# An Introduction to Python

Day 3

Renaud Dessalles

dessalles@ucla.edu

# Writing Modules

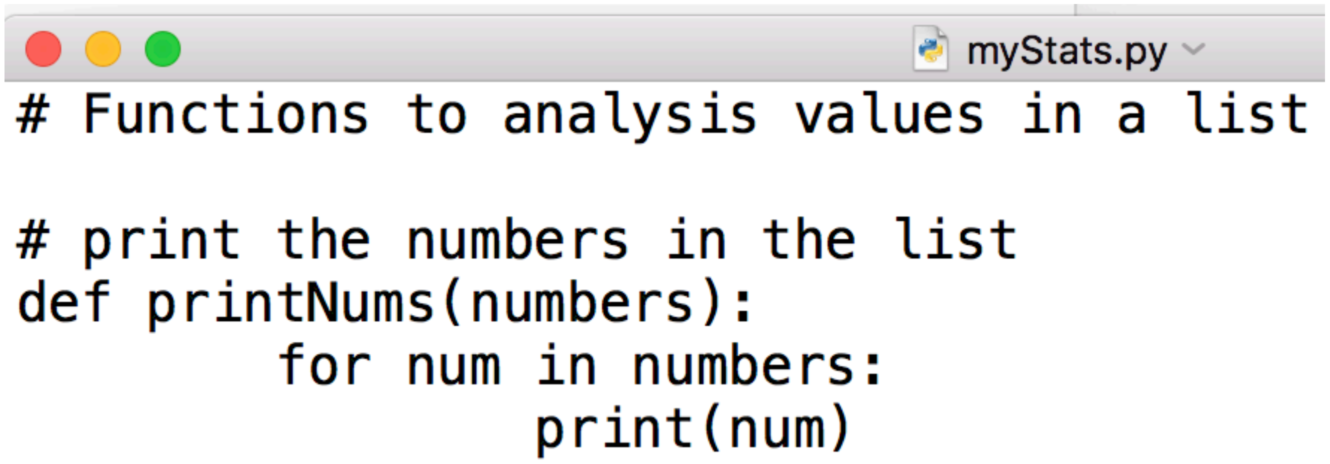
## Combining what we've learnt

Yesterday we learnt a lot of different bits of Python.

Let's summarize that knowledge by writing a **module** of functions to do various analysis on values in a list.

# myStats.py

* In a text editor:

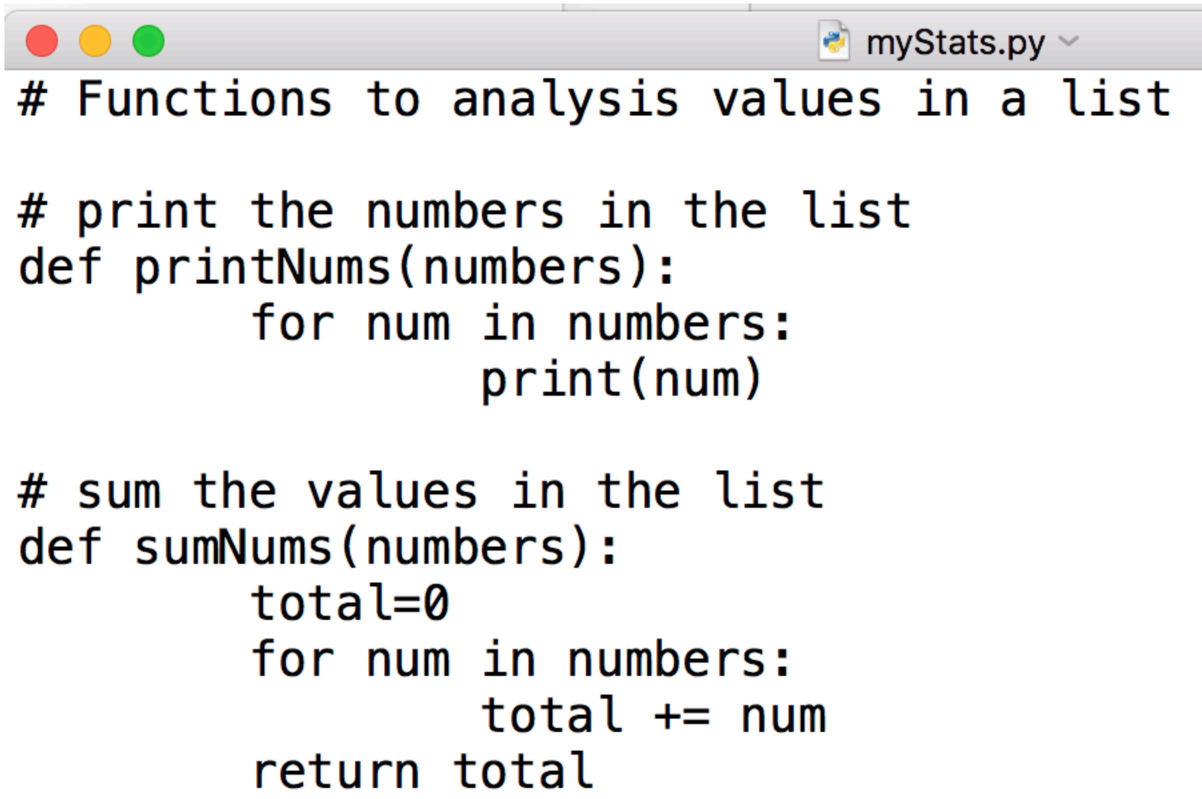


```
# Functions to analysis values in a list

# print the numbers in the list
def printNums(numbers):
        for num in numbers:
                print(num)
```

* Comment your code well so you remember what it does when you look at it again.

# myStats.py

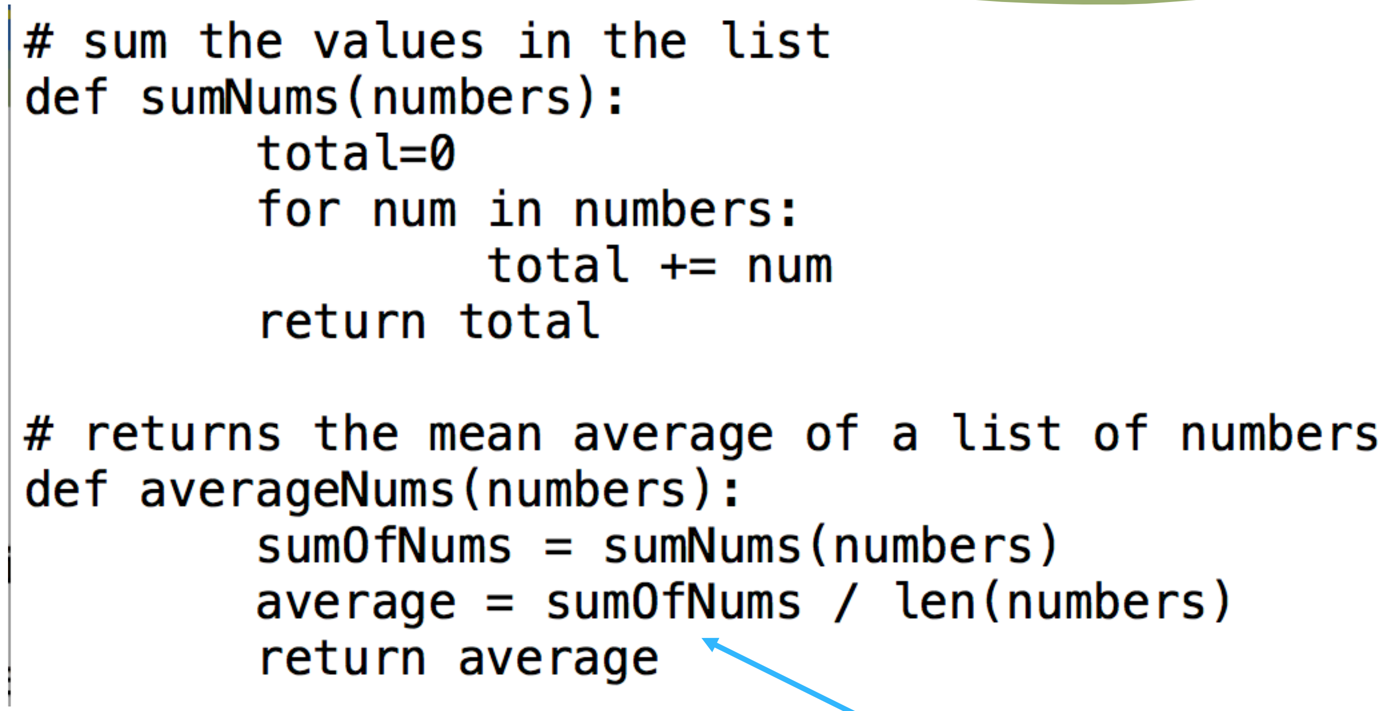
* A function to sum values:



```
# Functions to analysis values in a list

# print the numbers in the list
def printNums(numbers):
        for num in numbers:
                print(num)

# sum the values in the list
def sumNums(numbers):
        total=0
        for num in numbers:
                total += num
        return total
```

# myStats.py

* A function to average numbers:



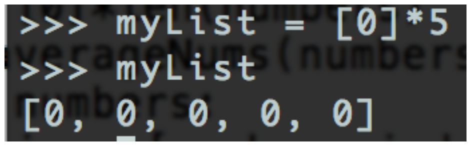
```
# sum the values in the list
def sumNums(numbers):
        total=0
        for num in numbers:
                total += num
        return total

# returns the mean average of a list of numbers
def averageNums(numbers):
        sumOfNums = sumNums(numbers)
        average = sumOfNums / len(numbers)
        return average
```

