Python Primer for PyRosetta

Most slides are blatantly copied from
http://karchinlab.org/fcbb2_spr14/practical-handouts/p01_opt/Python_Programming_Basics.pdf
Credits

• **Learn Python in 10 minutes**

• **Python maintains a site containing online tutorials:**

• **Google’s Python Class:**
  – [http://code.google.com/edu/languages/google-python-class/](http://code.google.com/edu/languages/google-python-class/)

• **Python’s tutorial:**
  – [http://docs.python.org/tutorial/](http://docs.python.org/tutorial/)
The Python Interpreter

• Open Terminal

• Type `python`
Strongly-typed, yet dynamic

- **Strongly-typed**: restricts intermingling of data types
- **Dynamic**: variable types are not pre-declared, interpreted implicitly
- Isn’t that a contradiction?
- Interpreter keeps track of variable types
  - The type is associated with the value
- Rarely uses types which limit usage
Getting help

- Use `help()` to get info on anything and everything
- Use `dir()` to find all of an object’s methods (will explain later)
- Use `__doc__` method to get quick documentation

```python
>>> import math
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.

>>> dir(math)

>>> print math.log.__doc__
log(x[, base])

    Return the logarithm of x to the given base. If the base not specified, returns the natural logarithm (base e) of x.
```
Python syntax

- Programming “blocks” **must be indented**
- **Indented** blocks are preceded by a colon
- Comments start with `#`
- Assign values with `=`
- Test equality with `==`

```python
>>> def counter(top):
    print "Beginning counter"
    for i in range(top):
        print i

>>> counter(5)
Beginning counter
0
1
2
3
4
>>> a = 3
>>> b = 5
>>> a + b
8
>>> a == b
False
```
Operators, type conversion

- Arithmetic operators: +, -, *, /, **, %

```python
>>> 1+2+3
6
>>> a = 2
>>> b = 5
>>> a*b
10
>>> b**a
25
>>> c = "3"
>>> a+c
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> a+int(c)
5
>>> c = int(c)
>>> a+c
5
>>> 5%2
1
>>> 5/2
2
>>> 5.0/2
2.5
>>> 5.0+3/5
5.0
>>> a = 2
>>> b = 5
>>> b/a
2
>>> float(b)/a
2.5
>>> b/a
2
>>> b = float(b)
>>> b/a
2.5
```
Data types: Lists

- One-dimensional arrays
- Anything can be put inside lists (including other lists)
- List indices start from 0
- Lists can be sliced and concatenated
- Last elements can be accessed using negative-indexing
- Individual elements can be changed (mutable)

```python
>>> a = ['spam', 'eggs', 100, 1234]
>>> a
['spam', 'eggs', 100, 1234]
>>> a[3]
1234
>>> a[:2] + ['bacon', 2*2]
['spam', 'eggs', 'bacon', 4]
>>> a
['spam', 'eggs', 123, 1234]
>>> q = [2, 3]
>>> p = [1, q, 4]
>>> p
[1, [2,3], 4]
>>> p[-1]
4
>>> p.append('BANANA')
>>> print(p)
[1, [2, 3], 4, 'BANANA']
```
Data types: Lists

- One-dimensional arrays
- Anything can be put inside lists (including other lists)
- List indices start from 0
- Lists can be sliced and concatenated
- Last elements can be accessed using negative-indexing
- Individual elements can be changed (mutable)

```python
>>> a = range(10)
>>> print a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> len(a)
10
>>> a[:] #shallow copy of list a
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> a[2:7]
[2, 3, 4, 5, 6]
>>> a[:7]
[0, 1, 2, 3, 4, 5, 6]
>>> a[:3]
[0, 1, 2]
>>> a[2:8:2]
[2, 4, 6]
>>> a[:2]
[0, 2, 4, 6, 8]
```
More on the range function

**range(stop) OR range(start, stop[, step])**

- Versatile function to create lists containing arithmetic progressions. It is most often used in for loops.
- **The arguments must be plain integers.**
- If step argument omitted, it defaults to 1. If start argument omitted, it defaults to 0.
- The full form returns a list of plain integers [start, start + step, start + 2 * step, ...].
- If step is positive, the last element is the largest start + i * step less than stop.
- If step is negative, the last element is the smallest start + i * step greater than stop.
- Step must not be zero (or else ValueError is raised).

