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." I far as we can figure it, 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/

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()
.lower()
.split()
.strip()

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.

"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. It has global scope, and is reassigned after each iteration of the loop. After the loop, its value is its value on the last iteration.

Working with Files

Files are opened and closed with the (you guessed it!) open() and close() methods. This opens a "file buffer," which can write to if you can write to if you give yourself permission. You can write with…write()

```
>>> file1 = open("my_data.txt", "r")  # "r" gives you read permission
>>> list = []
>>> for line in file1:
...     do super sweet transformations to each line
...     list.append(line)
>>> file2 = open("my_supersweet_data.txt","w")  # write permissions!
>>> for new_data in list:
...     file2.write(new_data)
>>> file1.close()  # remember to close file buffers
>>>file2.close()
```

Fancy Tricks

Multiple assignment:

```
>>> my_string, my_integer = "Rocky", 4
>>> my_string
'Rocky'
>>> my_integer
4
```

String Concatenation:

```
>>> str1 = "hello"
>>> str2 = "world"
>>> str3 = str1 + "," + str2
>>> str3
'hello, world'
```

How not to forget to close files – the "with" methods

```
>>> with open("file1.txt") as f:
...     do something with f
```

Assignments in the "with" block have global scope, but the file will automatically close after the block is fully executed. Cool!