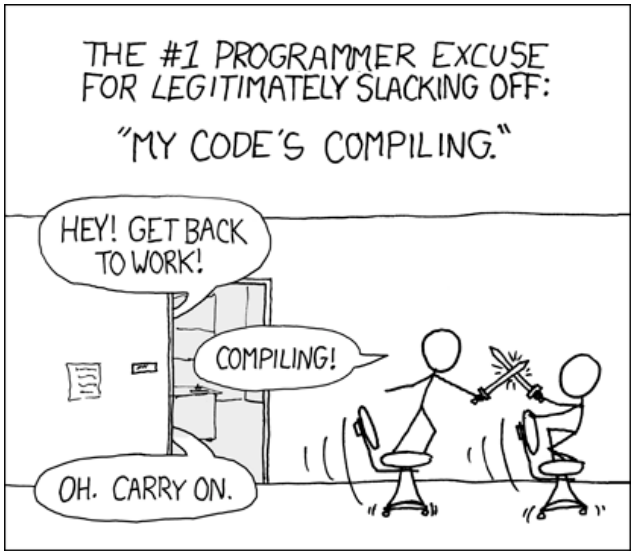
Today's Class

- Python
- Practice Exercises

Scripting Languages

Programs need to be converted into instructions the computer can understand – 'machine code'

http://xkcd.com/303/



Two methods:

Compilation - Code is converted ('compiled') to produce an executable file that can be directly read by the machine

Example of compiled languages: C, C++, Java

Interpreter – No compilation and therefore no executable file. Code is converted to machine instructions at run-time

Examples: Ruby, Python, Perl, R

Interpreted usually slower than compiled language for execution, but code development is faster.

Data Types

Type	Example

Number	`4567131 (int)`
	`4.321401 (float)`
	`6E10 (float)`

String	`"AGCT"` `'Hello World'`
Boolean	`True, False`
Tuple	`('apple', 'pear', 'orange')` `('apple',)`
Lists	`["Billy, "Bob", "Sue"]` `[1, 1423534, "Mary"]`
Sets	`{1,2,3} or set([1,2,3])`
Dictionary	`{'A':1, 'G':2, 'C':3, 'T':4}`

Use **`type()`** to determine what data type something is.
Change type: **`str(`**`1`**`)`**, **`float(`**`1`**`)`**, etc. - but not always possible.

Variables – Store values

Assignment:

```
>>> a = 1
>>> a = 'hello'
```

Naming Rules:
1) Must start with a letter
2) No spaces (use '_' instead)
3) Cannot use certain reserved words
   e.g., "`if`"

4) Try to choose a name that is short but descriptive:

   e.g., `outfile`, `indir`, `my_age`, etc.

Note how unclear it is what type '`a`' is…

Note:

"`a=1`" assigns the name `a` to the value 1

"`a`" is a variable – something whose value can vary

Assignment is assigning a space in memory for the object.

Variables (cont'd)

Assignment vs Testing:

```
>>> a=1
>>> a==1
True
>>> a=2
>>> a==1
False
```

Determining type:

```
>>> a=1
>>> type(a)
<type 'int'>

>>> a='hello'
>>> type(a)
<type 'str'>
```

Note: Operators can work differently, depending on the object's type:

```
>>> a = 1
>>> b = 2
>>> a + b
3
```

For ints, "+" will add

```
>>> a = '1'
>>> b = '2'
>>> a + b
'12'
```

For strings, "+" will concatenate

"Operator overloading"

Functions

Syntax:

```
function(parameter)
```

```
>>> print('hello')
hello
>>> len('hello')
5

>>> quit()
```

You can write your own functions.

Functions

Syntax:

```
def functionName(arguments):
    expressions and/or statements
    return result [OR: print result]
```

Arguments are optional!

Example:

```
def sumCart(price1, price2):
        total = price1 + price2
        return total
```

Difference between **print** vs **return**:

Print will just print the value. Return will return the result – which you can store, e.g.:

```
total_price = sumCart(200,350)
```

Importing Modules

- To keep your code concise, consider saving modules in a separate file and importing

e.g.,

Function definitions in calc_sums.py

```
$ cat myscript.py
import calc_sums
```

imports everything

Alternatively

```
$ cat myscript.py
from calc_sums import sumCart
```

Imports only `sumCost()`

Note: the module needs to be in the search path. You can a dir to your path:

```
import sys
sys.path.append('the/path')

import mymodule
```

Practice

- The preceding function `sumCart` works for only two items (i.e., takes exactly two arguments).

- Can you alter this function so that it will take *any number of arguments* and return the sum?

(List comprehension to convert strings to floats)

Object-oriented syntax

General format:

```
object.method()
```

For example:

```
>>> 'hello'.capitalize()
'Hello'
>>> "hello".swapcase()
'HELLO'
```

Following Assignment:

```
>>> my_string = 'hello'
>>> my_string.swapcase()
'HELLO'
```

`dir(str)` to see methods available for a string

`dir(int)` to see methods available

Methods (follow-up)

General format:

```
object.method()
```

You can chain methods together:

```
object.method_1().method_2()
```

Example 1.

```
>>> mystr='hello'
>>> mystr.swapcase()
>>> mystr.replace('e', 'a')
```

Now try combining:

```
>>> mystr.replace('e', 'a').swapcase()
```

Example 2.

```
>>> mystr='elizabeth    anne    ostrowski'
>>> mystr.split('\t')
>>> mystr.swapcase()
```

Now try combining:

```
>>> mystr.split('\t').swapcase()
```

What went wrong? Why?
How can you fix it?

