Python 1

Python Overview

Python is a scripting language. It is useful for writing medium-sized scientific coding projects. When you run a python script, the python program will generate byte code and interpret the byte code. This happens automatically and you don't have to worry about it. Compiled languages like C, C++ will run much faster, but are much much more complicated to program. Languages like java (which also gets compiled into byte code) are well suited to very large collaborative programming projects, but don't run as fast as C and are more complex that python.

Python has

- data types
- functions
- objects
- classes
- methods

Data types are just different types of data which are discussed in more detail later. Examples of data types are integer numbers and strings of letters and numbers (text). These can be stored in variables.

Funtions do something with data, such as a calculation. Some functions are already built into Python. You can create your own functions as well.

Objects are a way of grouping a set of data and functions (methods) that act on that data

Classes are a way to encapulate (organize) variables and functions. Objects get their variables and methods from the class they belong to.

Methods are just functions that belong to a Class. Objects that belong to the a Class can use Methods from that Class.

Running Python

There are two versions of python: python 2 and python 3. We will be using 3. This version fixes some of the problems with python 2 and breaks some other things. A lot of code has already been written for python 2 (it's older), but going forwards, more and more new code development will use python 3.

Interactive Interpreter

Python can be run one line at a time in an interactive interpreter. You can think of this as a python shell. To lauch the interpreter type the following into your terminal window:

`$ python3`

Note: '$' indicates the command line prompt. Recall from Unix 1 that every computer can have a different prompt!

First Python Commands:

```
>>> print("Hello, PFB2017!")
Hello, PFB2017!
```

> Note: `print` is a function. Function names are followed by (), so formally, the function is `print()`

Python Scripts

- The same code from above is typed into a text file using a text editor.
- Python scripts are always saved in files whose names have the extension '.py' (i.e. the filename ends with '.py').

File Contents:

```
print ("Hello, PFB2017!")
```

Running Python Scripts

Typing the python command followed by the name of a script makes python execute the script. Recall that we just saw you can run an interactive interpreter by just typing `python` on the command line

Execute the Python script like this (% represents the prompt)

```
% python3 test.py
```

This produces the following result:

```
Hello, PFB2017!
```

A quicker/better way to run python scripts

If you make your script exectuable, you can run it without typing `python3` first. Use `chmod` to change the permissions on the script like this

`chmod +x test.py`

You can look at the permissions with

```
% ls -l test.py
-rwxr-xr-x  1 sprochnik  staff  60 Oct 16 14:29 test.py
```

The first field of -, r, w and x characters define the permissions of the file. The three 'x' characters means anyone can execute or run this script.

We also need to add a line at the beginning of the script that tells the shell to run python3 to interpret the script. This line starts with #, so it looks like a comment to python. The '!' is important as is the space between env and python3. The program /usr/bin/env looks for where python3 is installed and runs the script with python3. The details may seem a bit complex, but you can just copy and paste this 'magic' line.

The file test.py now looks like this

```
#!/usr/bin/env python3
print ("Hello, PFB2017!")
```

Now you can simply type the name of the script to run it. Like this

```
% ./test.py
Hello, PFB2017!
```

Syntax

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, seq_id and seq_ID are two different identifiers in Python.

Naming conventions for Python Identifiers

- The first character is lowercase, unless it is a name of a Class. Classes should begin with an uppercase characters. (ex. Seq)
- Private identifiers begin with an underscore. (ex. `_private`)
- Strong private identifiers begin with two underscores. (ex. `__private`)
- Language-defined special names begin and end with two underscores. (ex. `__special__`)

Reserved Words

The following is a list of Python keywords. These are special words that already have a purpose in python and therefore cannot be used as indentifier names.

```
and         exec        not
as          finally     or
assert      for         pass
break       from        print
class       global      raise
continue    if          return
def         import      try
del         in          while
elif        is          with
else        lambda      yield
except
```

Lines and Indentation

Python denotes blocks of code by line indentation. Incorrect line spacing and/or indention will cause an error or could make your code run in a way you don't expect. You can get help with indentation from good text editors or Interactive Development Environments (IDEs).

The number of spaces in the indentation need to be consistent but a specific number is not required. All lines of code, or statements, within a single block must be indented in the same way. For example:

```
#!/usr/bin/env python3
for x in (1,2,3,4,5):
    if x > 4:
        print("Hello")
    else:
        print(x)
print('All Done!')
```

Comments

Comments are an essential programming practice. Making a note of what a line or block of code is doing will help the writer and readers of the code. This includes you!

Comments start with a pound or hash symbol `#`. All characters after this symbol, up to the end of the line are part of the comment and are ignored by python.

