An Introduction to Python

Day 3

Simon Mitchell

Simon.Mitchell@ucla.edu

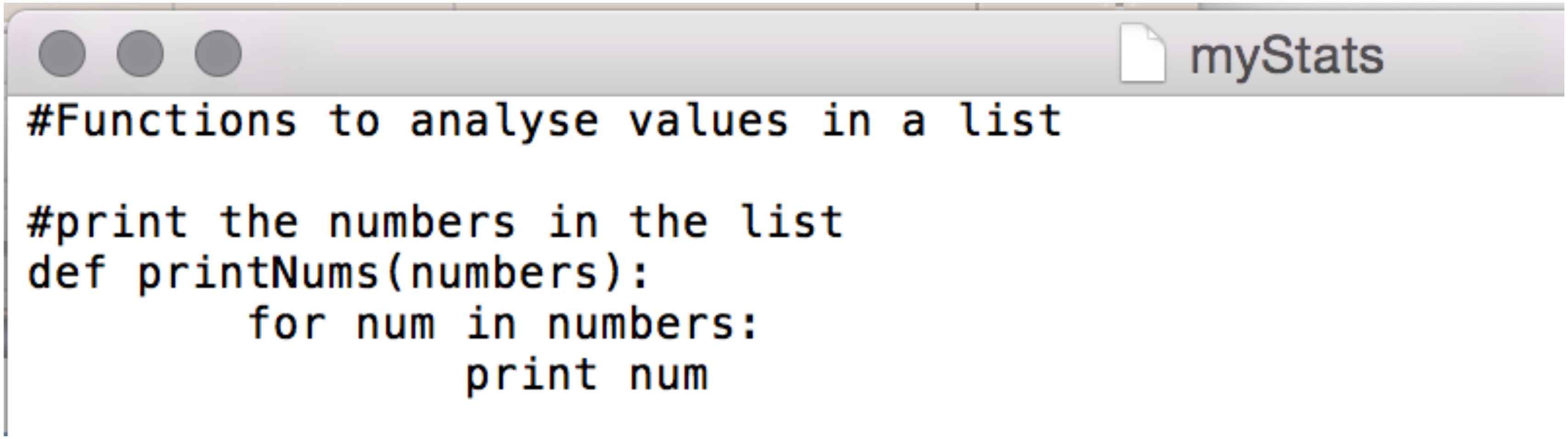
Combining what we’ve learnt

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:

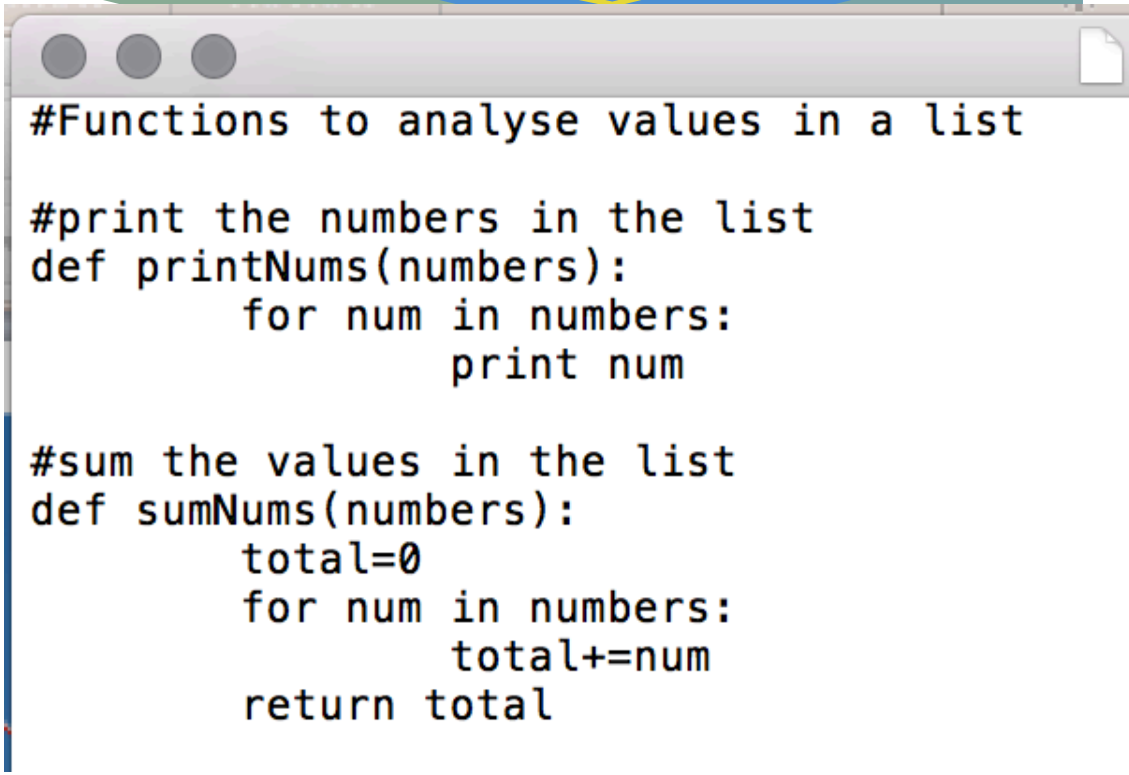


```
#Functions to analyse values in a list

#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

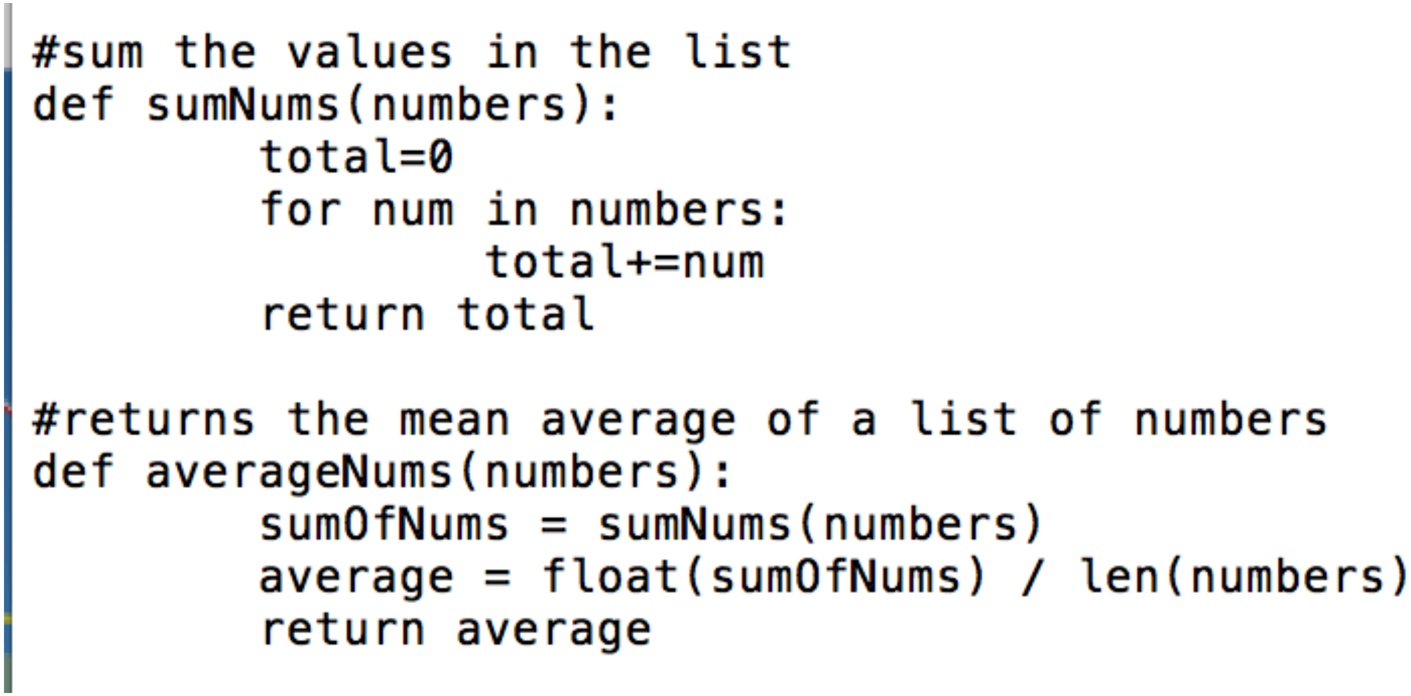
- A function to sum values:



```
#Functions to analyse 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:

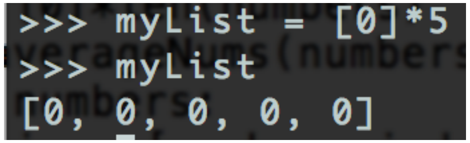


```
#sum the values in the list
def sumNums(numbers):
        total=0
        for num in numbers:
                total+=num
        return total

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

myStats.py

- A function to calculate the variance:

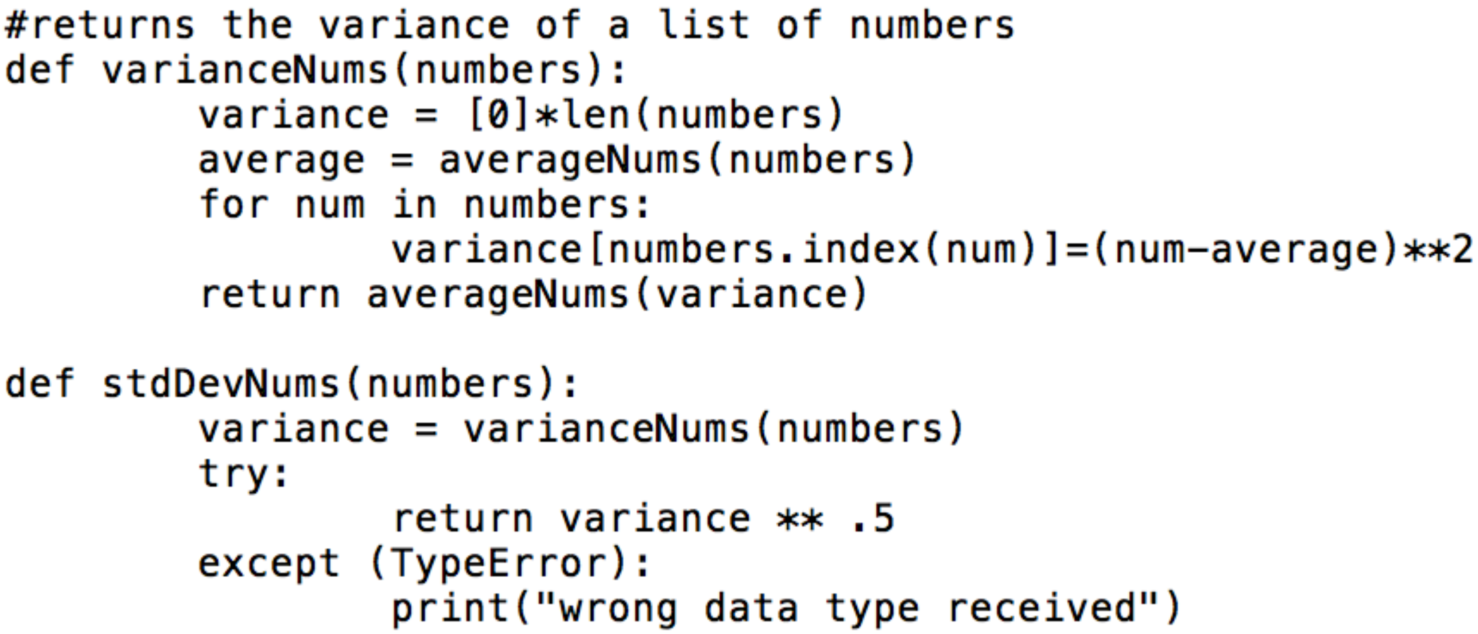
```
#returns the mean average of a list of numbers
def averageNums(numbers):
        sumOfNums = sumNums(numbers)
        average = float(sumOfNums) / len(numbers)
        return average

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



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

myStats.py

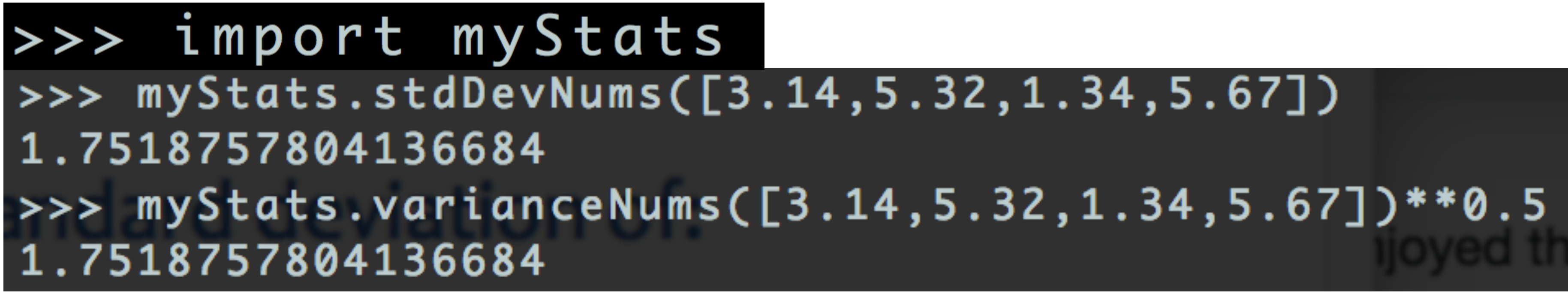
- A function to calculate the population standard deviation using the variance



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

def stdDevNums(numbers):
        variance = varianceNums(numbers)
        try:
                return variance ** .5
        except (TypeError):
                print("wrong data type received")
```

myStats.py

- Test it

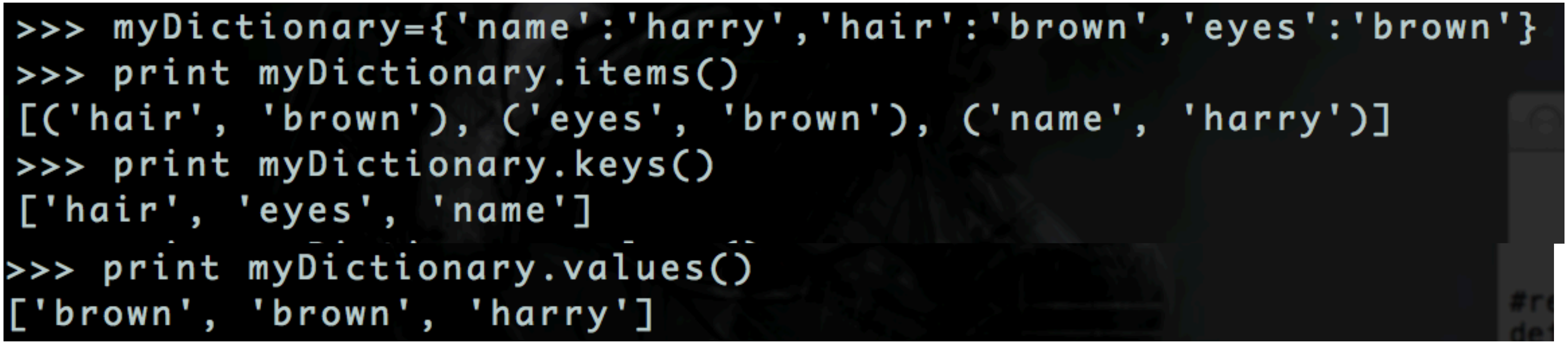


