Introduction to Python

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

If you have your own Win PC, install Python 2.6.9 and a syntax-highlighting text editor:

http://www.flos-freeware.ch/notepad2.html

http://www.python.org/download/releases/2.6.9/

If you have your own Mac, install Python (same site) and TextWrangler:

http://www.barebones.com/products/TextWrangler/download.html

This version of Python is installed on the lab computers, any Python 2.6 or 2.7 release should work fine.

Why Python?

- Python is
  - easy to learn
  - fast enough
  - object-oriented
  - widely used
  - fairly portable
- C and C++ are much faster but much harder to learn and use.
- Java is somewhat faster but harder to learn and use.
- Perl is harder to learn.

Getting started on the Mac

- Start a terminal session
- Type "python"
- This should start the <u>python interpreter</u> (often called "IDLE")
- Print "Hello, world!" as below

```
> python
Python 2.6.4 (something something)
details something something
Type "help", "copyright", "credits" or "license"
for more information.
>>> print "Hello, world!"
Hello, world!
```

if the prompt is '>>>' you are in the interpreter

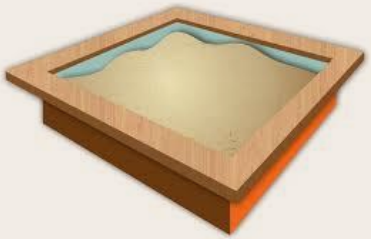
to leave the interpreter, type **Ctrl-D** or **exit()**

The interpreter

- Try printing various things (in your spare time)
  - Leave off the quotation marks.
  - Print numbers, letters and combinations.
  - Print two things, with a comma between them.
  - Enter a mathematical formula.

- Use the interpreter to test syntax, to try new commands, etc. <u>Don't write programs in the interpreter</u>.

the Python interpreter is a sandbox:
you play in it, you don't work in it



Your first program

- In your terminal, Ctrl-D out of the python interpreter.
- Type "pwd" to find your present working directory.
- Open TextWrangler.
- Create a file containing one line:

```
print "Hello, world!"
```

- Be sure that you end the line with enter.
- Save the file as "hello.py" in your present working directory.
- Back in your terminal, type "python hello.py"

```
> python hello.py
Hello, world!
```

(This tells the computer "use python to run the program hello.py". Yes, the result is somewhat anticlimactic.)

Notice that, once you save the file with ".py" as the extension, WordWrangler automatically colors the text according to the syntax.

Objects and types

- An object refers to any entity in a python program.
- Every object has a type, which determines the properties of the object.
- Python defines six main types of built-in objects:

Number	10 or 2.71828 or 1.23E-12
String	“Hello, world!”
List	[1, 17, 44] or [“pickle”, “apple”, “scallop”]
Tuple	(4, 5) or (“homework”, “exam”)
Dictionary	{“food” : “something you eat”, “lobster” : “an edible arthropod”}
File	we'll talk about this one later…

notice the different symbols used to define types - quote, bracket, parenthesis, curly brace

- Each type of object has its own properties, which we will learn about in the next few weeks.
- It is also possible to define your own type of object, comprised of combinations of the six base types.

a list of numbers

a list of strings

Literals and variables

- A variable is a name in your program for an object.
- For example, we can assign the name `pi` to the Number object `3.14159`, as follows:

```
>>> pi = 3.14159
>>> print pi
3.14159
```

notice I am back in the interpreter here, as you can tell by the prompt >>>

- When we write out the object directly, it is a literal, as opposed to when we refer to it by its variable name. Above, `3.14159` is a literal, `pi` is a variable.

Assignment operator

```
>>> pi = 3.14159
```

The '=' means <u>assign the value</u> `3.14159` to the variable `pi` (it does NOT assert that `pi` equals `3.14159`).

```
>>> pi = 3.14159
>>> print pi
3.14159
>>> pi = -7.2
>>> print pi
-7.2
```

you can see where "variable" comes from: pi can be changed

The `import` command

Many python functions are available via packages that must be imported (other functions are always available - called built-in). For example, the log function is in the math package:

```
>>> print log(10)
Traceback (most recent call last):
  File foo, line 1, in bar
NameError: name 'log' is not defined
>>> import math
>>> print math.log(10)
2.30258509299
>>> print log(10)
Traceback (most recent call last):
  File foo, line 1, in bar
    print log(10)
NameError: name 'log' is not defined
```

foo and bar mean something-or-other-goes-here

import the math package

use the log function in the math package

for now don't worry about the details of the error message - just be aware that this means there is an error in your program.

The command line

- To get information into a program, we can use the command line.
- The command line is the text you enter after the word "python" when you run a program.

```
python my-program.py 17
```

- The zeroth argument is the name of the program file.
- Arguments larger than zero are subsequent elements of the command line, <u>separated by spaces</u>.

zeroth argument

first argument

Reading command line arguments

Access in your program like this:

```
import sys
print sys.argv[0]
print sys.argv[1]

> python my-program.py 17
my-program.py
17
```

the sys module

zeroth argument

first argument

There can be any number of arguments, accessed by sequential numbers (`sys.argv[2]` etc).