The first line of a script starting with `#!` is a special example of a comment that also has the special function in unix of telling the unix shell how to run the script.

```
#!/usr/bin/env python3

# this is my first script
print ("Hello, PFB2017!") # this line prints output to the screen
```

Blank Lines

Blank lines are also important for increasing the readability of the code. You should separate pieces of code that go together with a blank line to make 'paragraphs' of code. Blank lines are ignored by the python interpreptor

Python Options

```
$ python3 -h
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit
```

Data Types and Variables

This is our first look at variables and data types. Each data type will be discussed in more detail in subsequent sections.

The first concept to consider is that python data types are either immutable (unchangeable) or not. Literal numbers, strings and tuples cannot be changed. Lists, dictionaries and sets can be changed. So can individual (scalar) variables. You can store data in memory by putting it in different kinds variables. You use the `=` sign to assign a value to a variable.

Numbers and Strings

Numbers and strings are two common data types. Literal numbers and strings like this `5` or `'my name is'` are immutable. However, their values can be stored in variables and then changed.

For Example:

```
first_variable = 5
first_variable = 10
```

Different types of data can be assigned to variables, i.e., integers (1,2,3), floats (floating point numbers, 3.1415), and strings (text).

For Example:

```
count   = 10      # this is an integer
average = 2.5     # this is a float
message = "Welcome to python" # this is a string
```

10, 2.5, and "Welcome to python" are singular pieces of data being stored in an indivudual variables.

Collections of data can also be stored in special data types, i.e., tuples, lists, sets, and dictionaries.

Lists

- Lists are used to store an ordered, *indexed* collection of data.
- Lists are mutable: the number of elements in the list and what's stored in each element can change
- Lists are enclosed in square brackets and items are separated by commas

```
[ 'atg' , 'aaa' , 'agg' ]
```

Index	**Value**
0	atg
1	aaa
2	agg

> The list index starts at 0

Command line parameters: A Special Built-in List

Command line parameters follow the name of a script or program and have spaces between them. They allow a user to pass information to a script on the command line when that script is being run. Python stores all the pieces of the command line in a special list called sys.argv.

You need to import the sys module at the beginning of your script like this

```
import sys
```

If you write this on the command line:

```
$ calculate_sum.py 5 7
```

This happens inside the script:

> the script name, and the numbers 5 and 7 are contained in a list called `sys.argv`.

> These are the command line parameters, or arguments you want to pass to your script.
> `sys.argv[0]` is the script name.
> You can access values of the other parameters by their indices, starting with 1, so `sys.argv[1]` is 5 and `sys.argv[2]` is 7.
> If you wanted to calculate the sum in your script, you would add these two variables and print the result. Maybe your code would look something like this

```
#!/usr/bin/env python3
import sys
a = float(sys.argv[1]) # convert string from command line to float
b = float(sys.argv[2]) # convert string from command line to float
print(a+b) # + is a sum operator on integers
```

Tuple

- Tuples are similar to lists and contain ordered, *indexed* collection of data.
- **Tuples are immutable: you can't change the values or the number of values**
- A tuple is enclosed in parentheses and items are separated by commas.

```
( 'Jan' , 'Feb' , 'Mar' , 'Apr' , 'May' , 'Jun' , 'Jul' , 'Aug' , 'Sep' , 'Oct' ,
'Nov' , 'Dec' )
```

Index	Value
0	Jan
1	Feb
2	Mar
3	Apr
4	May
5	Jun
6	Jul
7	Aug
8	Sep
9	Oct
10	Nov
11	Dec

Dictionary

- Dictionaries are good for storing data that can be represented as a two-column table.
- They store unordered collections of key/value pairs.
- A dictionary is enclosed in curly braces. and sets of Key/Value pairs are separated by commas
- A colon is written between each key and value. Commas separate key:value pairs.

```
{ 'TP53' : 'GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC' , 'BRCA1' : 'GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA' }
```

Key	Value
TP53	GATGGGATTGGGGTTTTCCCCTCCCATGTGCTCAAGACTGGCGCTAAAAGTTTTGAGCTTCTCAAAAGTC
BRCA1	GTACCTTGATTTCGTATTCTGAGAGGCTGCTGCTTAGCGGTAGCCCCTTGGTTTCCGTGGCAACGGAAAA

What kind of object am I working with?

You have an identifier in your code called `data`. Does it represent a string or a list or a dictionary? Python has a couple of functions that help you figure this out.

| Function | Description | | `type(data)` | tells you which class your object belongs to | | `dir(data)` | tells you which methods are available for your object |

We'll cover `dir()` in more detail later

```
>>> data = [2,4,6]
>>> type(data)
<class 'list'>
>>> data = 5
>>> type(data)
<class 'int'>
```

Link to Python 1 Problem Set