```
>>> import myStats
>>> myStats.stdDevNums([3.14,5.32,1.34,5.67])
1.7518757804136684
>>> myStats.varianceNums([3.14,5.32,1.34,5.67])**0.5
1.7518757804136684
```

Quiz Time:
What Is the average and standard deviation of:
[3.14, 5.32, 1.34, 5.67]

More Dictionary Methods

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

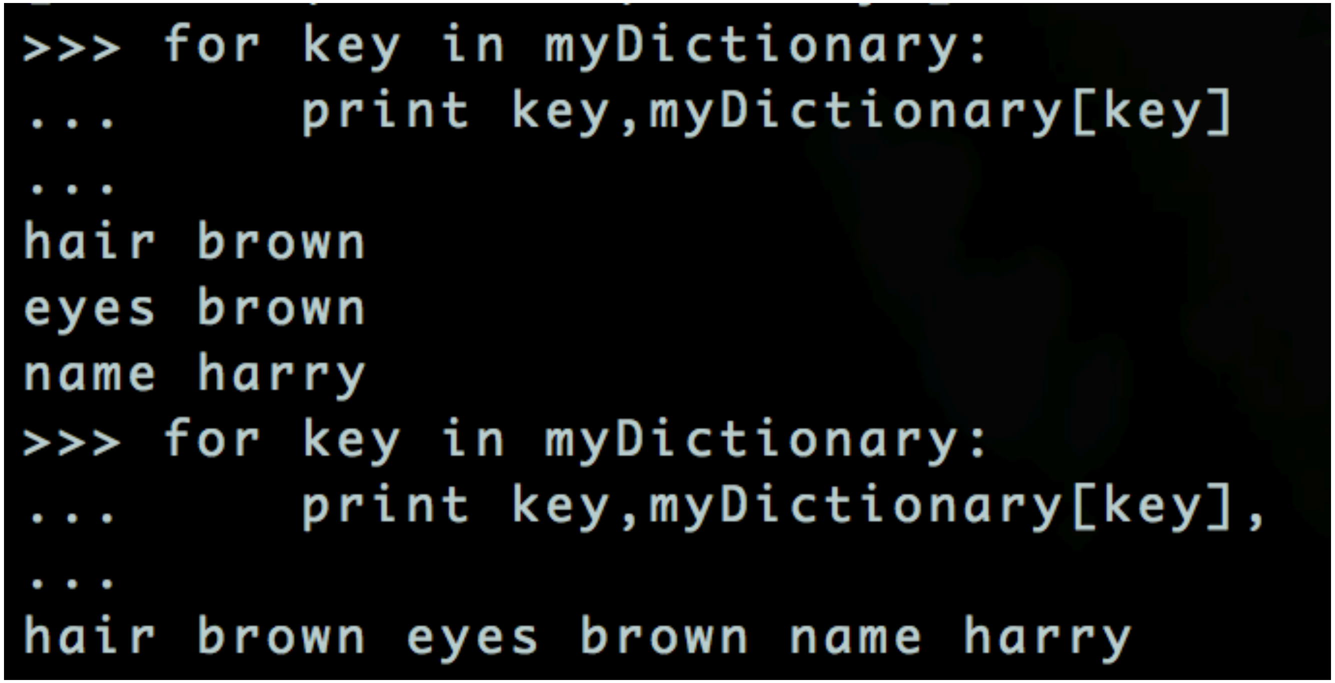


```
>>> myDictionary={'name':'harry','hair':'brown','eyes':'brown'}
>>> print myDictionary.items()
[('hair', 'brown'), ('eyes', 'brown'), ('name', 'harry')]
>>> print myDictionary.keys()
['hair', 'eyes', 'name']
>>> print myDictionary.values()
['brown', 'brown', 'harry']
```

- This is useful so we can iterate over dictionaries more easily…

Iterating over dictionaries

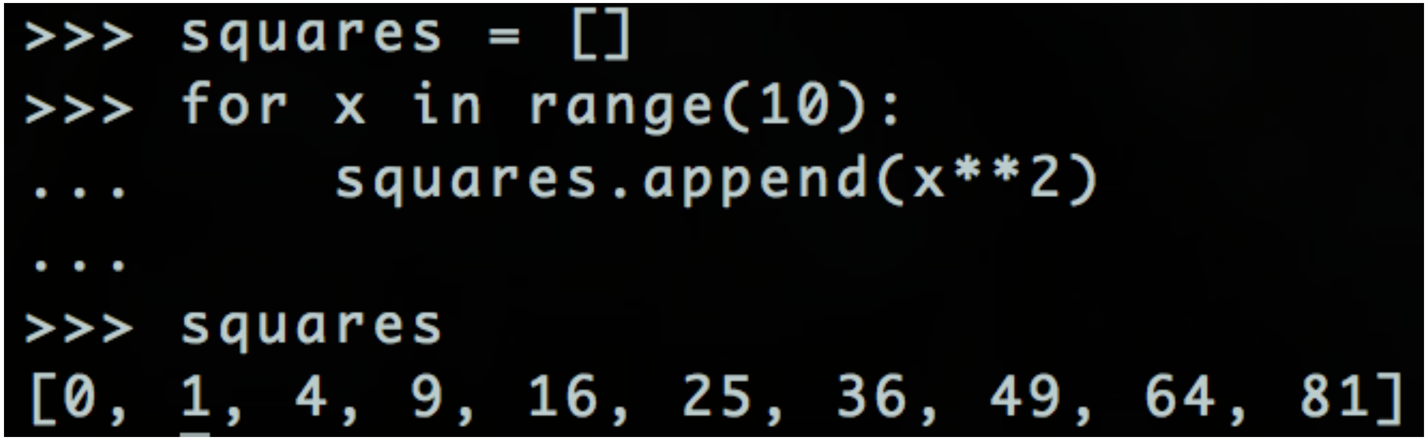
The comma means “on the same line”:



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

List Comprehension

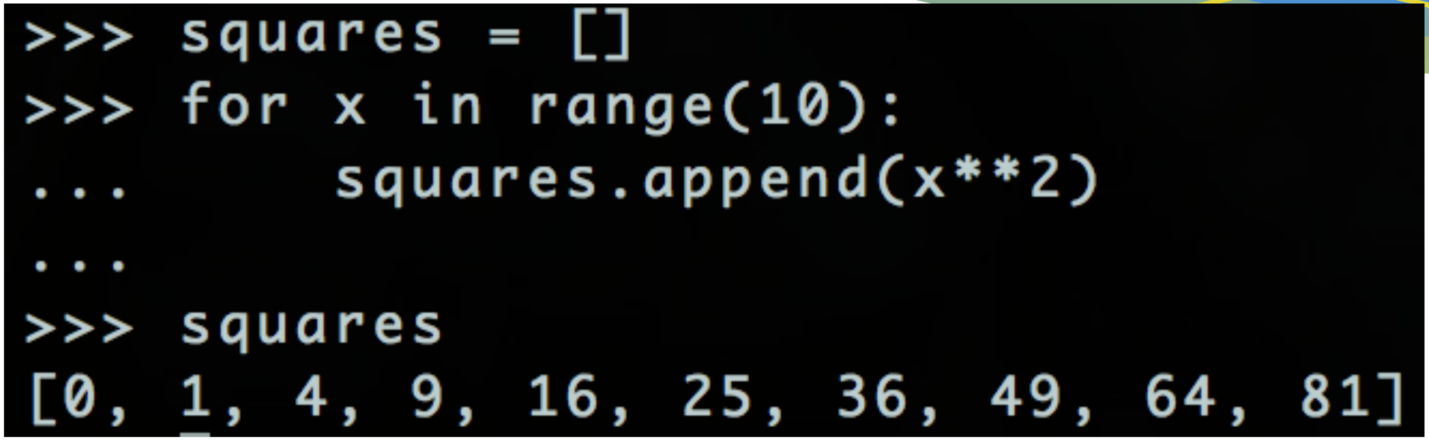
If we want to create a list that is a modified version of an existing list we usually do something like this:



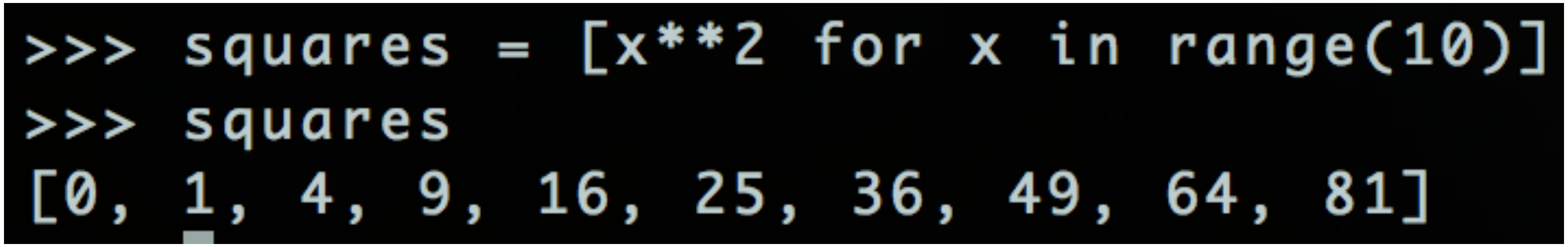
```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Python offers an easy alternative!