**Examples:**

```python
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(1, 11)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> range(0, 30, 5)
[0, 5, 10, 15, 20, 25]
>>> range(0, 10, 3)
[0, 3, 6, 9]
>>> range(-10, 0, 1)
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
>>> range(0, -10, -1)
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> range(0)
[]
>>> range(1, 0)
[]
```
Lists Methods (try these on your own)

**append(item)** Append an item to the end of the list:

```python
>>> l = [1, 2, 3]
>>> l.append(4)
>>> l
[1, 2, 3, 4]
```

**count(item)** Count the number of occurrences of the given item in the list. Note: This function also works on tuples (see next slides).

```python
>>> l = [1, 2, 3, 1, 2, 1]
>>> l.count(1)
3
>>> l.count(10)
0
```

**extend(other_list)** Extend one list with the contents of other_list.

```python
>>> l = [1, 2]
>>> l.extend([3, 4])
>>> l
[1, 2, 3, 4]
```
index(val) Returns the index of the first item in the list whose value matches the given value. ValueError if no match is found. Note: This function also works on tuples.

```python
>>> l = [1,2,2,1,3]
>>> l.index(2)
1
>>> l.index(10)
ValueError: list.index(x): x not in list
```

insert(pos, item) Insert the given item at the specified position. If the position is past the end of the list, insert at the end.

```python
>>> l = [1,2,3,4]
>>> l.insert(1,10)
[1,10,2,3,4]
>>> l.insert(10,"end")
>>> l
[1,10,2,3,4,"end"]
```
**Lists Methods (cont.)**

**pop([pos])**  Remove and return the element at the specified position. If no position is given, defaults to the last element in the list.

```python
>>> l = [1,2,3,4,5]
>>> l.pop()
5
>>> l
[1,2,3,4]
>>> l.pop(0)
1
```

**remove(val)**  Remove the first element in the list whose value matches the given value. ValueError if no match is found.

```python
>>> l = [1,2,3,2,1]
>>> l.remove(2)
>> l
[1,3,2,1]
>>> l.remove(10)
ValueError: list.remove(x): x not in list
```
**Lists Methods (cont.)**

**reverse()** Reverse the order of elements within the list. Changes the list in place instead of returning a modified copy.

```python
given_list = [1, 2, 3, 4, 5]
given_list.reverse()
given_list
```

```
[5, 4, 3, 2, 1]
```

**sort()** Sorts the list in place ordering elements from smallest to largest.

```python
given_list = [10, 2, 3, 10, 100, 54]
given_list.sort()
given_list
```

```
[2, 3, 10, 10, 54, 100]
```

See also `sorted(list)`:

```python
sorted([5, 2, 3, 1, 4])
```

```
[1, 2, 3, 4, 5]
```

```python
sorted([5, 2, 3, 1, 4], reverse=True)
```

```
[5, 4, 3, 2, 1]
```
Data Types: Strings

• Any text enclosed in single quotes (‘ ’) or double quotes (“ “)
• Backslash (\) is used as the escaping character.
  – \n: newline, \t: tab
• Strings can be concatenated with +, and repeated with *
• Strings can be indexed (index of the first character of a string is 0)
• Substrings are specified with the slice notation: two indices separated by a colon

```python
>>> word = 'Help' + 'A'
>>> word*3
'HelpAHelpAHelpA'
>>> word[3]
'p'
>>> word[2:4] # slicing
'lp'
>>> word[:2] # The first two characters
'He'
>>> word[2:] # Everything except the first two characters
'lpA'
```
String concatenation, slices, indexing

