Python 1: Objects and Operators

This week we'll start to program in Python. Python is very 'high level,' which means it has a ton functionality built into it that helps you code less and more readably. The downside of this is that you need to learn a larger list of items to explore python efficiently.

Python is "object oriented." This means that it is heavily involved with the way data is structured and represented. It supports many built-in structures and allows you build your own easily.

Our first task is therefore to understand what an object is or can be, and how this is different from other elements of the programming language.

References

Python Standard Library: http://docs.python.org/2/library/index.html
The Python Language Reference: http://docs.python.org/2/reference/
Stack Overflow: http://stackoverflow.com/ (Warning: can be snarky, but very helpful)

Objects (Ch. 3 of Python Language Reference): An abstraction of data.

Every object has an identity, a type and a value

Identity: Constant, like a computer memory address
Type: Constant, determines supported operations of the object
Value: May change. If an object's values can change, it is 'mutable'

Since the type determines what you can do with an object, it is important to know what types are built into Python. We have included a hierarchical list of common Python types in the type_hierarchy handout. We will only be dealing with a few of these this week: Integers, Floats, Strings, and Lists.

Type Name	Description	Conversion	Assignment
Integers	Whole numbers,	`int()`	`int = 2`
Floats	Floating point numbers,	`float()`	`flt = 2.0`
Strings	Ordered, immutable character set,	`str()`	`string = "hello"`
Lists	Ordered, mutable object container,	`list()`	`list = [2,2.0, "hello"]`

Common String Methods

```
.upper() # capitalize string
.lower() # make it lower case
.split(char) # split on char (a str) and return a list
.strip(char) # remove leading and trailing chars.
```

Common List Methods

```
.append()
.insert()
```

`.remove()`

Operators

Operators are like verbs. They are pretty easy to conceptualize since their meaning in python is not really different from their meaning in math.

Common Operators

+	Addition	`3+4=7`
-	Subtraction	`3-4=1`
*	Multiplication	`4*3=12`
/	Division	`12/4 = 3`
%	Modulus	`4%3 = 1`
**	Exponent	`4**3 = 64`
==	Equals	`>>>3==4` `False`
!=	Not equals	`>>>3!=4` `True`
>	Greater	`>>>3>4` `False`
<	Less	`>>>3<4` `True`
>=	Greater than or equal to	`>>>3>=4` `False`
<=	Less than or equal to	`>>>3<=4` `True`

The Python interpreter

Type 'python' at the command line. As long as python is installed, this will open the python interpreter, which is a command line-like python environment. The prompt looks like this: >>>. Type Ctrl+D to quit.

You can access documentation on most object by typing `help(object)`. These aren't always that helpful, though.

`For` Loops

Control the flow of your programs by iterating a command over a collection of objects. Syntax:

```
for item in collection:
   do something with item
   do something else
```

"item" is automatically created as a variable by the loop, you don't have to declare it anywhere else.

`While` Loops

Used less often than `for` loops because they're not as readable and clear. But they're good for some things, which we may get to later. Syntax:

```
while condition:
    do something
```

Note that if *condition* is always `True`, the loop goes on forever – not always a disaster, actually.

List and string slicing

This bread-and-butter capability allows you to select parts of *ordered* objects, like strings and lists. It takes a little getting used to, unfortunately, so make sure you get some practice. Notation:

```
collection[start:stop]
```

The first tricky thing is that computers start counting from 0, not 1.

```
>>> string1 = "Ben"
>>> string1[0]
'B'
```

The next oddity is that the starting position is *inclusive* and the stopping position is *exclusive*. So the stopping number is the first position which is *not* included in the slice.

```
>>> string1[0:1]
'B'
>>> string1[0:2]
'Be'
```

List and string occupancy

Python can check to see if characters or objects are in lists and strings. It will return a Boolean type.

```
>>> mystring = "Rocky 4"
>>> "4" in mystring
True
```

```
>>> 4 in mystring
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'in <string>' requires string as left operand

>>> mylist = ["Rocky", 4]
>>> "4" in mylist
False
>>> 4 in mylist
True
>>> print mylist[0], mylist[1], "the best movie"
Rocky 4 the best movie
```