List Comprehension

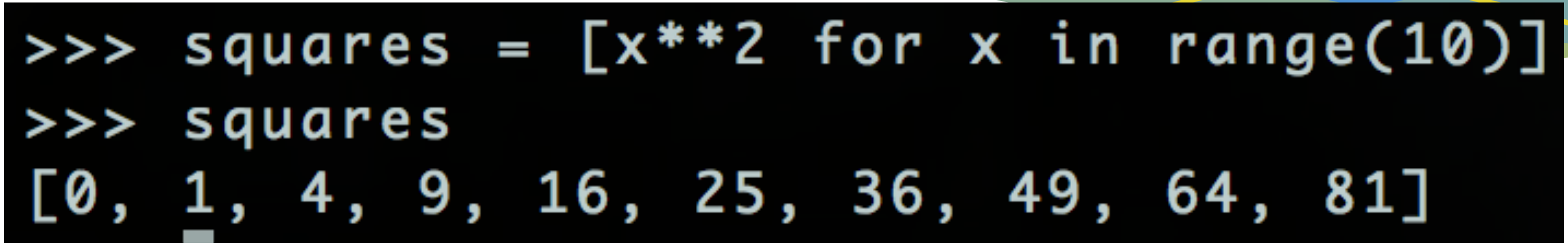


```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



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

List Comprehension

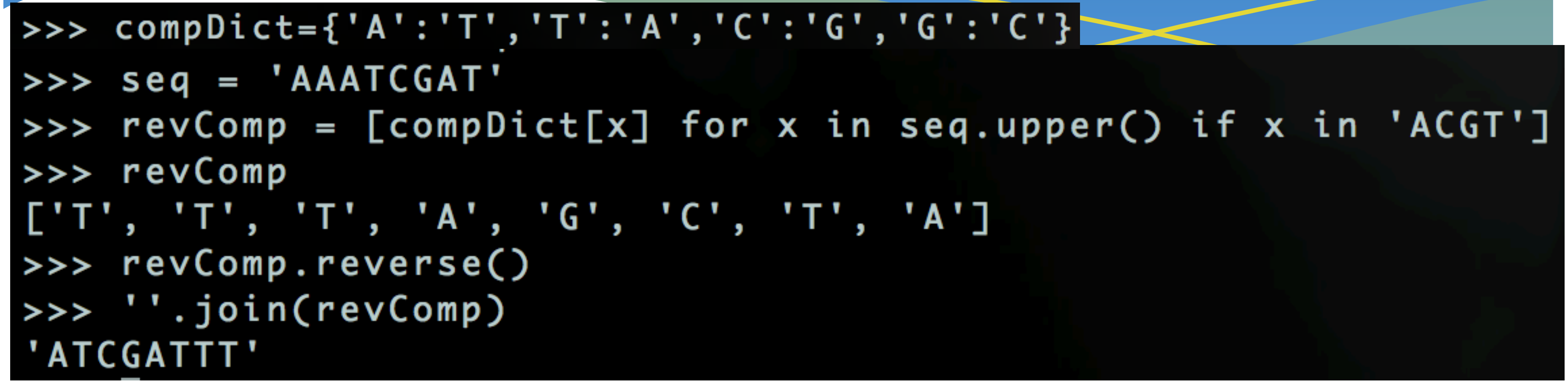


```
>>> squares = [x**2 for x in range(10)]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

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 Stride)

listName[start:end:stride]