• “print” can print built-in objects to standard out

```python
>>> x = '01234'
>>> y = "56789"
>>> x+y
'0123456789'
>>> x = x+y
>>> x
'0123456789'
>>> x[0]
'0'
>>> x[9]
'9'
>>> x[2:8]
'234567'
>>> x[:2]
'01'
>>> x[8:]
'89'
>>> x[-2:]
'89'
```
String and “print”

• “print” can print built-in objects to standard out

```python
>>> x[2:10]
'23456789'
>>> x[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> x = 3.1415926535
>>> print x
3.1415926535
>>> print "pi is "+x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'float' objects
>>> print "pi is "+str(x)
pi is 3.1415926535
>>> print "pi is\n"+str(pi)
pi is
3.1415926535
```
Some String Methods (try these on your own)

# Try these useful string methods:

```python
>>> "Python".index("thon")
>>> "Python".index("py")
>>> '_'.join(["a","b","c"])
>>> print "\t".join(["John","Smith"])
>>> "PyThon".lower()
```
Dictionaries – {} 

- Use dictionaries when storing a value with some key and extracting the value given the key (e.g. address book – key=names values=addresses) 
  - Keys should be unique!
- A pair of braces creates an empty dictionary: {} 
  - You can initialize an dictionary with initial key:value pairs 
  - You can add or modify a key:value pair 
  - You can delete a key:value pair using del
- dictionaries are indexed by keys, which can be any immutable type 
  - strings or numbers can always be keys 
  - Tuples can be used as keys if they contain only immutable type 
- Useful methods for dictionaries: d.keys(), d.values()

```python
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127  #add new key:value pair
>>> tel
{'sape': 4139, 'jack': 4098, 'guido': 4127}
>>> tel['jack']  #extract the value with the key ‘jack’
4098
```
Dictionaries – {}  

- Use dictionaries when storing a value with some key and extracting the value given the key  
  - Keys should be unique!

- A pair of braces creates an empty dictionary: {}
  - You can initialize an dictionary with initial key:value pairs
  - You can add or modify a key:value pair
  - You can delete a key:value pair using del

- dictionaries are indexed by keys, which can be any immutable type
  - strings or numbers can always be keys
  - Tuples can be used as keys if they contain only immutable type

```python
>>> del tel['sape'] #delete the key:value pair with the key 'sape'
>>> tel
{'jack': 4098, 'guido': 4127}
>>> tel.keys() #retrieve the list of keys in the dictionary
['jack', 'guido']
>>> 'guido' in tel #test whether 'guido' key is in the dictionary
True
```
Five ways to initialize a dictionary

The following examples all return a dictionary equal to
{"one": 1, "two": 2, "three": 3}:

```python
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([( 'two', 2), ( 'one', 1), ( 'three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```
Dictionary Methods (try these on your own)

**len(d)**  Return the number of items in the dictionary d.

**del d[key]**  Remove d[key] from d. Raises a **KeyError** if key is not in the map.

**key in d**  Return **True** if d has a key key, else **False**.  **Make sure you add the ‘…’ in the key**

**key not in d**  Equivalent to  **not key in d**.

**clear()**  Remove all items from the dictionary.

**fromkeys(seq[, value])**  Create a new dictionary with keys from seq and values set to value.

**items()**  Return a copy of the dictionary’s list of (key, value) pairs.

**CPython implementation detail:**  Keys and values are listed in an arbitrary order which is non-random, varies across Python implementations, and depends on the dictionary’s history of insertions and deletions

**update([other])**  Update the dictionary with the key/value pairs from other, overwriting existing keys.  Return **None**.
Dictionaries of Lists and Lists of Dictionaries ...
Writing Scripts

• Create a file with a .py extension
• Write code and save
• Open a terminal. At the command line, type:
  python filename.py
• Note: Making the script executable,
  – Add the following on the first line:
    #!/usr/bin/env python  Or, #!/usr/bin/python
  – Change the script mode:
    chmod u+x filename.py  To make the file executable for you
  – Run the script:
    ./filename.py
Two sample scripts using list or dict

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>C</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>G</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>T</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

\[ M_{ij} = P(\text{nucleotide}_i \text{ precedes \text{nucleotide}_j}) \]

dna2int = {'A':0,'C':1,'G':2,'T':3}

model = [[0.1,0.2,0.3,0.4],
          [0.4,0.3,0.2,0.1],
          [0.2,0.2,0.3,0.3],
          [0.3,0.3,0.2,0.2]]

# what is the probability of seeing 'A' followed by 'C'? 
print model[dna2int['A']][dna2int['C']]

model-1.py
Cont.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.1</td>
<td>0.2</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>C</td>
<td>0.4</td>
<td>0.3</td>
<td>0.2</td>
<td>0.1</td>
</tr>
<tr>
<td>G</td>
<td>0.2</td>
<td>0.2</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>T</td>
<td>0.3</td>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
</tr>
</tbody>
</table>

---

**model-2.py**