Reminder: in Python2, should put a **float** somewhere

## myStats.py

* A function to calculate the variance:

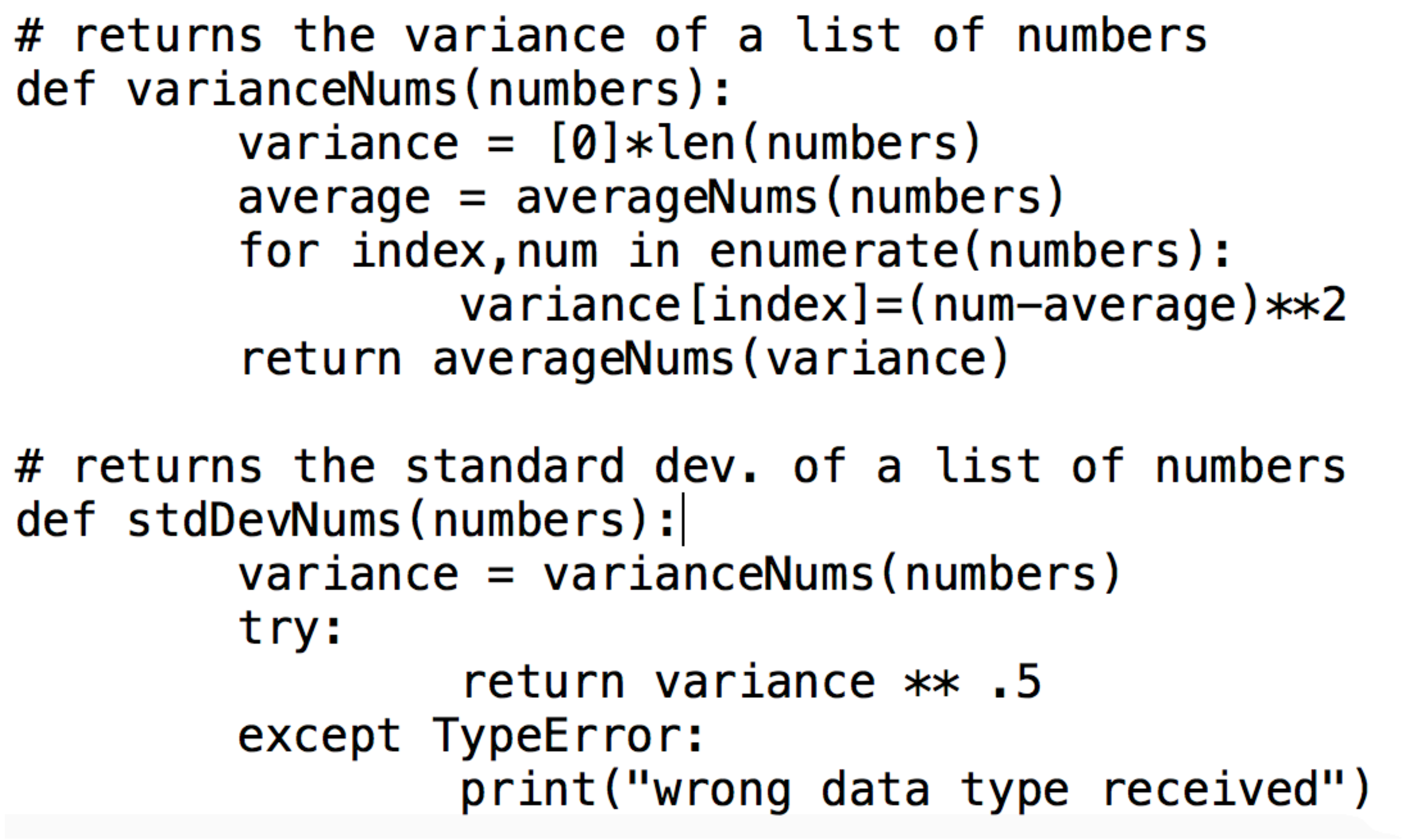
```
# returns the mean average of a list of numbers
def averageNums(numbers):
        sumOfNums = sumNums(numbers)
        average = sumOfNums / len(numbers)
        return average

# returns the variance of a list of numbers
def varianceNums(numbers):
        variance = [0]*len(numbers)
        average = averageNums(numbers)
        for index,num in enumerate(numbers):
                variance[index]=(num-average)**2
        return averageNums(variance)
```



```
>>> myList = [0]*5
>>> myList
[0, 0, 0, 0, 0]
```

# myStats.py

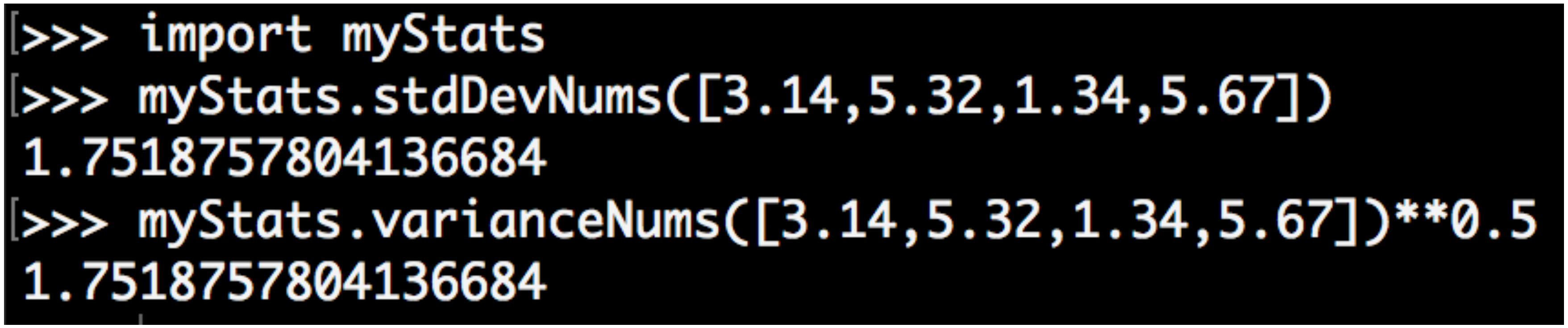
* A function to calculate the population standard deviation using the variance



```
# returns the variance of a list of numbers
def varianceNums(numbers):
        variance = [0]*len(numbers)
        average = averageNums(numbers)
        for index,num in enumerate(numbers):
                variance[index]=(num-average)**2
        return averageNums(variance)

# returns the standard dev. of a list of numbers
def stdDevNums(numbers):
        variance = varianceNums(numbers)
        try:
                return variance ** .5
        except TypeError:
                print("wrong data type received")
```

## myStats.py

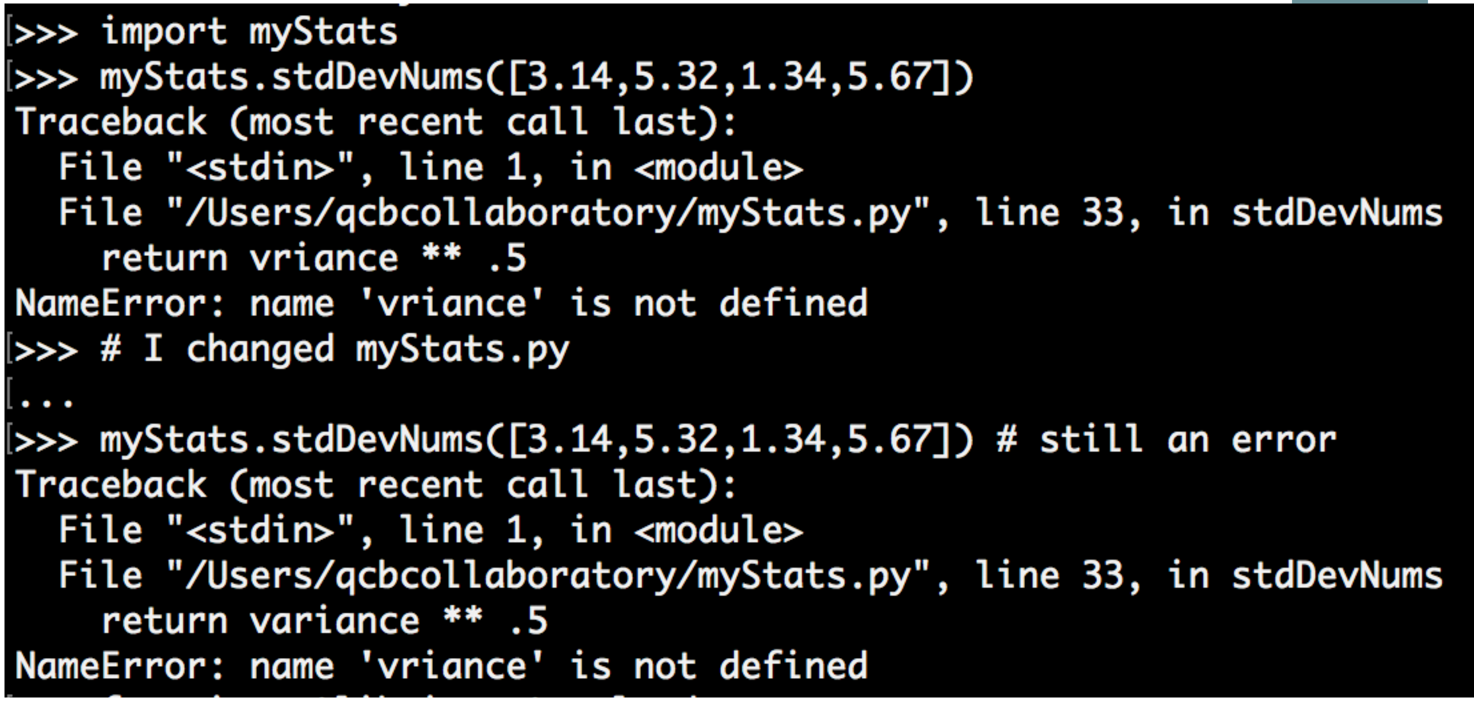
- Test it



```
[>>> import myStats
[>>> myStats.stdDevNums([3.14,5.32,1.34,5.67])
1.7518757804136684
[>>> myStats.varianceNums([3.14,5.32,1.34,5.67])**0.5
1.7518757804136684
```