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

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

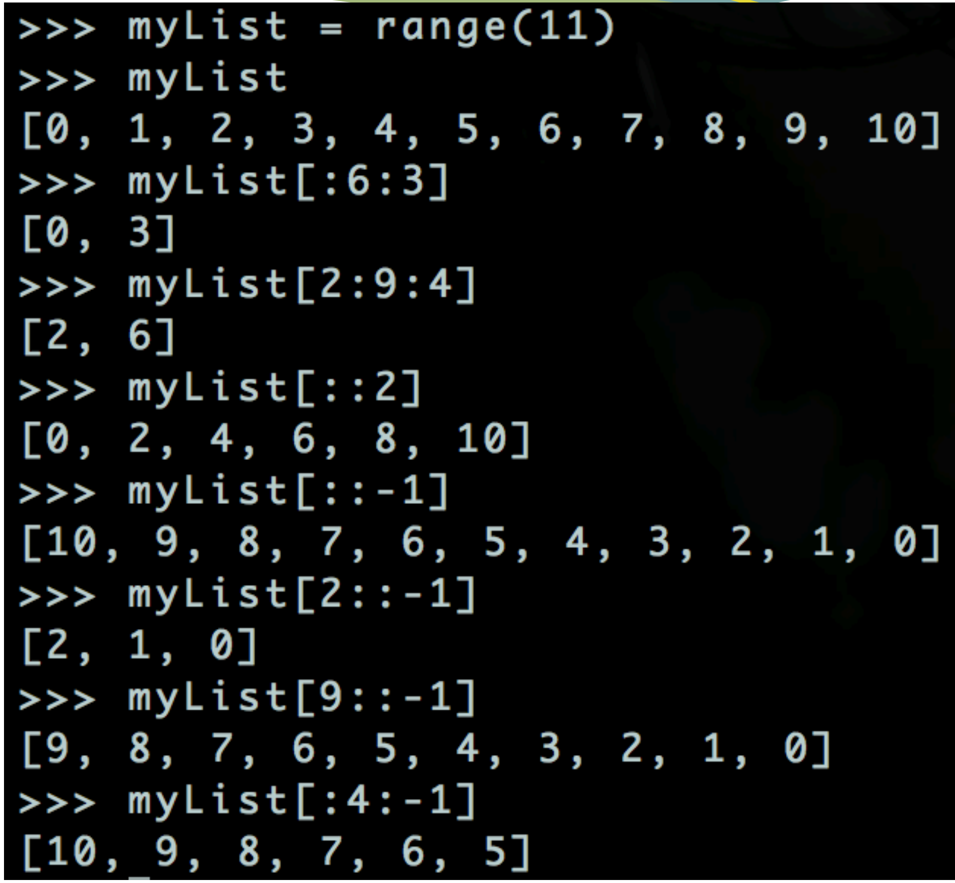
Entire list, every other value

Entire list, every value, in reverse

2nd value to 1st, don't skip any

9th value to end of list, in reverse

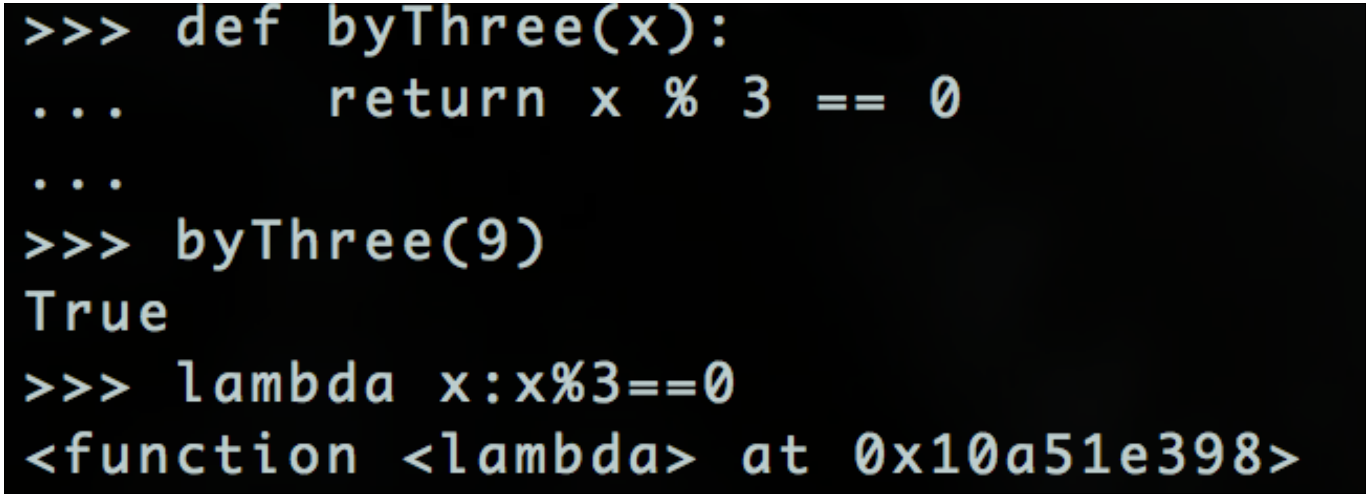
From beginning of list to 4th value, in reverse



```
>>> myList = 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.

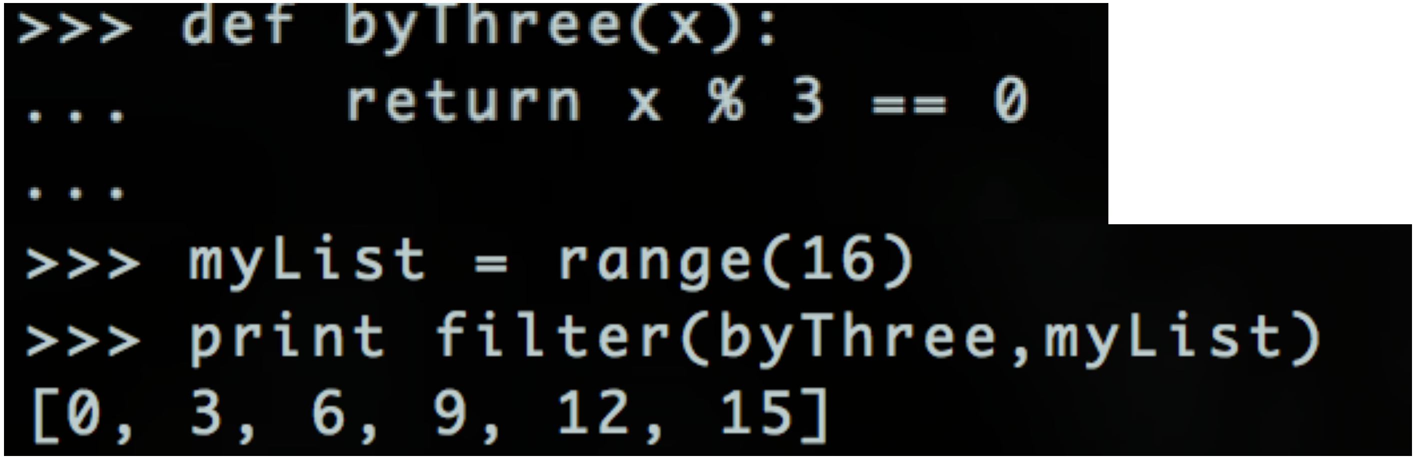


```
>>> def byThree(x):
...     return x % 3 == 0
...
>>> byThree(9)
True
>>> lambda x:x%3==0
<function <lambda> at 0x10a51e398>
```

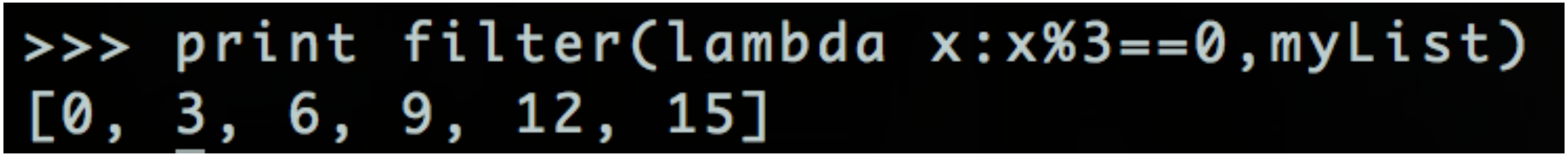
Not useful on its own but for use in conjunction with other functions!

Filters (use lambda functions!)

filter(function, list)



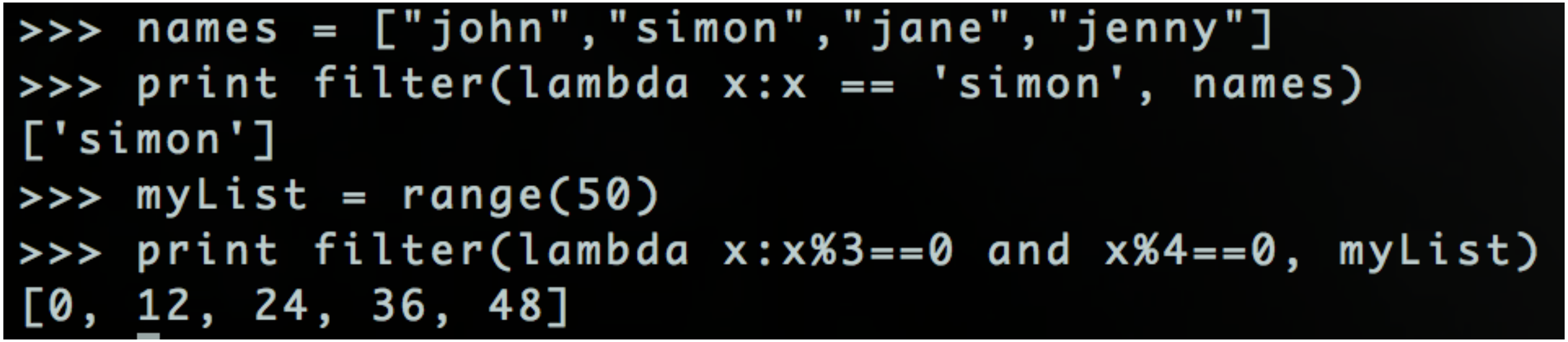
```
>>> def byThree(x):
...     return x % 3 == 0
...
>>> myList = range(16)
>>> print filter(byThree,myList)
[0, 3, 6, 9, 12, 15]
```



```
>>> print filter(lambda x:x%3==0,myList)
[0, 3, 6, 9, 12, 15]
```

Filters (use lambda functions!)

filter(function, list)



