Introduction to Programming for Scientists

Lecture 1

Introduction
Datatypes

Programming

- What is and isn't programming ?

Language History

8512 documented lanuages (vs. 2376)

- Four of the first modern languages (50s):
  - FORTRAN (FORmula TRANslator)
  - LISP (LISt Processor)
  - ALGOL
  - COBOL (COmmon Business Oriented Language)
- C (1972)
- C++ (1983)
- Perl (1990)
- Python (1991)
- Ruby (1992)
- HTML (1994)
- Java (1995)

Programming

- Programming Languages
  - Small vocabulary
  - Standard constructs across languages
  - Simple, exactly specified syntax
  - Datatypes

Python ?

PYTHON OOL- developed by Guido van Rossum, and named after Monty Python.(No one Expects the Inquisition) a simple high-level interpreted language. Combines ideas from ABC, C, Modula-3, and ICON. It bridges the gap between C and shell programming, making it suitable for rapid prototyping or as an extension of C. Rossum wanted to correct some of the ABC problems and keep the best features. At the time, he was working on the AMOEBA distributed OS group, and was looking for a scripting language with a syntax like ABC but with the access to the AMOEBA system calls, so he decided to create a language that was extensible; it is OO and supports packages, modules, classes, user-defined exceptions, a good C interface, dynamic loading of C modules and has no arbritrary restrictions.

www.python.org

Hello World

- Python

```
print “Hello, world!”
```

- Perl

```
print "Hello, world!\n";
```

- C:

```
#include <stdio.h>

main() {
    printf(“Hello, world!\n”);
    exit(0);
}
```

The first program you learn in any programming language.

Install Python

Note that these are specific to version 2.4.3, in the future, you may want to just go to www.python.org and follow the links...

Windows

browse to:
http://www.python.org/download/releases/2.4.3/
follow instructions

Linux

probably already installed

OSX

browse to:
http://www.python.org/download/releases/2.4.3/
follow instructions, or you can use 'fink'

Python as a Calculator

> *python* (under windows, run 'Idle' from the start menu)

....

IDLE 1.0.2

>>> ***5*10*** <-- Note that bold italics indicate what you should type at the prompt

50

>>> ***5*10/3***

16

>>> ***5.0*10/3***

16.666666666666668

>>> ***5**2***

25

>>> ***5**3***

125

>>> ***sqrt(4)***

Traceback (most recent call last):

  File "<pyshell#5>", line 1, in -toplevel-

    sqrt(4)

NameError: name 'sqrt' is not defined

Python as a Calculator

```
>>> import math          <-- before we use special math functions, we need to 'import' the
>>> math.sqrt(4)               math library
2.0
>>> math.sqrt(-1)        <-- normal math library does not support imaginary numbers
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in -toplevel-
    math.sqrt(-1)
ValueError: math domain error

>>> import cmath         <-- cmath stands for 'complex math' and supports complex numbers
>>> cmath.sqrt(-1)
1j

>>> from math import *   <-- the '*' is a wildcard meaning read everything, after doing
>>> pi                         this, all math operations are available without 'math.'
3.1415926535897931
>>> sin(pi/2)
1.0
```

Variables

```
>>> a=5                 <--- An integer
>>> b=4.0               <--- A floating point number
>>> a*b
20.0
>>> y=m*x+b             <--- Not a function, just a 1 time calculation, so all variables must exist
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in -toplevel-
    y=m*x+b
NameError: name 'm' is not defined

>>> m=1
>>> x=2
>>> y=m*x+b
>>> y
6.0
>>> x=3
>>> y
6.0                     <--- Since y=m*x+b was a 1-time calculation, y does not change when x changes
>>> y=m*x+b
>>> y
7.0
```

Your Own Functions

```
>>> def y(m,x,b):                <-- Now we define it as an actual function we can re-use
        return m*x+b

>>> y(1,2,3)                     <-- Call the function with m=1, x=2 and b=3
5
>>> f(1.0,2,3)
5.0
>>> def g(x,y):
        return float(x)**int(y)          <-- '**' means raise to the power of in python
>>> g(5.0,3)
125.0
>>> g(5,3.0)
(what do you think?)

>>> cos(pi)
-1.0
>>> def cos(x): return x+1.0
>>> cos(pi)
4.1415926535897931
```

Simple Strings

```
>>> "Hello there"                <--- A simple string
'Hello there'
>>> 'Hello there'                <--- Single quotes equivalent to double
'Hello there'
>>> """Hello There"""            <--- Triple-double-quotes let you span multiple lines
'Hello There'
>>> "'Hello' there"
"'Hello' there"
>>> print "'Hello' there"
'Hello' there
>>> print """This is a
multiline string, denoted by
triple quotes"""
This is a
multiline string, denoted by
triple quotes
```

String Math

```
>>> "This"+" is "+"a test"          <-- Adding strings concatenates
'This is a test'
>>> "5+2="+(5+2)                    <-- Cannot add strings and numbers
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in -toplevel-
    "5+2="+(5+2)
TypeError: cannot concatenate 'str' and 'int' objects

>>> "5+2="+str(5+2)                 <-- First convert the number to a string
'5+2=7'
>>> "abc"*3              <-- Multiplication makes copies of strings
'abcabcabc'
>>> str(5+2)*3
(what do you think?)
```

String Slicing and Dicing

```
>>> "Hello there world"
'Hello there world'
>>> “Hello there world”[1]          <-- Returns element number 1 from the string
'e'                                      but the first element is 0, so 1 is the second
>>> "Hello there world"[6:11]       <-- A 'slice of the string, the 7th thru 11th elements
'there'                                  the last element (number 11) is never included
>>> "Hello there world"[:5]         <-- Starts at the beginning if the first number
'Hello'                                  is missing
>>> "Hello there world"[-5:]        <-- Negative values count from the END of the
'world'                                  string, if the 2nd number is missing, go to end
```

```
          1111111
0123456789012345          <-- counting from the beginning
Hello there world
        987654321         <-- counting from the end
```

Homework

Practice on your own:

- Install Python on your machine. Also, here are a couple of quick tests you can try yourself to see if you understood all of the material today. Try to figure out what the results will be, then enter them in python and check yourself. They may be tricky, so do check yourself even if you think you know the answer:

1)

```
a="18"
b="5"
int(a+b)/3
```

2)

```
1/2+1.0/2
```

3)

```
int('9'*3)+1
```