## Reload modules

* Problem when debugging a module:



```
>>> import myStats
>>> myStats.stdDevNums([3.14,5.32,1.34,5.67])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/qcbcollaboratory/myStats.py", line 33, in stdDevNums
    return vriance ** .5
NameError: name 'vriance' is not defined
>>> # I changed myStats.py
...
>>> myStats.stdDevNums([3.14,5.32,1.34,5.67]) # still an error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/qcbcollaboratory/myStats.py", line 33, in stdDevNums
    return variance ** .5
NameError: name 'vriance' is not defined
```

An error

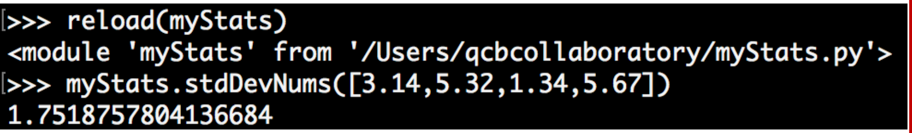
Have edited myStat.py

The error still occurs

* Solution 1 : Exit python, re-open python and import again
* Solution 2 : reload the module

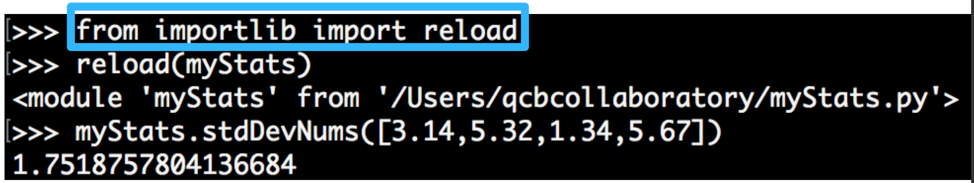
# Python 2 vs 3: Reload modules

- Python2
  - **reload** is directly available



```
>>> reload(myStats)
<module 'myStats' from '/Users/qcbcollaboratory/myStats.py'>
>>> myStats.stdDevNums([3.14,5.32,1.34,5.67])
1.7518757804136684
```

- Python3
  - Need to import **reload** via **importlib**



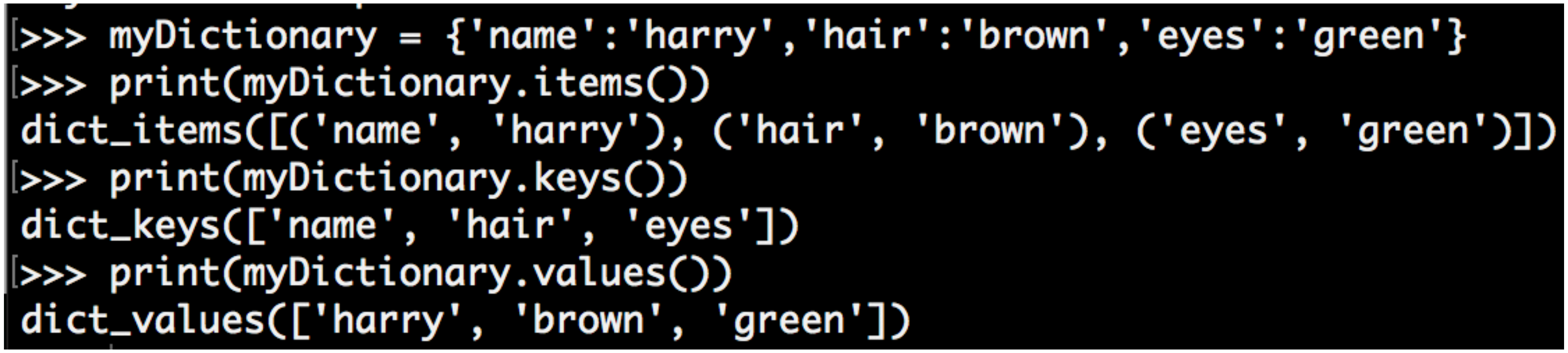
```
>>> from importlib import reload
>>> reload(myStats)
<module 'myStats' from '/Users/qcbcollaboratory/myStats.py'>
>>> myStats.stdDevNums([3.14,5.32,1.34,5.67])
1.7518757804136684
```

## Other features

# More Dictionary Methods

| | |
|---|---|
| * .items() | returns key value pairs |
| * .keys() | returns just the keys |
| * .values() | returns just the value |

dictionaries not necessarily stored in order defined... careful

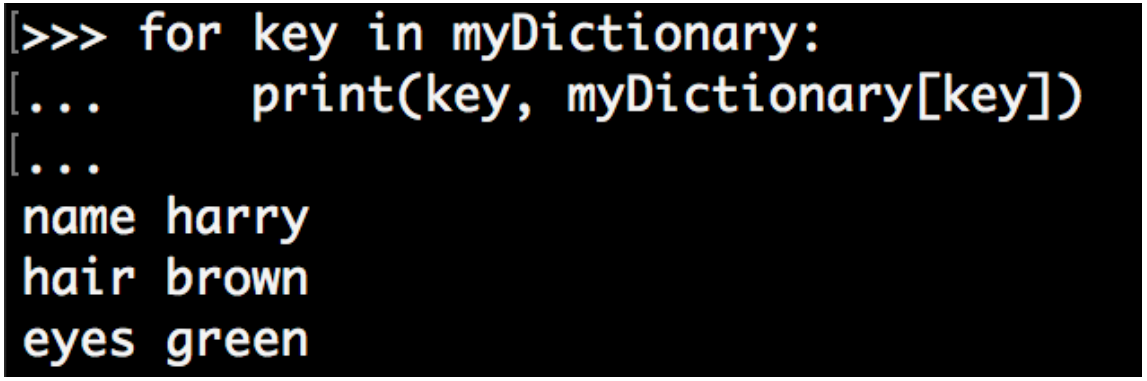


```
>>> myDictionary = {'name':'harry','hair':'brown','eyes':'green'}
>>> print(myDictionary.items())
dict_items([('name', 'harry'), ('hair', 'brown'), ('eyes', 'green')])
>>> print(myDictionary.keys())
dict_keys(['name', 'hair', 'eyes'])
>>> print(myDictionary.values())
dict_values(['harry', 'brown', 'green'])
```

* This is useful so we can iterate over dictionaries more easily...
* Note: in Python2, they returns simple lists

# Iterating over dictionaries

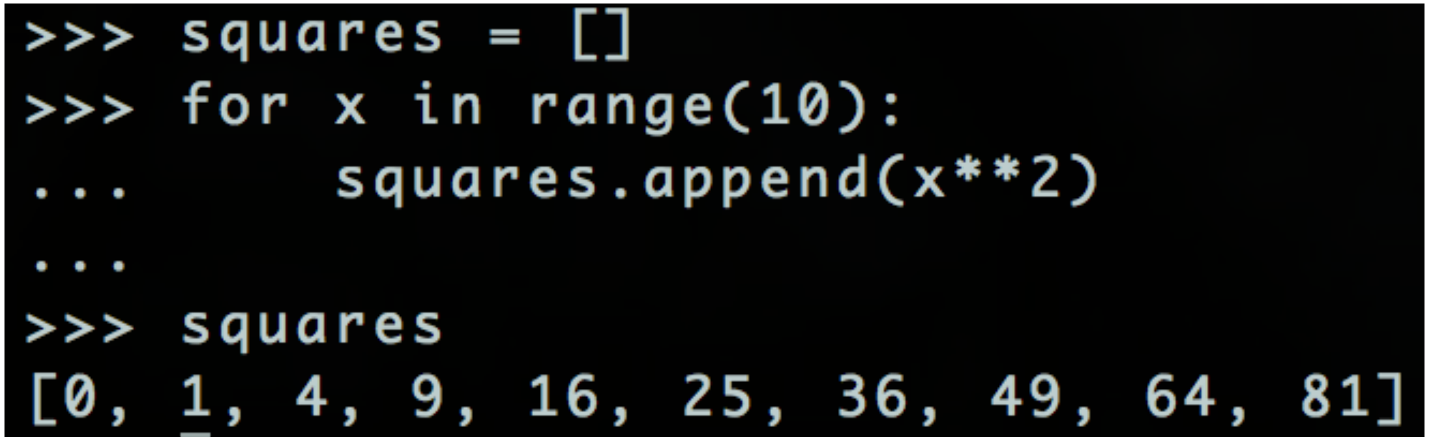
Let’s iterate over the keys:



```
>>> for key in myDictionary:
...     print(key, myDictionary[key])
...
name harry
hair brown
eyes green
```

