api/

2. http://www.umiacs.umd.edu/ nmadnani/pdf/crossroads.pdf

3. http://openbookproject.net//thinkCSpy/

4. http://docs.python.org/tut/