NB `argv` stands for argument vector (vector is essentially another name for a list).

Sample problem #1

- Write a program called "print-two-args.py" that reads the first two command line arguments after the program name, stores their values as variables, and then prints them to screen on the same line with a colon between.
- Use the python interpreter for quick syntax tests if you want.

```
> python print-two-args.py hello world
hello : world
```

Hint – to print multiple things on one line, separate them by commas:

```
>>> print 7, "pickles"
7 pickles
```

Solution #1

```
import sys
valA = sys.argv[1]
valB = sys.argv[2]
print valA, ":", valB
```

assign the first command line argument to the variable `valA`

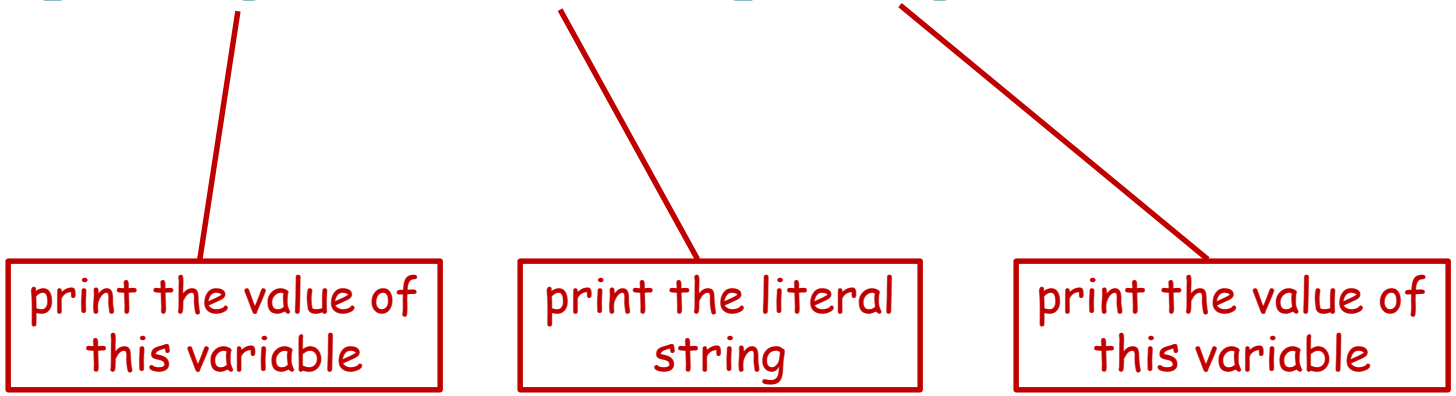
print the value of this variable

print the literal string

print the value of this variable

Alternative solution #1

```
import sys
print sys.argv[1], ":", sys.argv[2]
```



This doesn't assign the variable names, as requested in the problem, but it is otherwise functionally equivalent.

Sample problem #2

- Write a program called "add-two-args.py" that reads the first two command line arguments after the program name, stores their values as <u>number</u> variables, and then prints their sum.

```
> python add-two-args.py 1 2
3.0
```

Hint - to read an argument as a decimal number, use the syntax:

```
foo = float(sys.argv[1])
```

or for an integer number:

```
bar = int(sys.argv[1])
```

The technical name for this is "casting" - the value starts as a string object and is <u>cast</u> to a float or int object (two kinds of Number objects in Python).

Command line arguments always start as string objects

Solution #2

```
import sys
arg1 = float(sys.argv[1])
arg2 = float(sys.argv[2])
print arg1 + arg2
```

notice that this expression gets evaluated first, then printed

Alternative solutions #2

```
import sys
arg1 = float(sys.argv[1])
arg2 = float(sys.argv[2])
argSum = arg1 + arg2
print argSum
```

or

```
import sys
print float(sys.argv[1]) + float(sys.argv[2])
```

Challenge problems

Write a program called "circle-area.py" that reads the first command line argument as the radius of a circle and prints the area of the circle.

```
> python circle-area.py 15.7
774.371173183
```

Do the same thing but read a second argument as the unit type and include the units in your output.

```
> python circle-area2.py 3.721 cm
43.4979923683 square cm
```

Challenge solutions

```
import sys
radius = float(sys.argv[1])
print 3.1415 * radius * radius
```

(or slightly better)

```
import sys
import math
radius = float(sys.argv[1])
print math.pi * radius * radius
```

the math package contains most simple math constants and functions that are not built in

the math constant pi to many significant digits

```
import sys
import math
radius = float(sys.argv[1])
units = sys.argv[2]
print math.pi * radius * radius, "square", units
```

a literal string

Reading

- Chapter 1 of *Think Python* by Downey.
- Legal free PDF linked on web site.