```python
model = {'AA': 0.1, 'AC': 0.2, 'AG': 0.3, 'AT': 0.4,
         'CA': 0.4, 'CC': 0.3, 'CG': 0.2, 'CT': 0.1,
         'GA': 0.2, 'GC': 0.2, 'GG': 0.3, 'GT': 0.3,
         'TA': 0.3, 'TC': 0.3, 'TG': 0.2, 'TT': 0.2}

# what is the probability of seeing 'A' followed by 'C'? 
print model['AC']
```
The *if* Statement -

```python
if (cond): (body)
```

- For conditional execution
  - run body statements (*body*) only when the condition (*cond*) is true
- (*cond*) The standard comparison operators can be used:
  - `==` (equal to)
  - `!=` (not equal to)
  - `<` (less than)
  - `>` (greater than)
  - `<=` (less than or equal to)
  - `>=` (greater than or equal to)

```python
a = int(raw_input("input \number: "))
if a < 0:
    print "negative"
elif a == 0:
    print "zero"
else:
    print "positive"
```
(cond)

• Any non-zero integer value is true; zero is false.
  Any sequence with a non-zero length is true; empty sequences are false

if-else-1.py

```python
if -1:
    print "-1 means True"
else:
    print "-1 means False"

if 0:
    print "0 means true"
else:
    print "0 means false"
```

-1 means True
0 means false

if-else-2.py

```python
If [0]:
    print "[0] means True"
else:
    print "[0] means False"

If []:
    print "[] means true"
else:
    print "[] means false"
```

[0] means True
[] means false
The **for** Statement –

\[
\text{for } (\text{obj}) \text{ in } (\text{seq}): (\text{body})
\]

- Used to iterate over the elements (**obj**) of a sequence (**seq**)
- The (**body**) should be *indented*
- Example (**for.py**)
  ```python
  nts = ['A', 'C', 'G', 'T']
  for i in range(len(nts)):
      print(i, nts[i])
  ```
- Output:
  
  0 A
  1 C
  2 G
  3 T
The **while** Statement -

**while** \((cond)\): \((body)\)

- Used for repeat execution
  - Executes \((body)\) as long as the \((cond)\) remains true
- Conditions are the same as the **if** statement
- The \((body)\) should be *indented*
- Example
  
  ```
  a = 0
  b = 1
  while b < 10:
      print b
      a, b = b, a+b
  ```
break

- The **break** statement “breaks” out of the **smallest enclosing loop**

    break.py

```python
for i in range(2, 4):  #what are the values of i?
    series = []
    x = 0

    while 1:
        series.append(i*10+x)
        x += 1
        if x == 10:
            break

    print ','.join([str(num) for num in series])
```

**Output:**

```
20,21,22,23,24,25,26,27,28,29
30,31,32,33,34,35,36,37,38,39
```
The `continue` statement continues with the next iteration of the loop; steps within smallest loop that are after `continue` are skipped.