Decision-making – if statements

Syntax:

```
if condition:
    statement
elif condition:
    statement
else:
    statement
```

IMPORTANT!!

- Python REQUIRES indentation
- Four spaces is the convention
- There is no “end” or “done” statement – failure to indent subsequent lines signifies the end of a loop.
- To end a loop, press return twice.

Example:

```
>>> elizabeth_age = 16
>>> jane_age = 20

>>> if elizabeth_age > jane_age:
...     print "Elizabeth is older than Jane."
... else:
...     print "Jane is older than Elizabeth."
```

Q: What is wrong with the code above? How can you make it better?

Flow control – for loops

Syntax:

```
for each_item in items:
        statement
```

```
>>> for i in range(1,5):
...     print i
...
1
2
3
4

>>> refrig = {'eggs':10, 'pears':4, 'apples':7}
>>> for items in refrig:
...     print items
...
eggs
```

Flow control – while loops

```
>>> i=0
>>> while(i<5):
...     i+=1
...     print i
...
1
2
3
4
5
```

Tuples (immutable) and Lists [mutable]

Tuple:

```
>>> a = (1,2,6,8)
>>> a[0]
1
>>> a[1:3]
2,6
```

Specify the index using square brackets [ ]:

`a[0]` is the first element in `a`

`a[1]` is the second element in `a`

`a[1:3]` prints elements 1:3 of `a`, non-inclusive
(non-inclusive – i.e., 1 to 3, *not including* 3)

Reminder: Python counts from 0

List:

```
>>> a = [1,2,6,8]
>>> a[0]
1
>>> a[1:3]
[2,6]
>>> a = [1,2,6,8]
>>> a[2:]
[6, 8]
```

```
>>> a.extend([6,7,8])
>>> a
[1, 2, 6, 8, 6, 7, 8]


>>> del a[0]
>>> a
[2, 6, 8]

(try del on a tuple!)
```

Note the square brackets, *even* for indexing into an (tuple)!

List Comprehensions

Syntax:

```
[ expression for item in list if conditional ]
```

Examples:

```
>>> a=[1,2,6,8]

>>> b = [x*2 for x in a]
>>> b
[2, 4, 12, 16]
```

```
>>> a
[1, 2, 6, 8]

>>> b=[x+1 for x in a if x>2]
>>> b
[7, 9]
```

Loop:

```
x=[]
for i in range(10):
    x.append(i**2)
```

OR

List Comprehension:

```
x = [i**2 for i in range(10)]
```

Practice

- Can you take our earlier example of a for loop that prints the values 1 through 5, and write it instead as a list comprehension?
- Modify it so that it instead prints the first 10 multiples of 5.
- Write a list comprehension that selects even numbers from a tuple containing a sequence of numbers (even and odd).
- Alter it to return odd numbers instead.

- *Note: The output is different. In the for loop, we are printing values to stdout. With a list comprehension, we are returning a list containing the values.*

Dictionary – look up table

- Associates a key with a value
- Other languages refer to as a *hash* or an *associative array*
- Unordered collection (unlike a list or a tuple)

Syntax:

```
d = {'key1': 'value1','key2': 'value2'}
```

Examples:

```
>>> in_refrig = {'eggs':10,
'pears':4, 'apples':7}

>>> in_refrig['eggs']
10
```

```
>>> convert = {'DDB0232428':1,
'DDB0232429':2, 'DDB0232430':3,
'DDB0232431':5, 'DDB0232432':6}

>>> convert['DDB0232430']
3
```

Practice

(1) Write a function `in_fridge()` that tells you how many of a particular item are in your refrigerator.

- Question: What inputs will you need to your function?

(2) Place your function definition in a separate file and import it as a module into a script that calls it.

Dictionary (cont'd)

Example. Use a dictionary to replace DDB format with chr number

```
# Create the dictionary
convert = {'DDB0232428':1, 'DDB0232429':2, 'DDB0232430':3,
'DDB0232431':5, 'DDB0232432':6}

# Original line (Chr, Pos, SNP – where chromosome is in DDB format)
orig_line = 'DDB0232429     458     A'

# Split line by tabs
cols = orig_line.split("\t")

# Create an output line (also tab-delimited)
new_line = "{}\t{}\t{}".format(convert[cols[0]],cols[1],cols[2])

print new_line
```

Q: What is **`cols`** - i.e., what type of object is returned by the **`split()`** method? What type of object is **`orig_line`**?

Formatting Output

```
>>> a = 'hello'
>>> b = 'world’

>>> print "My first word is {0}, my second word is {1}".format(a,b)
My first word is hello, my second word is world

>>> print "My first word is {}, my second word is {}".format(a,b)
My first word is hello, my second word is world

>>> print "My first word is %s, my second word is %s" % (a,b)
My first word is hello, my second word is world

>>> name = 'Elizabeth'
>>> age = 20
>>> print "My name is %s, and I am %d years old" % (name, age)
My name is Elizabeth, and I am 20 years old
```

Homework

- LCTHW
  - Complete Exercises 10-13,15,16

- For credit: Send me your history file and any scripts you created:
  1) Clear your history

     ```
     $ history –c
     ```

  2) Do the exercises

  3) Save your history

     ```
     $ history > lastname_history_111014.txt
     ```

  4) Email me the file before the start of the next class

Quiz

- Python!

- Do you know the basic data types in Python? Can you create objects of these types and store them?

- Can you use flow-control statements?
- Can determine what type something is?
- Can you write a function?