```
>>> names = ["john","simon","jane","jenny"]
>>> print filter(lambda x:x == 'simon', names)
['simon']
>>> myList = range(50)
>>> print filter(lambda x:x%3==0 and x%4==0, myList)
[0, 12, 24, 36, 48]
```

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.

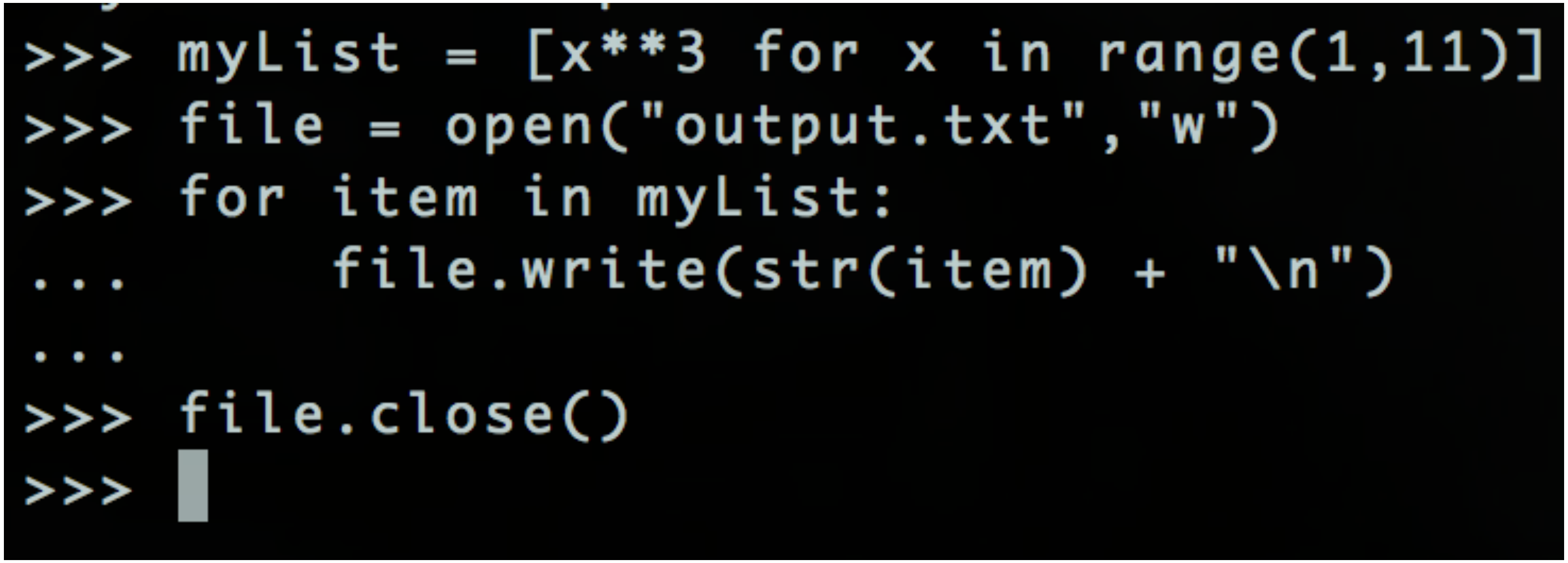
fileVariable.tell()

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

fileVariable.seek(offset,0)

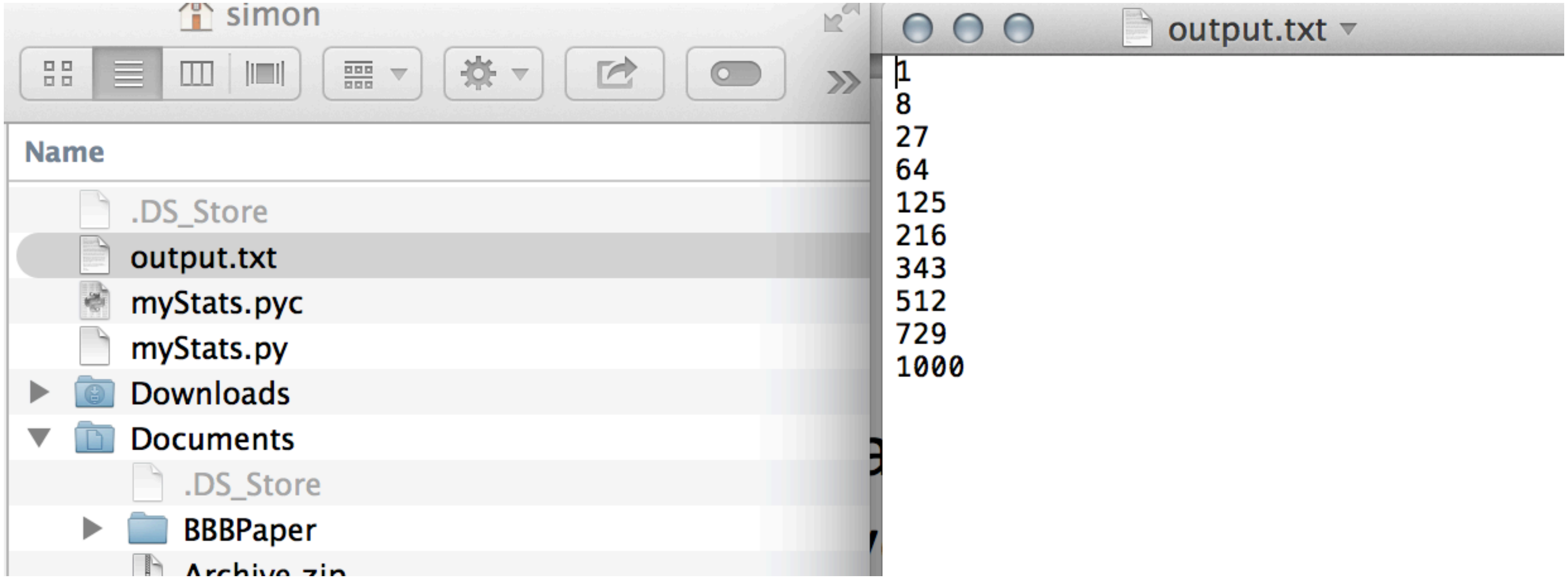
change current position in file to *offset* bytes from the beginning. To offset from current position or end do *seek(offset,1)* or *seek(offset,2)* respectively.

File Input/Output Example.



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

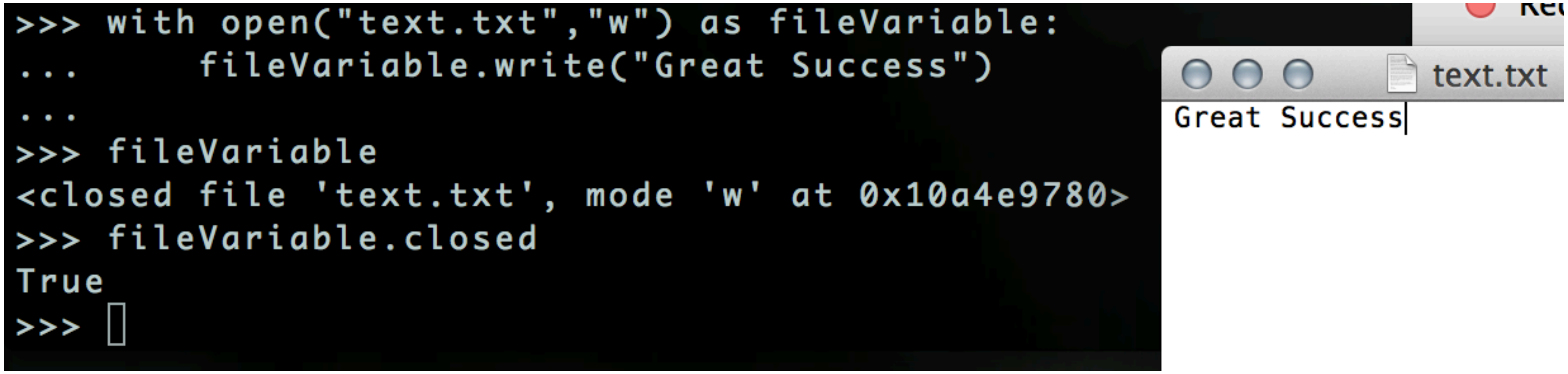
File Output.



```
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.



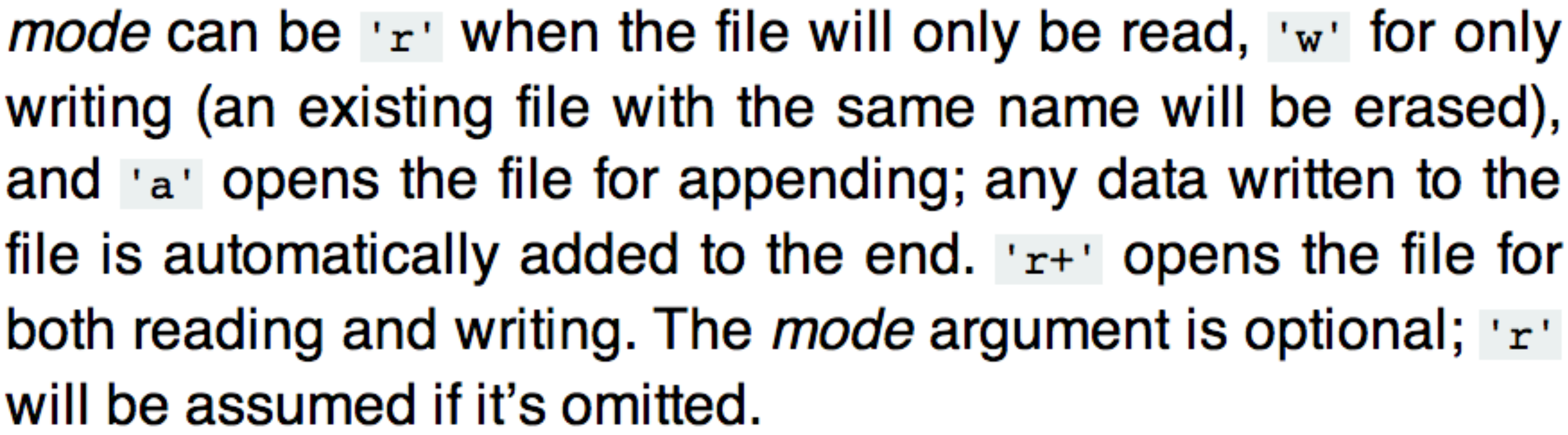
```
>>> with open("text.txt","w") as fileVariable:
...     fileVariable.write("Great Success")
...
>>> fileVariable
<closed file 'text.txt', mode 'w' at 0x10a4e9780>
>>> fileVariable.closed
True
>>> 
```