```python
scores = ['1.2', '1.5', 'NA', '6.8', '5.0']

total = 0
for value in scores:
    if value == 'NA':
        continue

    total += float(value)

print total
```

Output:

```
14.5
```
Defining Functions: `def` Statement

- The keyword `def` introduces a function definition.
- It must be followed by the `function name` and the parenthesized list of formal parameters.
- The statements that form the body of the function start at the next line, and must be indented by exactly one tab.
- All functions must be defined prior to being used.
- The function may be required to `return` a value at the end.
Example

```python
# Function to calculate a vector
def vector(ptA, ptB):

# Function to calculate vector magnitude
def vabs(vec):
    return (math.sqrt(vec[0]**2 + vec[1]**2 + vec[2]**2))

# Function to calculate dot product of vectors
def dot(v1, v2):
    return (v1[0]*v2[0] + v1[1]*v2[1] + v1[2]*v2[2])
```
Passing Parameters into Functions

• Pass by Value:
  – Only value passed
  – Local copy of the parameter generated
  – After the function returns, the caller’s scope is unchanged

• Pass by Reference
  – Implicit reference in argument
  – Function can modify the parameter in the caller’s scope
  – This is what you’ll be doing when manipulating a pose in PyRosetta

Original Reference  http://www.python-course.eu/passing_arguments.php
So what does Python do? Both!

```python
# This function shows the memory address of the variable x
def ref_demo(x):
    print "x=",x," id=",id(x)
    x=42
    print "x=",x," id=",id(x)

>>> x=9
>>> id(x)
140567329703816
>>> ref_demo(x)
x= 9  id= 140567329703816
x= 42  id= 140567329705000
>>> x
9
```

Call-by-reference

Call-by-value
So what does Python do? Both!

```python
def modifier(list):
    print list
    list += [47,11]
    print list

>>> fib = [0,1,1,2,3,5,8]
>>> modifier(fib)
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 47, 11]
>>> print fib
[0, 1, 1, 2, 3, 5, 8, 47, 11]

>>> fib = [0,1,1,2,3,5,8]
>>> modifier(fib[:])
[0, 1, 1, 2, 3, 5, 8]
[0, 1, 1, 2, 3, 5, 8, 47, 11]
>>> print fib
[0, 1, 1, 2, 3, 5, 8]  # Shallow Copy
```
Deep vs Shallow Copying w.r.t. PyRosetta Pose

• Shallow Copy
  – Constructs a new compound object
  – Inserts references to the objects into it
  – `new_pose=old_pose`  # `old_pose` is an existing pose object

• Deep Copy
  – Constructs a new compound object
  – Inserts copies of the objects
  – `new_pose=Pose()`
    
    `new_pose.assign(old_pose)'

File I/O

1. `filehandle = file("/path/filename.txt","r")`
2. `text = filehandle.readlines()`
   - text will be a list of strings
   - Each line will be one string element of the list
3. `filehandle.close()`
4. Use file or open command
   - ‘r’ - read
   - ‘w’ - write
   - ‘a’ - append
5. Note: `filehandle.readline()` will read one line at a time

`filehandle.write()` allows you to write to a file opened in the ‘w’ or ‘a’ mode
**mutation.txt**

<table>
<thead>
<tr>
<th>chr</th>
<th>strand</th>
<th>pos</th>
<th>ref</th>
<th>mut</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>chr3</td>
<td>+</td>
<td>526452</td>
<td>A</td>
<td>C</td>
<td>0.52</td>
</tr>
<tr>
<td>chrX</td>
<td>+</td>
<td>55267</td>
<td>T</td>
<td>C</td>
<td>0.21</td>
</tr>
<tr>
<td>chr1</td>
<td>-</td>
<td>256288</td>
<td>C</td>
<td>G</td>
<td>0.77</td>
</tr>
<tr>
<td>chr21</td>
<td>+</td>
<td>526345</td>
<td>C</td>
<td>A</td>
<td>0.24</td>
</tr>
<tr>
<td>chr12</td>
<td>+</td>
<td>98224</td>
<td>G</td>
<td>T</td>
<td>0.37</td>
</tr>
<tr>
<td>chr5</td>
<td>+</td>
<td>425552</td>
<td>G</td>
<td>A</td>
<td>0.55</td>
</tr>
</tbody>
</table>

**parseMutation.py**