# List Comprehension

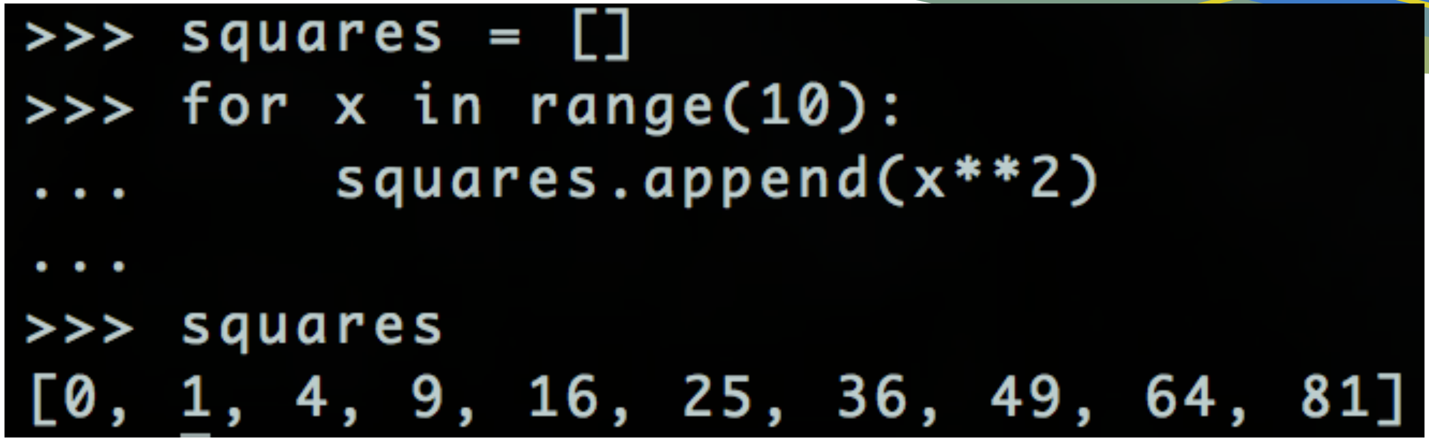
If we want to create a list that is a modified version of an existing list we usually do something like this:



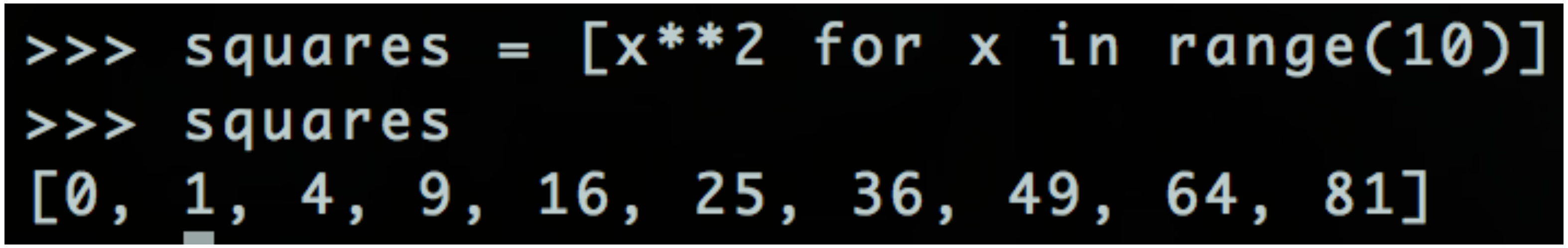
```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Python offers an easy alternative!

## List Comprehension

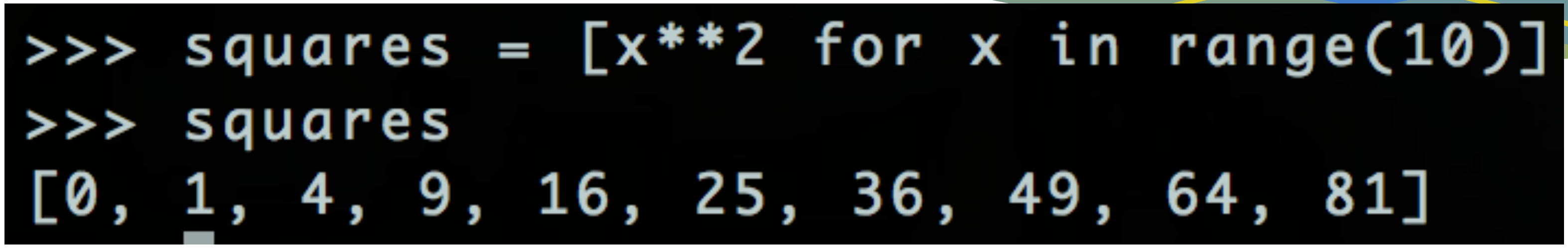


```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# List Comprehension

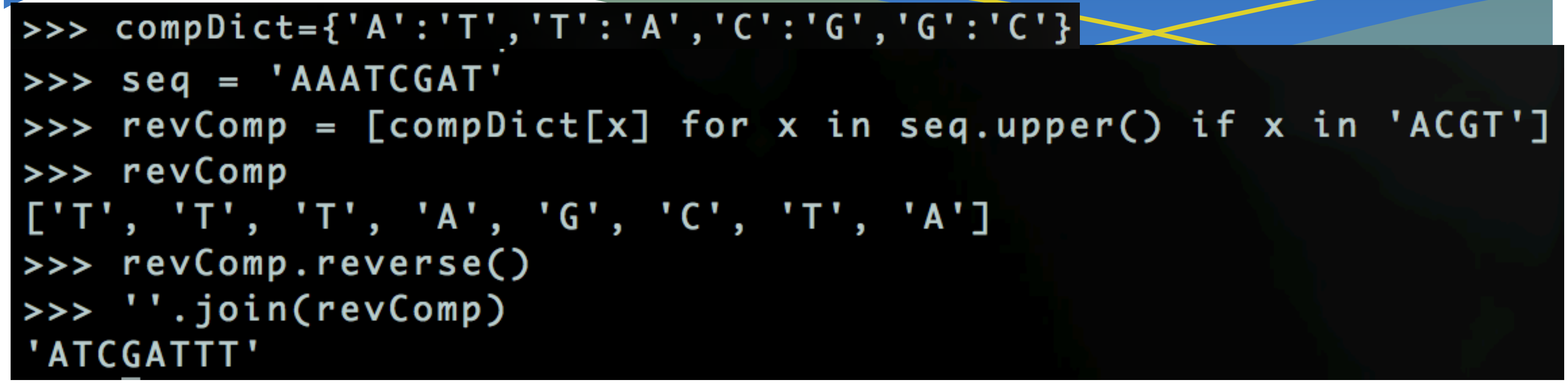


```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

To create a list this way:

newList = [expression **for** value **in** oldList]

# List Comprehension



