

Introduction to Python

Genome 559: Introduction to Statistical
and Computational Genomics
Prof. James H. Thomas

Use python interpreter for quick syntax tests.

Write your program with a syntax-highlighting text editor.

Save your program in a known location and using ".py" extension.

Use the command window (or terminal session) to run your program (make sure you are in the same directory as your program).

Getting started on the Mac

- Start a terminal session
- Type "python"
- This should start the Python interpreter (often called "IDLE")

```
> python
```

```
Python 2.6.4 (something something)
```

```
details something something
```

```
Type "help", "copyright", "credits" or "license"  
for more information.
```

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

```
Hello, world!
```

Run your program

- In your terminal, Ctrl-D out of the python interpreter (or start a new terminal).
- Type "pwd" to find your present working directory.
- Open TextWrangler.
- Create a file with your program text.
- Be sure that you end the line with a carriage return.
- Save the file as "prog.py" in your present working directory.
- In your terminal, type "python prog.py"

```
> python hello.py
```

```
hello, world!
```

Objects and types

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

Number	10 or 2.71828
String	"hello"
List	[1, 17, 44] or ["pickle", "apple", "scallop"]
Tuple	(4, 5) or ("homework", "exam")
Dictionary	{"food" : "something you eat", "lobster" : "an edible arthropod"}
File	more later...

- It is also possible to define your own types, comprised of combinations of the six base types.

Literals and variables

- A variable is simply a name 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
```

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

The command line

- The command line is the text you enter after the word "python" when you run a program.

```
python my-program.py GATTCTAC 5
```

- The zeroth argument is the name of the program file.
- Arguments larger than zero are subsequent elements of the command line.

zeroth
argument

first
argument

second
argument

Reading command line arguments

Access in your program like this:

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

zeroth
argument



first
argument



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

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

Assigning variables

In order to retain program access to a value, you have to assign it to a variable name.

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

this says "give me access to all the stuff in the sys module"

this doesn't do anything - it says "get the string that is stored at index 0 in the list sys.argv and do nothing with it"

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

this says "get the string that is stored at index 0 in the list sys.argv and print it" (but it doesn't do anything else)

```
import sys  
s = sys.argv[0]
```

this says "get the string that is stored at index 0 in the list sys.argv and assign it to the variable s"

Basic string operations:

<code>s = "AATTGG"</code>	<code># assignment - or use single quotes ''</code>
<code>s1 + s2</code>	<code># concatenate</code>
<code>s2 * 3</code>	<code># repeat string</code>
<code>s2[i]</code>	<code># get character at position 'i'</code>
<code>s2[x:y]</code>	<code># get a substring from x to y (not including y)</code>
<code>len(s)</code>	<code># get length of string</code>
<code>int(s)</code>	<code># turn a string into an integer</code>
<code>float(s)</code>	<code># turn a string into a floating point decimal number</code>
<code>len(s[x:y])</code>	<code># the length of s[x:y] is always y - x</code>

Methods:

- `S.upper()`
- `S.lower()`
- `S.count(substring)`
- `S.replace(old,new)`
- `S.find(substring)`
- `S.startswith(substring)`
- `S.endswith(substring)`

Printing:

<code>print var1,var2,var3</code>	<code># print multiple variables</code>
<code>print "text",var1,"text"</code>	<code># print a combination of explicit text and variables</code>

Basic list operations:

```
L = ['dna', 'rna', 'protein']
L2 = [1, 2, 'dogma', L]
L2[2] = 'central'
L2[0:2] = 'ACGT'
del L[0:1] = 'nucs'
L2 + L
L2*3
L[x:y]
len(L)
''.join(L)
S.split(x)
list(S)
list(T)
```

```
# list assignment
# list can hold different object types
# change an element (mutable)
# replace a slice
# delete a slice
# concatenate
# repeat list
# define the range of a list
# length of list
# convert a list to string
# convert string to list- x delimited
# convert string to list - explode
# converts a tuple to list
```

Methods:

```
L.append(x)
L.extend(x)
L.count(x)
L.index(x)
L.insert(i, x)
L.remove(x)
L.pop(i)
L.reverse()
L.sort()
```

```
# add to the end
# append each element from x to list
# count the occurrences of x
# give element location of x
# insert at element x at element i
# delete first occurrence of x
# extract element i
# reverse list in place
# sort list in place
```

File reading and writing

The `open()` command returns a file object:

```
<file_object> = open(<filename>, <access type>)
```

Access types:

- 'r' = read
- 'w' = write
- 'a' = append

```
myFile = open("data.txt", "r") - open for reading
```

```
myFile = open("new_data.txt", "w") - open for writing
```

```
myString = myFile.read() - read the entire text as a string
```

```
myStringList = myFile.readlines() - read all the lines as a list of strings
```

```
myString = myFile.readline() - read the next line as a string
```

```
myFile.write("foo") - write a string (does not append a newline)
```

```
myFile.close() - always close a file after done
```

if - elif - else

```
if <test1>:  
    <block1>  
elif <test2>:  
    <block2>  
elif <test3>:  
    <block3>  
else:  
    <block4>
```

- Only one of the blocks is ever executed.
- A block is all code with the same indentation.

Comparison operators

- Boolean: `and`, `or`, `not`
- Numeric: `<`, `>`, `==`, `!=`, `>=`, `<=`
- String: `in`, `not in`

`<` is less than

`>` is greater than

`==` is equal to

`!=` is NOT equal to

`<=` is less than or equal to

`>=` is greater than or equal to