```python
scoreDict = {}

fd = file("mutation.txt","r")
line = fd.readline()    #skip header
line = fd.readline()    #first mutation
while line != "":
    cols = line.rstrip().split("\t")
    mutationKey = "_".join(cols[:5])
    #mutationKey = tuple(cols[:5])
    scoreDict[mutationKey] = float(cols[-1])
    line = fd.readline()
fd.close()

for key in scoreDict.keys():
    print key, scoreDict[key]
```

**Code Explanation:**
- The script reads a file named `mutation.txt`.
- It skips the header line.
- Each line is split into columns (chr, strand, pos, ref, mut), and the score is extracted.
- The key for each mutation is created from the first five columns.
- Scores are stored in a dictionary using the mutation key.
- Finally, the dictionary is printed out.
Output (using string as key):

chr3_+__526452_A_C 0.52  
chr1_-__256288_C_G 0.77  
chr12_+__98224_G_T 0.37  
chrX_+__55267_T_C 0.21  
chr5_+__425552_G_A 0.55  
chr21_+__526345_C_A 0.24

Output (using tuple as key):

('chr21', '+', '526345', 'C', 'A') 0.24  
('chr3', '+', '526452', 'A', 'C') 0.52  
('chr1', '-', '256288', 'C', 'G') 0.77  
('chrX', '+', '55267', 'T', 'C') 0.21  
('chr5', '+', '425552', 'G', 'A') 0.55  
('chr12', '+', '98224', 'G', 'T') 0.37
Commandline Arguments

• Use the `sys` module

• In the code:

  ```python
  import sys
  program_name = sys.argv[0]
  arg1 = sys.argv[1]
  ```

• At the command line:

  ```bash
  python program_name.py arg1
  ```
Coding Styles in Python

• **Indentation:** Use 4 spaces or 1 tab per indentation level
  – spaces-only are recommended over tabs
  – Never mix tabs and spaces

• **Maximum Line Length:** Limit all lines to a maximum of 79 characters
  – Use backslash (\) to join two or more physical lines into one logical line, i.e.

    ```python
    if width == 0 and height == 0 and \
    color == 'red' and emphasis == 'strong' or \
    highlight > 100:
    ```

• **Imports should be on separate lines**
  – Yes: `import os
  import sys`
  – No: `import sys, os`

• **Read PEP 8 (Python Enhancement Proposals) for more detailed style guide for Python code**
Writing Comments with docstrings

• A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.

• Use """"triple double quotes"""" around docstrings.

• Two forms of docstrings: one-liners and multi-line docstrings.
  – One-liner for really obvious cases, for example:
    ```python
def kos_root():
    """Return the pathname of the KOS root dir."""
```
  – Multi-line docstring consist of a summary line just like a one-line docstring, followed by a blank line, followed by a more elaborate description, for example:
    ```python
def read_fastafilename(filename):
    """Read a file in FASTA format and return lists of headers and sequences
    
    Argument:
    filename -- name of a file in FASTA format
    """
```

• Read PEP 257 for more information
def x_intercept(m, b):
    """Return the x intercept of the line y=m*x+b. The x intercept of a line is the point at which it crosses the x axis (y=0)."""
    return -b/m

>>> print x_intercept.__doc__
    Return the x intercept of the line y=m*x+b. The x intercept of a line is the point at which it crosses the x axis (y=0).

>>> from pydoc import help

>>> help(x_intercept)
Help on function x_intercept in module __main__:

x_intercept(m, b)
    Return the x intercept of the line y=m*x+b. The x intercept of a line is the point at which it crosses the x axis (y=0).
Structure of Codes

• Start each file with a Python docstring that describes the file contents

• Following this, the code should be arranged in this order:
  – import statements
  – Global data   You won’t be working too much with global variables
  – Function definitions, including main() function, which is the first function to be called
  – Main method invocation. The only global execution code should be the following, located as the last section of the file:
    • if __name__ == '__main__': main()
Modules: random, math, etc.

- **random**: Generate pseudo-random numbers with various common distributions
  
  ```python
  import random
  nums = [1,2,3,4,5,6,7,8,9]
  random.choice(nums)
  random.sample(nums, 4)
  random.shuffle(nums)
  ```

- **math**: Mathematical functions (\(\sin()\) etc.)
  
  ```python
  import math
  math.sqrt(2*3)
  ```

- **Other useful modules**
  - **string**: Common string operations
  - **re**: Regular expression operations **Has methods to read and manipulate pdb**s
  - **sys**: Access system-specific parameters and functions, i.e. command line arguments
  - **os**: Miscellaneous operating system interfaces; os.path: Operations on pathnames
  - **pickle**: Convert Python objects to streams of bytes and back
  - **pdb**: The Python debugger for interactive interpreters
Module: numpy

- A package for scientific computing with Python
  - Support N-dimensional array object and useful linear algebra functions
  - Numpy is already installed in your Ubuntu
  - [http://numpy.scipy.org/](http://numpy.scipy.org/) for more information