```
>>> compDict={'A':'T','T':'A','C':'G','G':'C'}
>>> seq = 'AAATCGAT'
>>> revComp = [compDict[x] for x in seq.upper() if x in 'ACGT']
>>> revComp
['T', 'T', 'T', 'A', 'G', 'C', 'T', 'A']
>>> revComp.reverse()
>>> ''.join(revComp)
'ATCGATTT'
```

Reverse complement function we wrote previous in much less code!

Have to **reverse()** the list and then use a **string** method (**join**) to turn the list of characters into a **string**.

# Slicing Up a List (with step)

listName[start:end:step]

From 1st value to 6th, choosing every 3rd value.

From 2nd value to 9th value, choosing every 4th

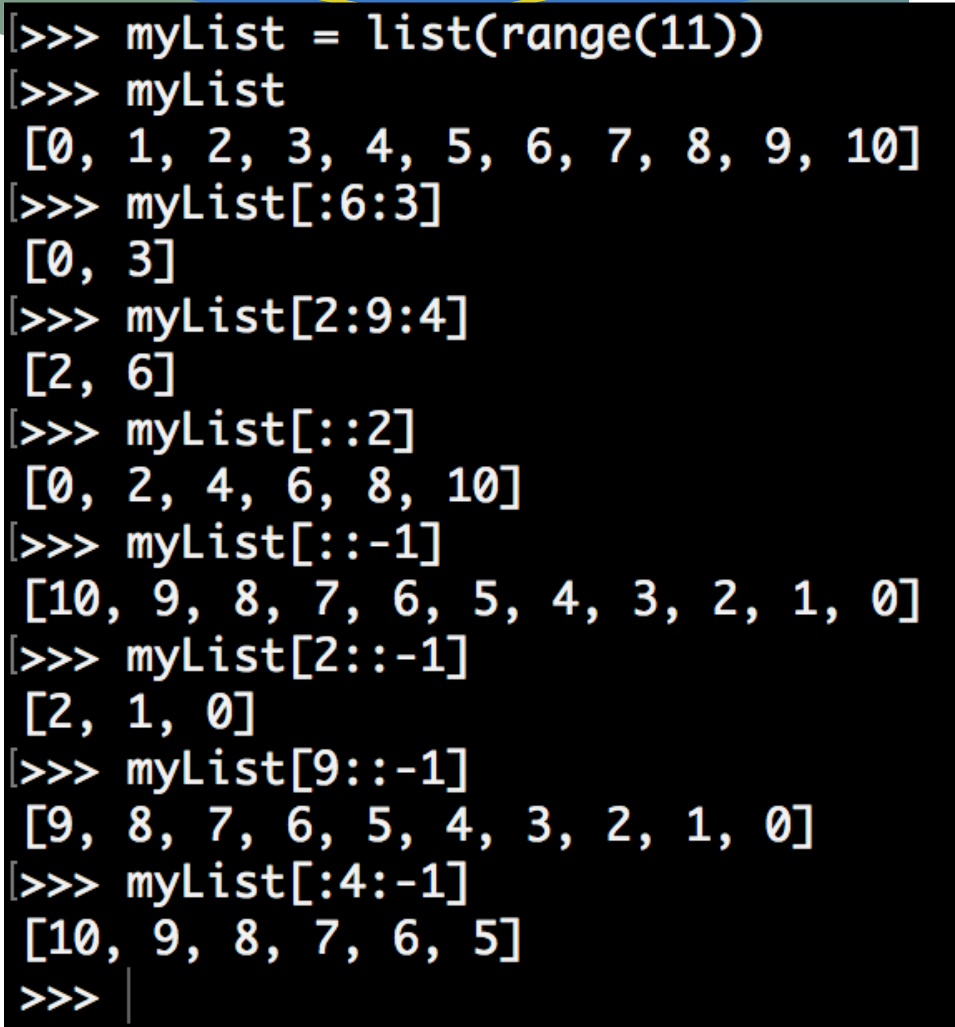
List with values with only pairwise index

Entire list, every value, in reverse

2nd value down to the beginning

9th value down to the beginning

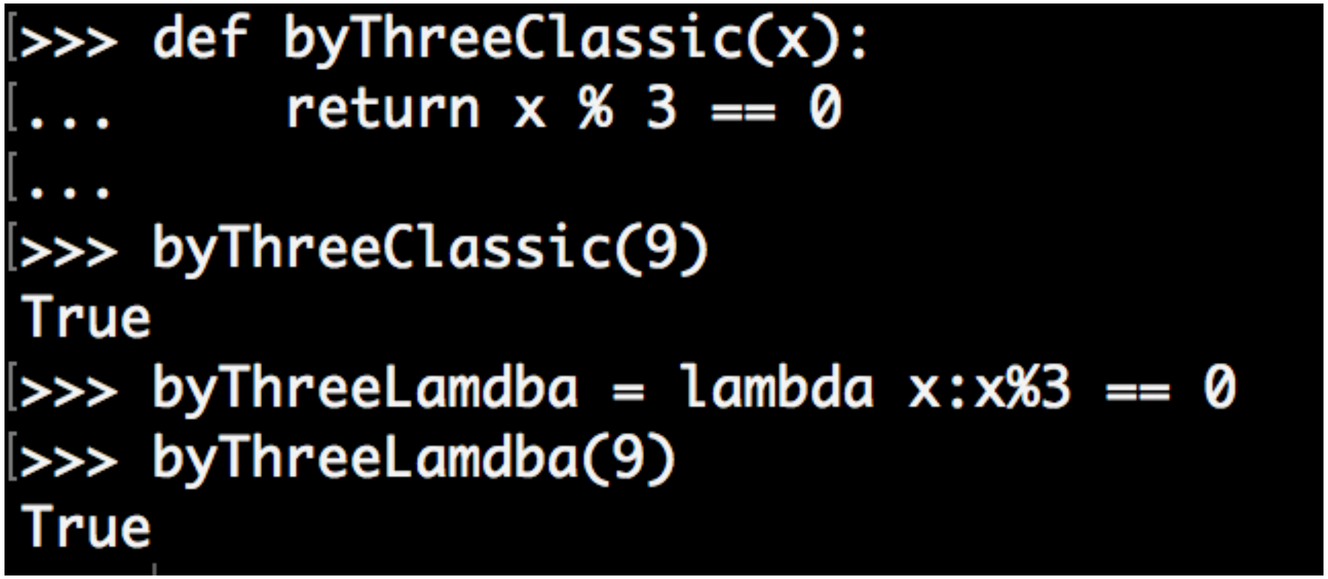
From beginning of list to 4th value, in reverse



```
>>> myList = list(range(11))
>>> myList
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> myList[:6:3]
[0, 3]
>>> myList[2:9:4]
[2, 6]
>>> myList[::2]
[0, 2, 4, 6, 8, 10]
>>> myList[::-1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> myList[2::-1]
[2, 1, 0]
>>> myList[9::-1]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
>>> myList[:4:-1]
[10, 9, 8, 7, 6, 5]
>>>
```

## Lambda functions

An alternative way to define a function.

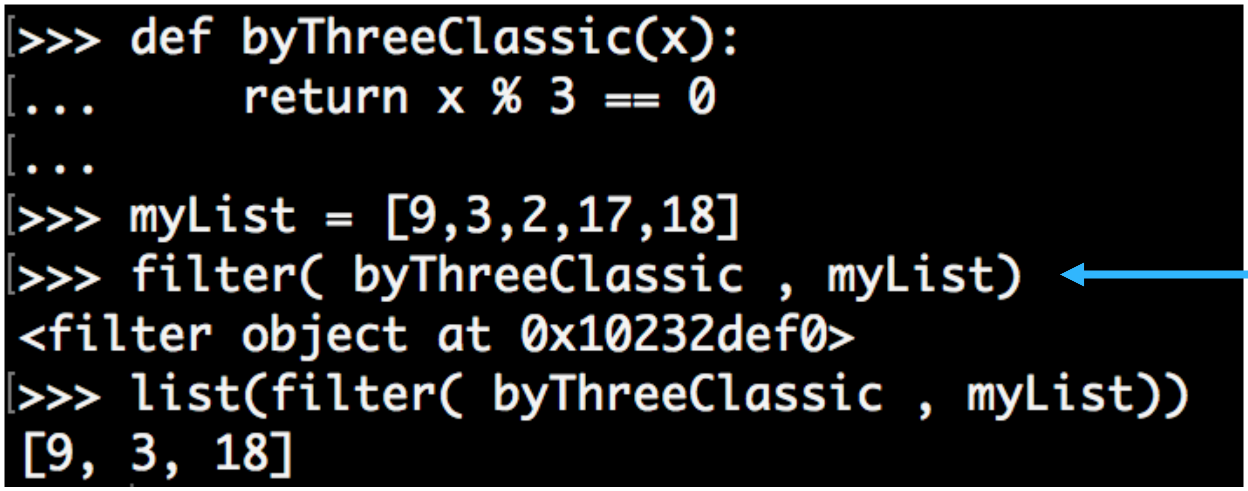


```
>>> def byThreeClassic(x):
...     return x % 3 == 0
...
>>> byThreeClassic(9)
True
>>> byThreeLamdba = lambda x:x%3 == 0
>>> byThreeLamdba(9)
True
```

More compact, but also useful in conjunction with other functions.

# Filters

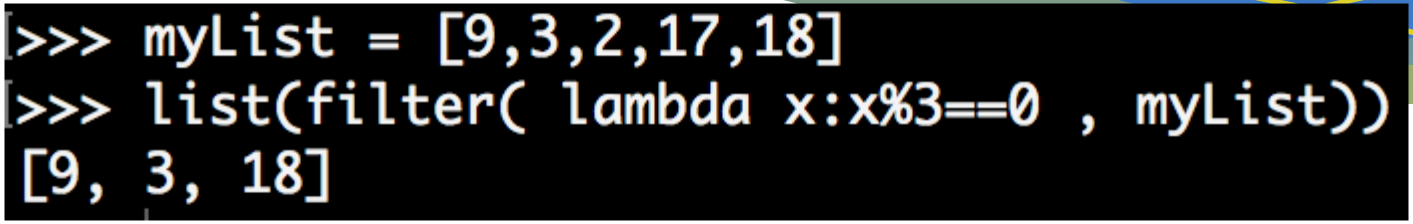
The function **filter(**function, list**)**



```
>>> def byThreeClassic(x):
...     return x % 3 == 0
...
>>> myList = [9,3,2,17,18]
>>> filter( byThreeClassic , myList)
<filter object at 0x10232def0>
>>> list(filter( byThreeClassic , myList))
[9, 3, 18]
```

Like for **range**, in Python2, **filter** would directly returns a list

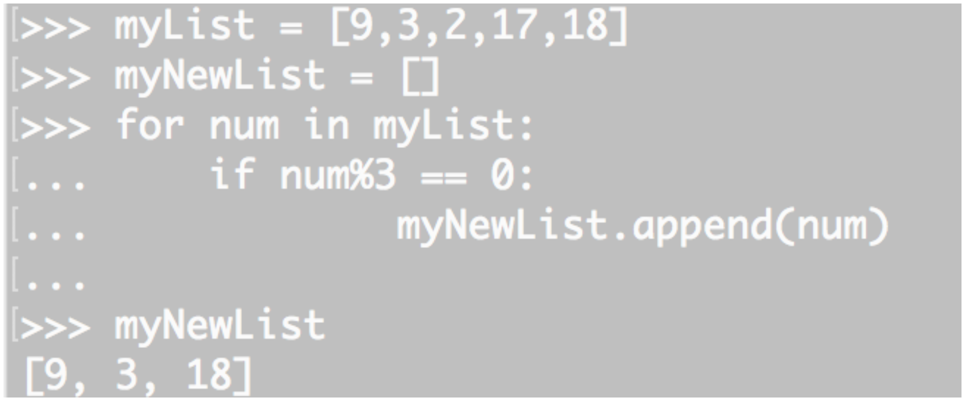
# Let's combine **lambda** and **filter**



```
>>> myList = [9,3,2,17,18]
>>> list(filter( lambda x:x%3==0 , myList))
[9, 3, 18]
```

* Only 1 line !
* To compare with the old fashioned way :

4 lines



