Python for Scientific Data Analysis

Basic Python

Section 1: Printing and Variables

Hello World

The first step in learning Python or any other language is to ask a piece of code to print a statement.

To do this from interactive Python ("iPython"), first you type `python` in your terminal window, which will bring up the interactive Python prompt.

Printing with statements with Python 3 are enclose by parentheses.

E.g.

```
>>> print("hello world")
```

will print

```
>>> hello world
```

Note: if you do not enclose this string with ", then it will trigger a traceback error because *hello world* is not a named variable. If you try to treat *hello world* as a variable you will get a traceback error: e.g. `>>> hello world = 5` triggers an error. However, the following does not trigger an error:

`>>> hello_world=7` `>>> hello_world='yo'` `>>> hello, world = 7,11` `>>> hello, world = 'yo','checkit'`,

where then you type `print(hello_world)` or `print(hello,world)` at the prompt.

What happened in these four examples? In the first two, we declared a single variable,

`hello_world`, as an integer 7 or a string `yo`. But Python allows you to declare multiple variables in a single line. This is shown in third and fourth example: here, `hello` and `world` are two separate variables simultaneously set to two different strings. Cool, right?

The example code ex1.py shows demonstrates how this looks within a piece of code. To execute, type `python ex1.py` in your Terminal window. You should see the following:

```
hello world
yo
5
yo checkit
7 11
```

Variables and Printing

Okay, now we are going to make things slightly more complicated. When you print something with Python (or any other language), you want to control the format of what you write: do you want to write a string? An integer? decimal? floating point? That's where *formatted* print comes into play.

Formatted Printing: The standard way

Python's *formatted* print statements come after a `%` for a print statement: e.g. `print("my name is %[formatting statement]" % [variable]`).

Fortunately, these are pretty easy to figure out: `%s` (e.g. "yo") for string, `%d` or `%i` for decimal integers (e.g. 5), `%f` for floating point (e.g. 3.141569), `%e` for lower case exponential notation (e.g. 1.41569e+03), etc. There are a few others but these are the most important ones for now.

Now, look at ex5a.py. It should read as follows

```
my_name = 'The Great Cornholio'
my_age = 21 # ya right
my_height = 72 # with stilts
my_weight = 9 # lbs
my_jump = 15 #feet
my_eyes = 'White'

print("My name is %s." % my_name)
print("I am %d inches tall and I weigh %d pounds." % (my_height,my_weight
))
print("Even though my eyes are %s, I can jump %d feet in the air" % (my_e
yes,my_jump))
print("If you add %d, %d, and %d together you get %d.  Scared now of the
%s?" % (my_age,my_height,my_jump,my_age+my_height+my_jump,my_name))
```

If you execute this code, you should see:

```
My name is The Great Cornholio.
I am 72 inches tall and I weigh 9 pounds.
Even though my eyes are White, I can jump 15 feet in the air
If you add 21, 72, and 15 together you get 108.  Scared now of the The Gr
eat Cornholio?
```

Notice another thing. In the soure code we put #s all over the place. These look like comments, right? That they are! These do not print out when you execute the code.

Formatted Print Statements: The second, more 'modern' way

Another way to do formatted print statements are with square brackets `{ }` and a format statement. Here's an example

```
statement='far out'
print("this statement is {}, dude".format(statement))
```

You should see `this statement is far out, dude`.

Now, slightly more complicated

```
print("this statement is {0} {1}, dude".format(statement,'seriously'))`
```

See what we've done here? we now have two variables with formatted print.

More control over formatting

Now we will take the second formatting approach and make this more complicated ...

```
 print("this statement is {0:.1s} {1:s}, dude".format(statement,'seriousl
y'))
```

which prints out `this statement is f seriously, dude` . See what happened? if not, look to see what happens here:

```
import numpy as np
eval=np.e
print("this stupid number is {0:f}, not {1:.3f} or {2:.5f}, \
but if we want more than the \
 default precision of {3:d} spaces after the decimal, \
 we can write it as something different like {4:.10f}".format(eval,eval,e
val,6,eval))
```

which prints out as `this stupid number is 2.718282, not 2.718 or 2.71828, but if we want more than the default precision of 6 spaces after the decimal, we can write it as something different like 2.7182818285`

Note: all of this is on one line: the `\` signifies "continue the line".

So what happened there? we can control the output. For example, in the first case, we said "format this as a floating point". In the second "format this as a floating point but only leave 3 spaces after the decimal". And so on. By default, Python 3.x prints out 6 decimal spaces for a floating point. If you want more precision, then you can just add more space, as we do in the last formatted print for the variable `eval` .

Now what if we wanted to use `%` for formatted print with all that control?. Here you go:

```
print("this stupid number is %f, not %.3f or %.5f, \
but if we want more than the \
 default precision of %d spaces after the decimal, \
 we can write it as something different like %.10f" %(eval,eval,eval,6,ev
al))
```

A Note on how to `run' Python code

In general, to execute Python code, you either do what we just did above and 1) type

`python [name of python code]` in your Terminal if the code is also in your local working directory. Or 2) in interactive mode where you have to import a module from a piece of Python code and then do something: this works so long as the piece of code is in your Python path (either local or within your path variable). A third way is accessing Python code through Jupyter notebooks: we will for go that for now. We will learn Python through route #1 first and then introduce the more complicated route #2 later.

For more details, see this nice discussion of the Python interpreter on the official site, here: https://docs.python.org/3/tutorial/interpreter.html