```python
>>> from numpy import *
>>> # create a 2d array from list of list
>>> a = array([[10, 20, 30], [40, 50, 60]])
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> a[1, :]
# selecting the second row
array([40, 50, 60])
>>> a[:, 1:3]
# selecting the second and the third columns
array([[20, 30],
       [50, 60]])
>>> a[:, [2, 0]]
# selecting columns with array
array([[30, 10],
       [60, 40]])
>>> a.shape
# the dimensions of the array
(2, 3)
```
```python
>>> b = array([[1, 2, 3], [4, 5, 6]])

>>> a + b
array([[11, 22, 33],
        [44, 55, 66]])

>>> a * b  # element-wise product
array([[ 10,  40,  90],
        [160, 250, 360]])

>>> dot(a, b.transpose())  # matrix product
array([[140, 320],
        [320, 770]])

>>> zeros((2, 3), dtype=int)  # similarly, look at ones()
array([[0, 0, 0],
        [0, 0, 0]])

>>> a.reshape((3, 2))  # changing the dimensions
array([[10, 20],
        [30, 40],
        [50, 60]])

>>> a.astype(float)  # changing type
array([[ 10.,  20.,  30.],
        [ 40.,  50.,  60.],
        [ 40.,  50.,  60.]])

>>> a.sum(axis=1)  # adding all elements along the specified axis
array([ 60, 150])

• other useful functions:
  • min(), max(), argmin(), argmax(), mean(), median(), average(),
    var(), std(), cov(), floor(), ceil(), round(), ... and more!
```
Easy Python Exercises

Purpose
These exercises are here to make sure you have enough familiarity with basic programming to complete the projects and assignments for FCBB2 without running into major stumbling-blocks. If you run into one or more exercises where you have no idea how to approach the problem, please talk with the TA.
Easy Exercises Part 1

1. What is $2^{38}$?
2. What is the square root of 42 (in decimal form)?
3. Compute the sum of the first 20 perfect squares $(1^2; 2^2; 3^2; ...)$
4. Compute the 50th Fibonacci number.
5. Out of the 50 Fibonacci numbers, count how many are divisible by 3 (use modulus or remainder functions)
   
   http://stackoverflow.com/questions/7225168/testing-divisibility-by-multiple-numbers

6. Count the number of vowels in the string: “Sally sold seashells by the seashore.”
   
   http://stackoverflow.com/questions/2926383/counting-vowels

7. Create a function that will test to see if a string is palindromic (e.g., “Racecar” spelled backwards is “racecar”).
   
   http://stackoverflow.com/questions/931092/reverse-a-string-in-python
Easy Exercises Part 2

1. Create a function that will read in an arbitrary text file and count the number of words in the text file.
2. Create a function that will read in a string and give you the character frequency count (14 E’s, 12 T’s, 8 A’s, etc.)
3. Create a function “base_pair” that converts a nucleotide (A,T,G,C) to its paired nucleotide and prints it.
   

4. Create a function “opposite_strand”, given a single DNA strand, e.g. a text file containing “ATGCGGAATT...” computes the opposite strand.
5. Create a function “reverse_complement” that computes the reverse complement of a DNA sequence