```
>>> myList = [9,3,2,17,18]
>>> myNewList = []
>>> for num in myList:
...     if num%3 == 0:
...             myNewList.append(num)
...
>>> myNewList
[9, 3, 18]
```

## File Input/Output

## File Input.

Reading from a file is the main way of getting biological data into Python.

***fileVariable = open("fileName.txt" , "w")***

***fileVariable.read(size)***

size is optional and specifies how many bytes to read

***fileVariable.readLine()***

reads and returns a single line of the file

# File Output

Writing results to a file is useful for large data sets and for exporting to other programs to create graphs etc.

***fileVariable.write(string)***

writes the contents of *string* to the file.

***fileVariable.tell()***

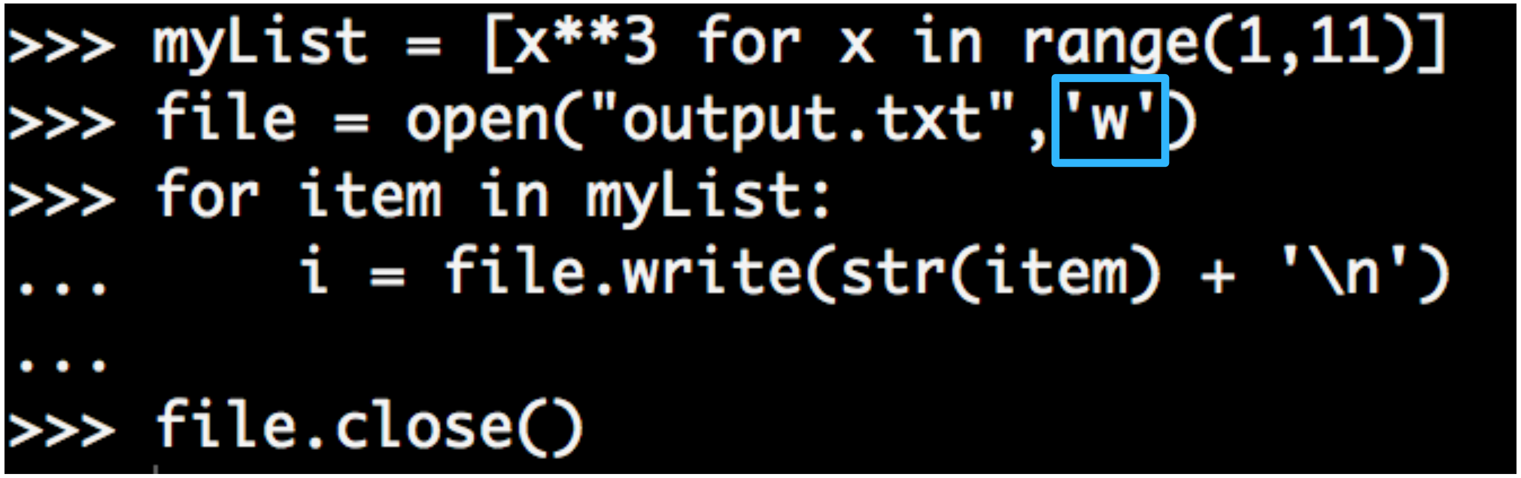
returns an integer value representing how far through the file you currently are, in bytes.

***fileVariable.seek(offset,0)***

change current position in file to *offset* bytes from the beginning. To offset from current position or end do *seek(offset,1)* or *seek(offset,2)* respectively.

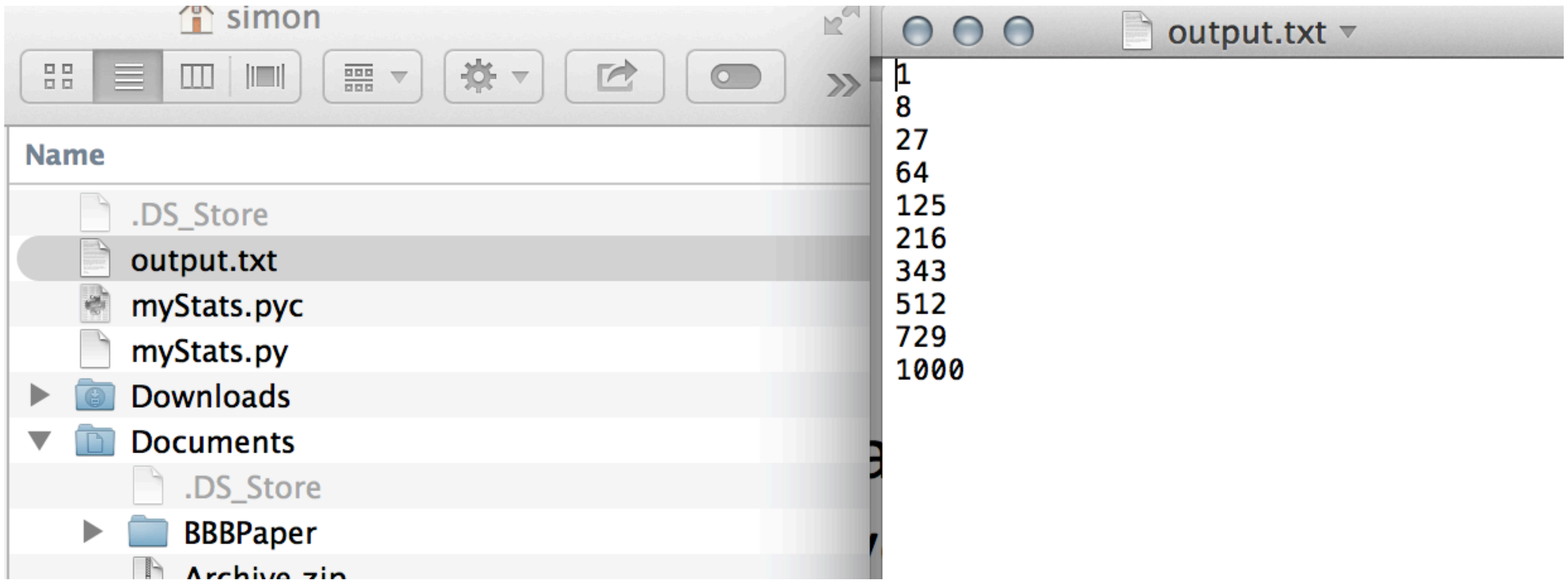
# File Input/Output Example.

* Write in a file:



```
>>> myList = [x**3 for x in range(1,11)]
>>> file = open("output.txt",'w')
>>> for item in myList:
...     i = file.write(str(item) + '\n')
...
>>> file.close()
```

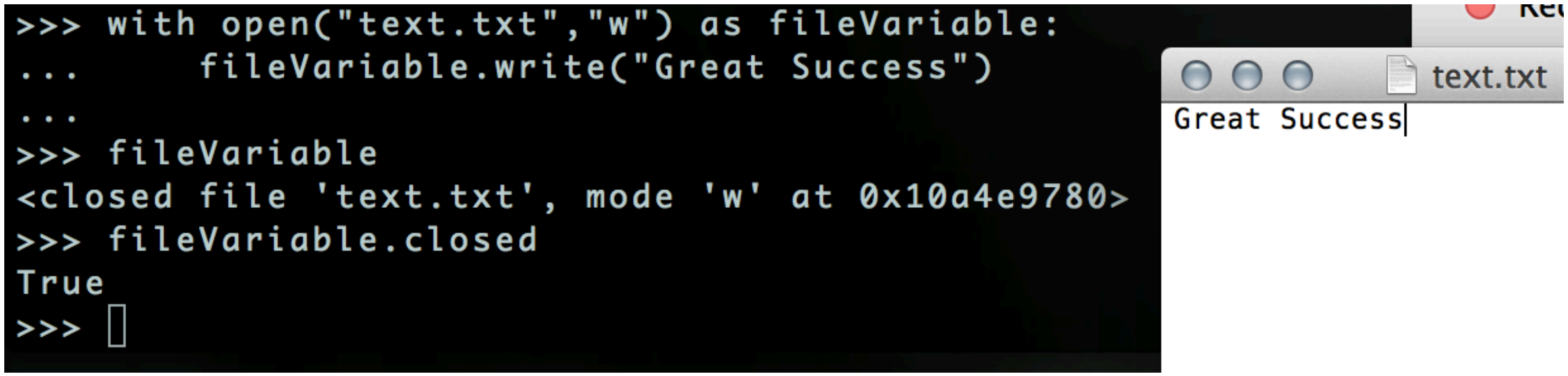
## File Output.