Great Success

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

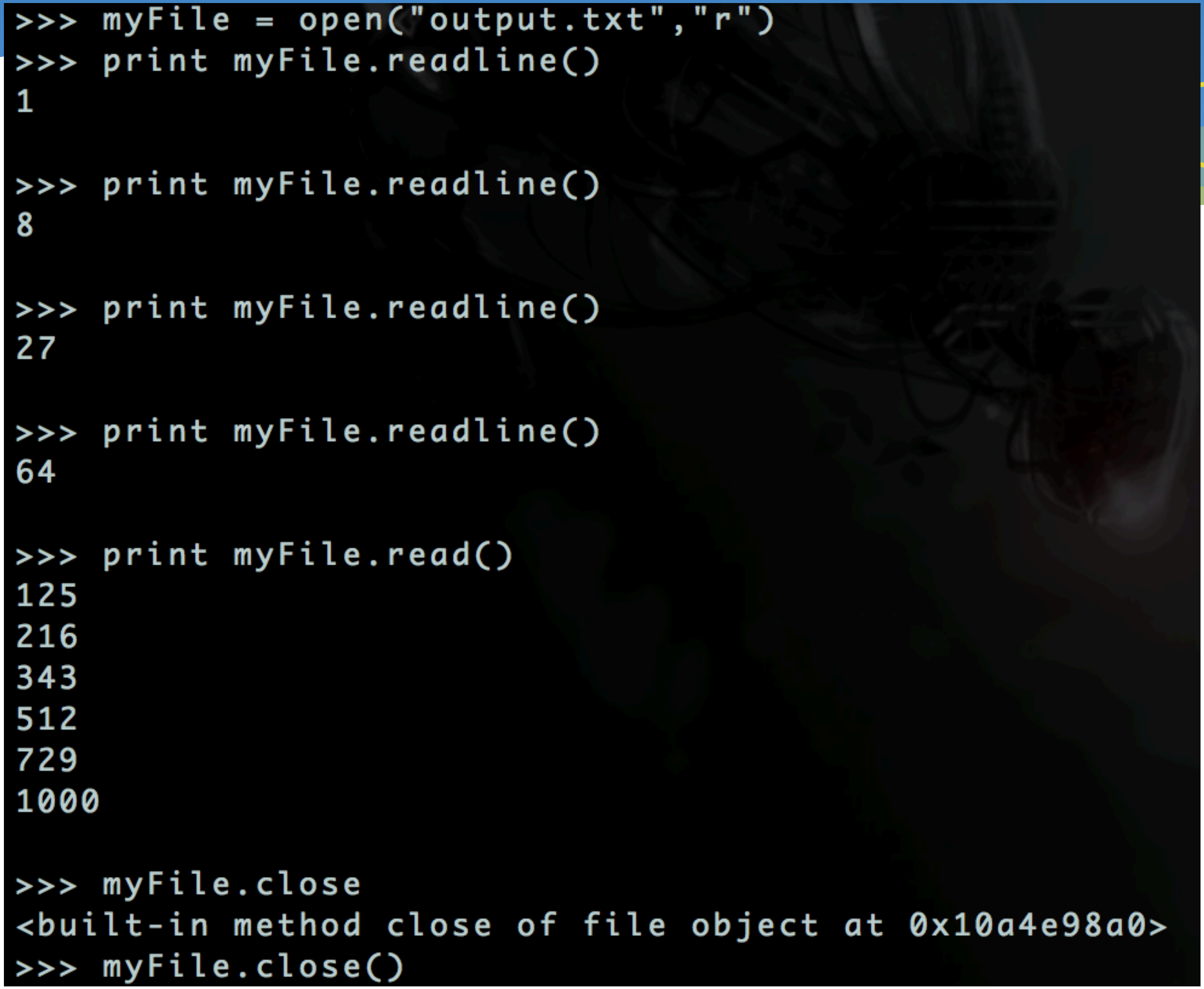
File Mode

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.

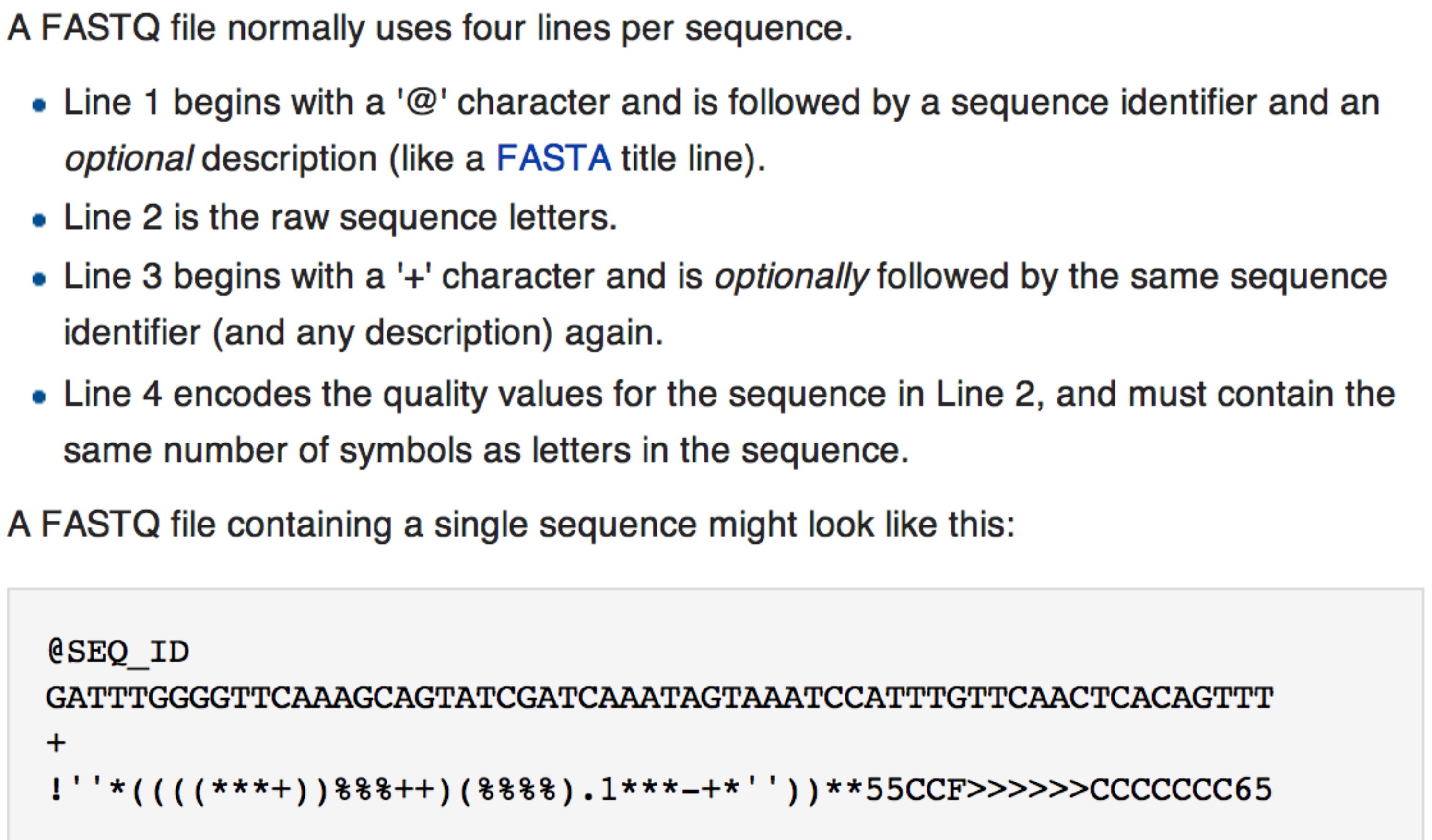
File Mode



```
>>> 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
<built-in method close of file object at 0x10a4e98a0>
>>> 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:

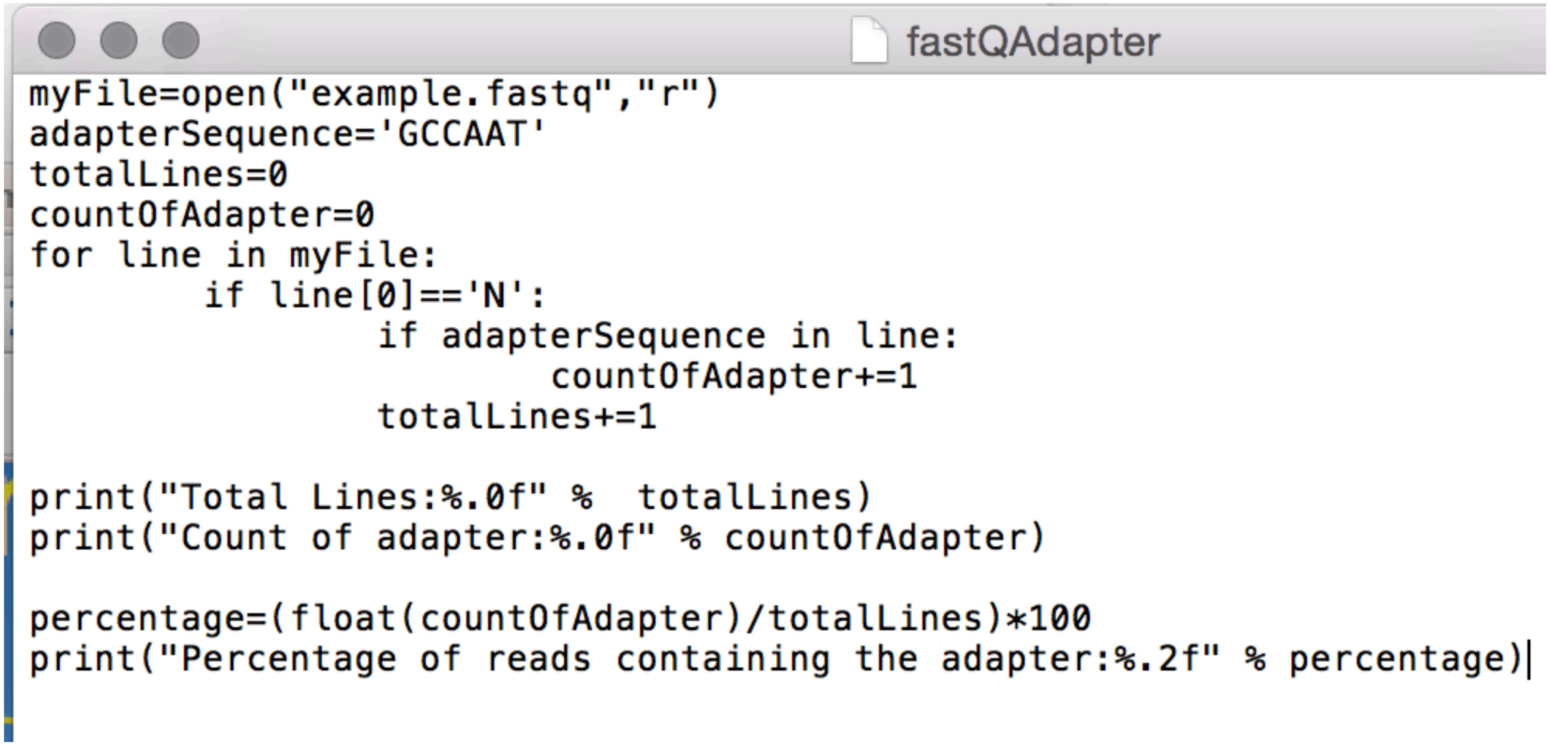
```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