```
1
8
27
64
125
216
343
512
729
1000
```

## Always close() Files

It's important to close() a file when you have finished writing or reading from it.



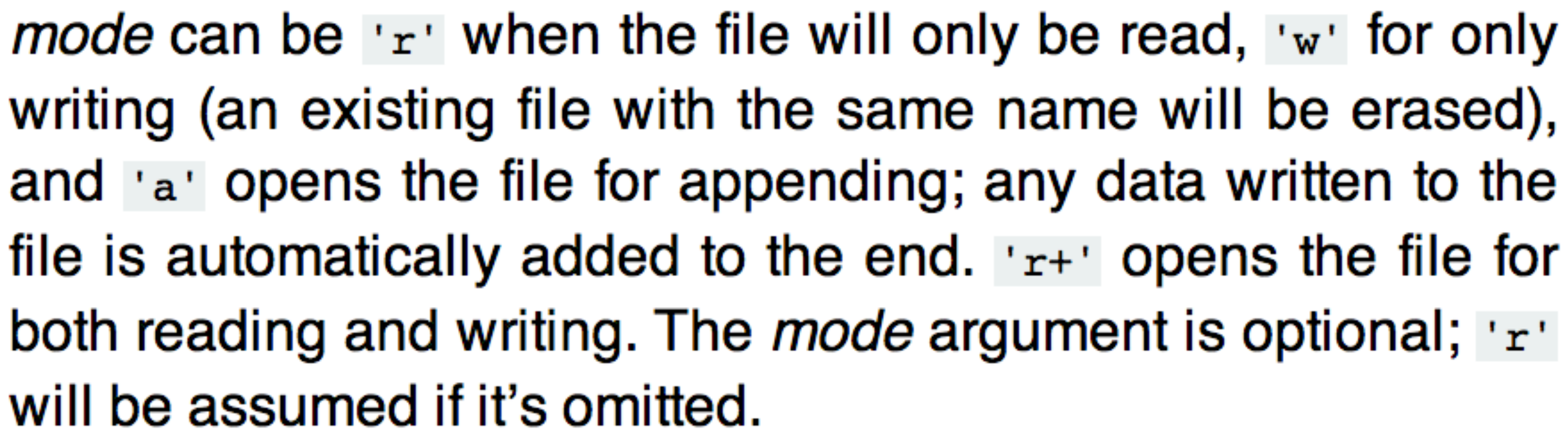
```
>>> with open("text.txt","w") as fileVariable:
...     fileVariable.write("Great Success")
...
>>> fileVariable
<closed file 'text.txt', mode 'w' at 0x10a4e9780>
>>> fileVariable.closed
True
>>> 
```

```
Great Success
```

Alternatively use **with open() as variable:** to automatically close the file after the code is executed.

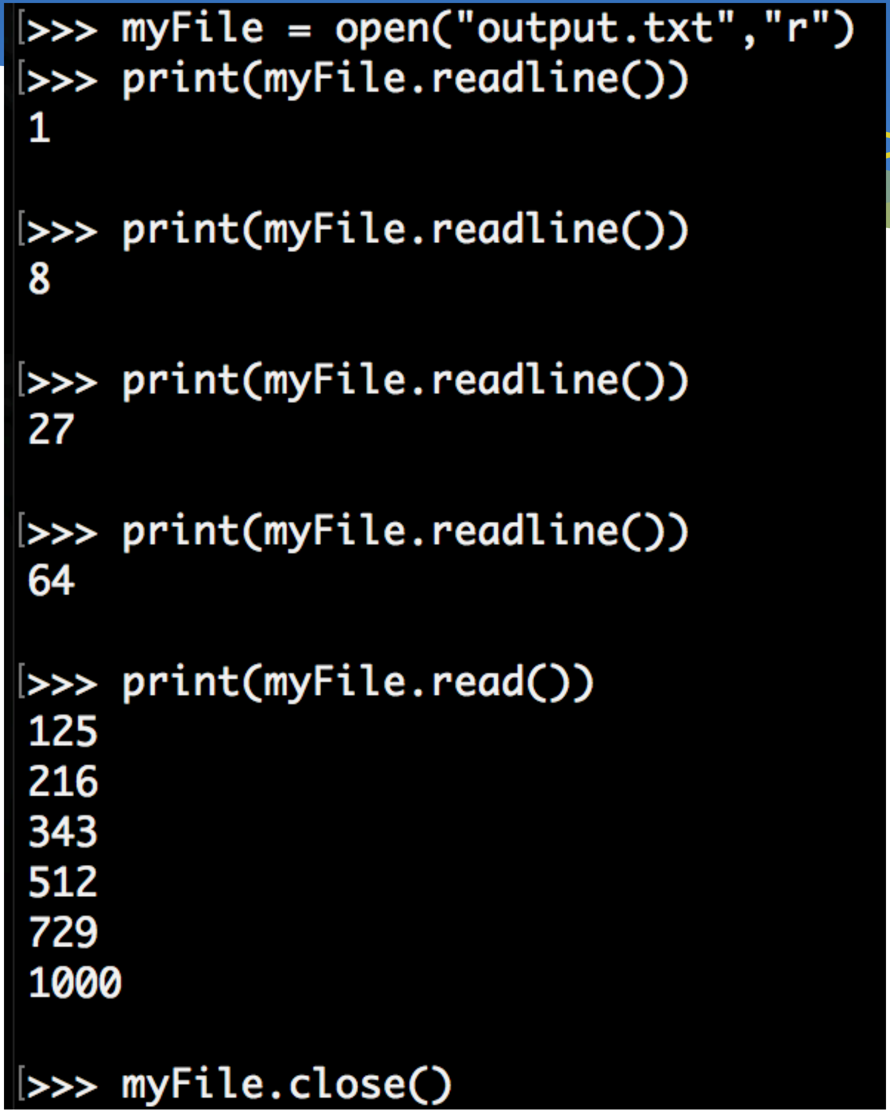
## File Mode

What does the “w” do in: Open(“fileName.txt”, “w”)



*mode* can be `'r'` when the file will only be read, `'w'` for only writing (an existing file with the same name will be erased), and `'a'` opens the file for appending; any data written to the file is automatically added to the end. `'r+'` opens the file for both reading and writing. The *mode* argument is optional; `'r'` will be assumed if it’s omitted.

## File Mode



```
>>> myFile = open("output.txt","r")
>>> print(myFile.readline())
1

>>> print(myFile.readline())
8

>>> print(myFile.readline())
27

>>> print(myFile.readline())
64

>>> print(myFile.read())
125
216
343
512
729
1000

>>> myFile.close()
```

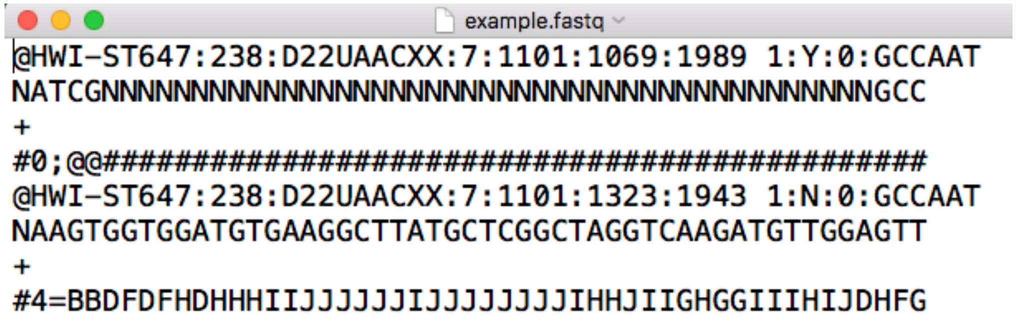
# fastQ file

## Contain reads for sequencing analysis.

A FASTQ file normally uses four lines per sequence.

- Line 1 begins with a '@' character and is followed by a sequence identifier and an *optional* description (like a FASTA title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a '+' character and is *optionally* followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

A FASTQ file containing a single sequence might look like this:



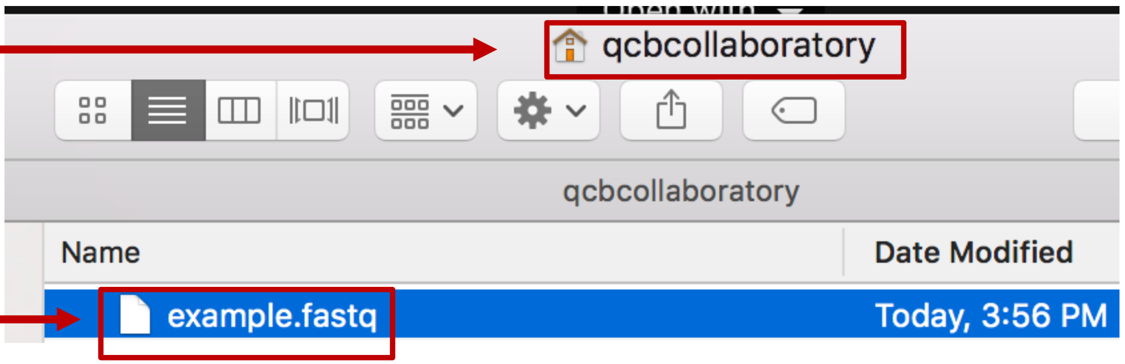
```
@HWI-ST647:238:D22UAACXX:7:1101:1069:1989 1:Y:0:GCCAAT
NATCGNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGCC
+
#0;@@#################################################
@HWI-ST647:238:D22UAACXX:7:1101:1323:1943 1:N:0:GCCAAT
NAAGTGGTGGATGTGAAGGCTTATGCTCGGCTAGGTCAAGATGTTGGAGTT
+
#4=BBDFDFHDHHHIIJJJJJJIJJJJJJJJIHHJIIGHGGIIIHIJDHFG
```

# fastQ file

Download the fastQ file at : https://goo.gl/tYYftm

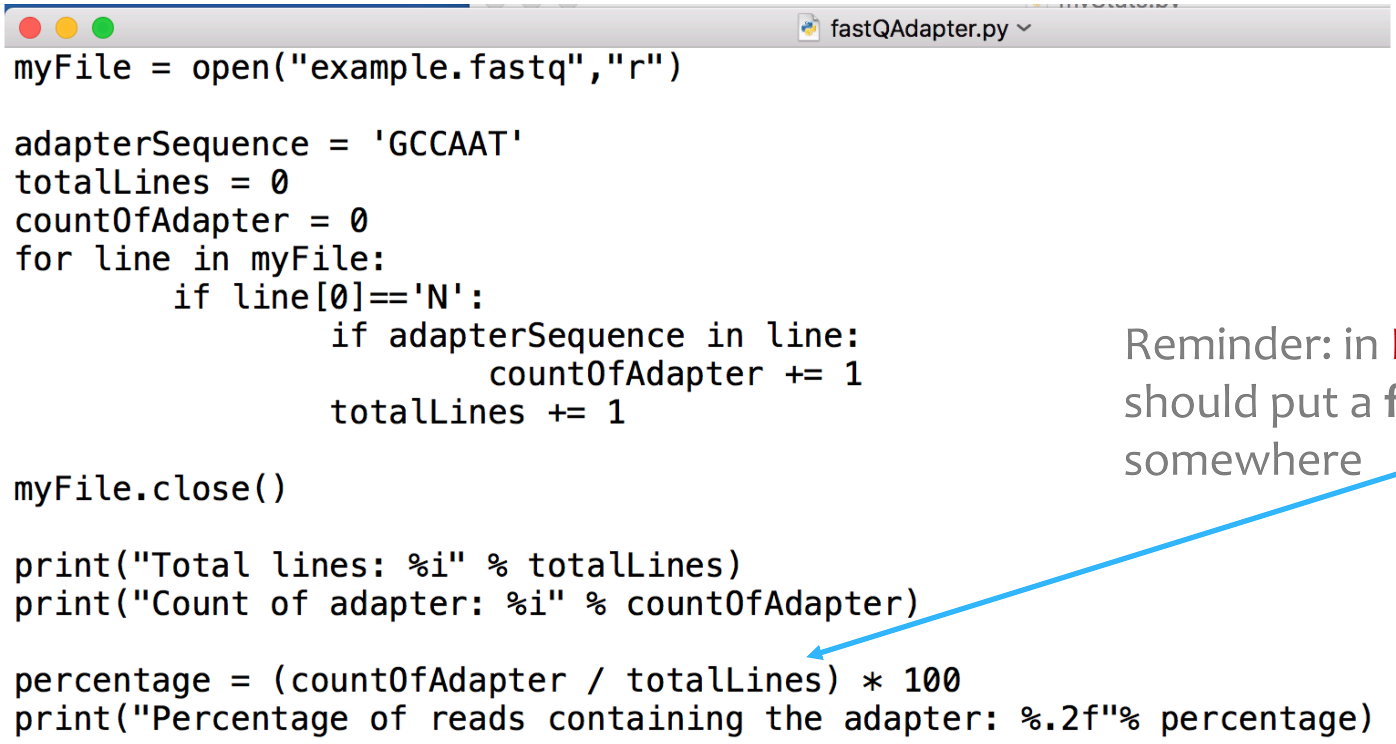
In **home** directory

**.fastq** extension



# fastQ Example

## Code to find which reads contain an adapter sequence

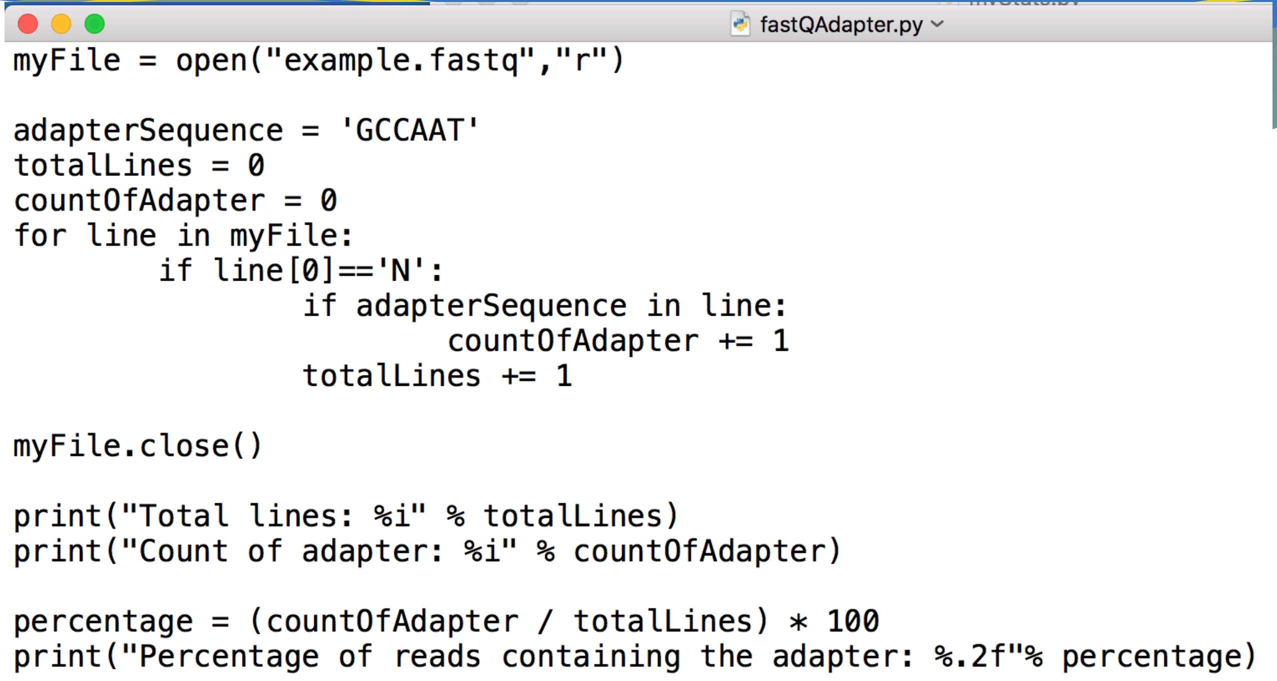


```
myFile = open("example.fastq","r")

adapterSequence = 'GCCAAT'
totalLines = 0
countOfAdapter = 0
for line in myFile:
        if line[0]=='N':
                if adapterSequence in line:
                        countOfAdapter += 1
                totalLines += 1

myFile.close()

print("Total lines: %i" % totalLines)
print("Count of adapter: %i" % countOfAdapter)

percentage = (countOfAdapter / totalLines) * 100
print("Percentage of reads containing the adapter: %.2f"% percentage)
```

Reminder: in Python2, should put a **float** somewhere

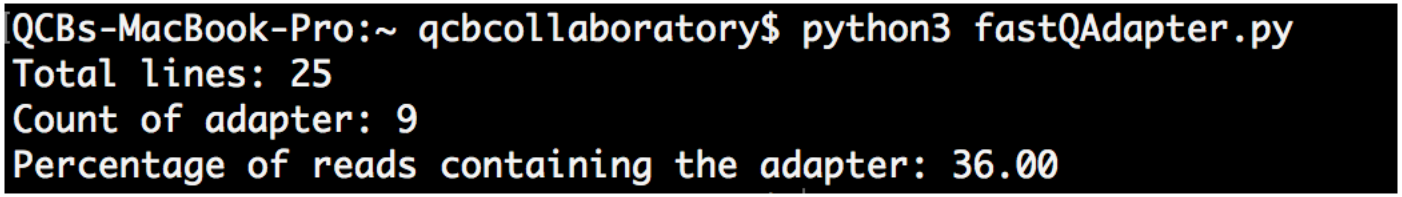
# fastQ Example

Let’s test it!



fastQAdapter.py

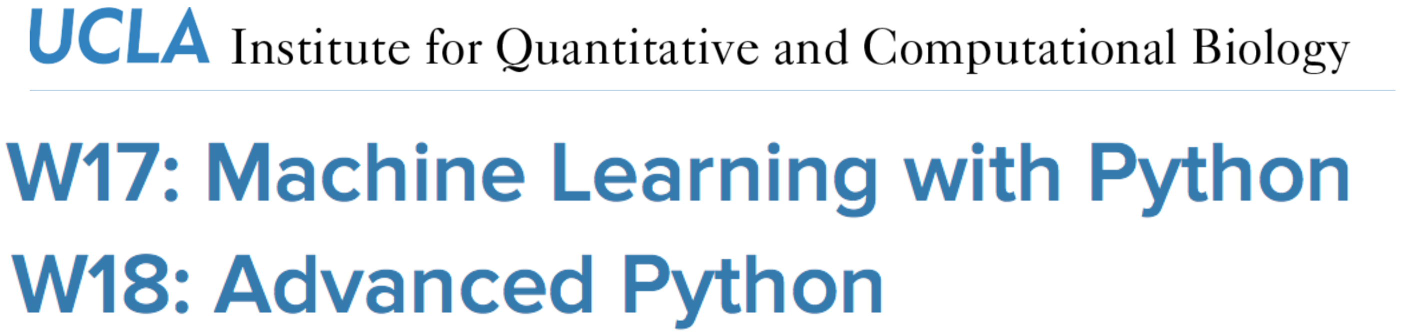
```
myFile = open("example.fastq","r")

adapterSequence = 'GCCAAT'
totalLines = 0
countOfAdapter = 0
for line in myFile:
        if line[0]=='N':
                if adapterSequence in line:
                        countOfAdapter += 1
                totalLines += 1

myFile.close()

print("Total lines: %i" % totalLines)
print("Count of adapter: %i" % countOfAdapter)

percentage = (countOfAdapter / totalLines) * 100
print("Percentage of reads containing the adapter: %.2f"% percentage)
```



```
QCBs-MacBook-Pro:~ qcbcollaboratory$ python3 fastQAdapter.py
Total lines: 25
Count of adapter: 9
Percentage of reads containing the adapter: 36.00
```

## To continue with Python:

* Other workshops



UCLA Institute for Quantitative and Computational Biology

W17: Machine Learning with Python

W18: Advanced Python

* http://www.codecademy.com/en/tracks/python

# Thanks!

Before you leave please fill out the survey, it really helps us and only has a few tick-boxes:

**Surveymonkey.com/r/PythonJun2016**

you leave please