fastQ file

www.signalingsystems.ucla.edu/users/Simon/example.fastq

fastQ Example

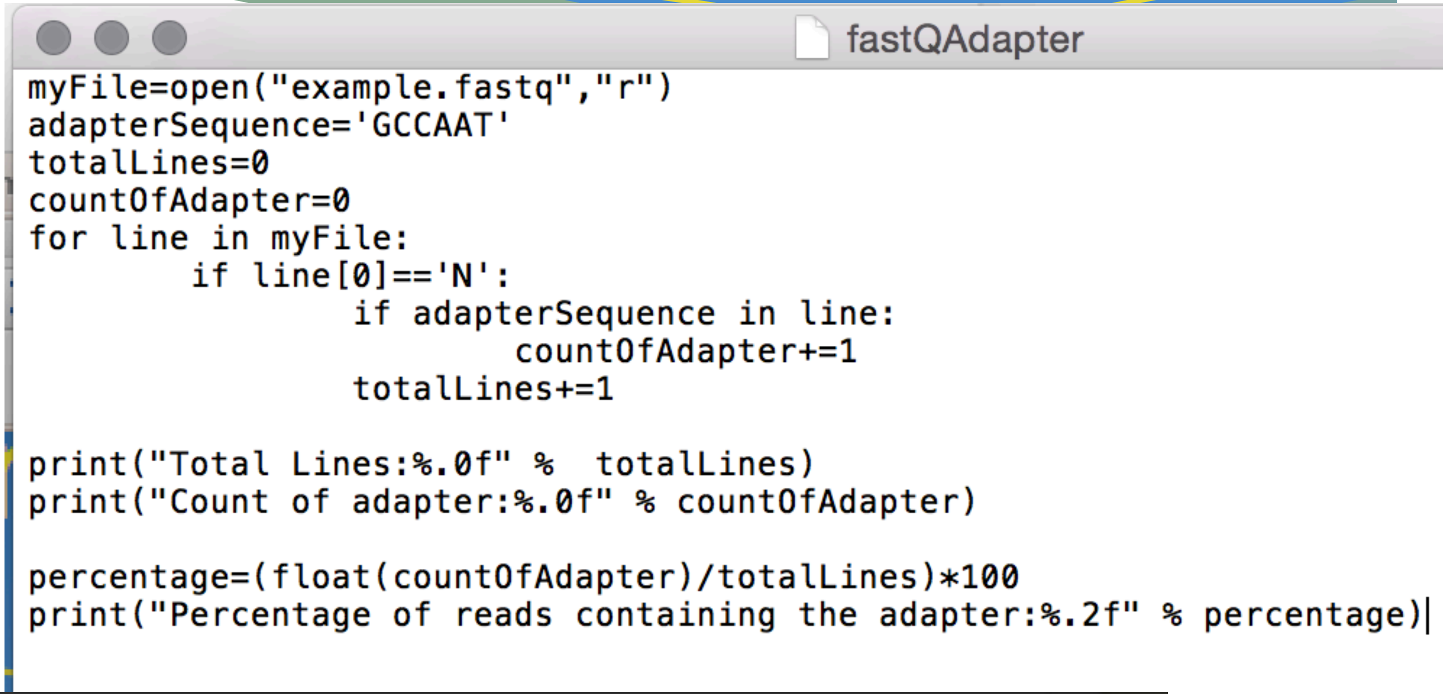
Code to find which reads contain an adapter sequence



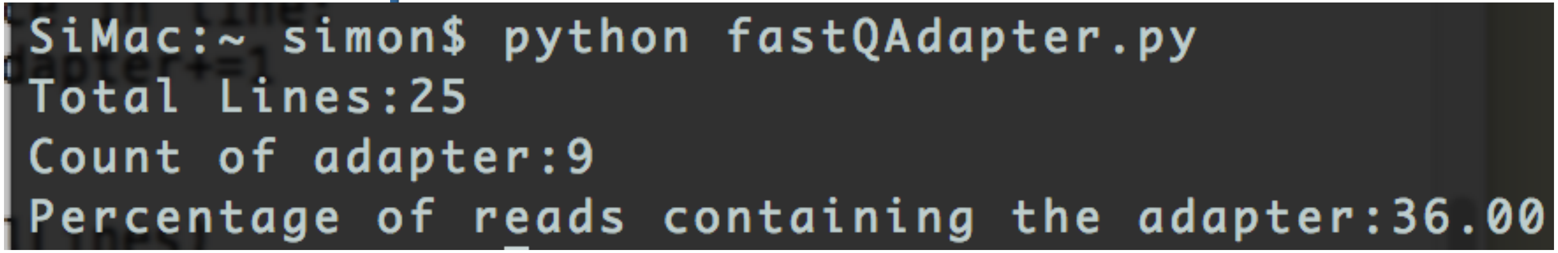
```
myFile=open("example.fastq","r")
adapterSequence='GCCAAT'
totalLines=0
countOfAdapter=0
for line in myFile:
        if line[0]=='N':
                if adapterSequence in line:
                        countOfAdapter+=1
                totalLines+=1

print("Total Lines:%.0f" %  totalLines)
print("Count of adapter:%.0f" % countOfAdapter)

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

fastQ Example

Let’s test it!

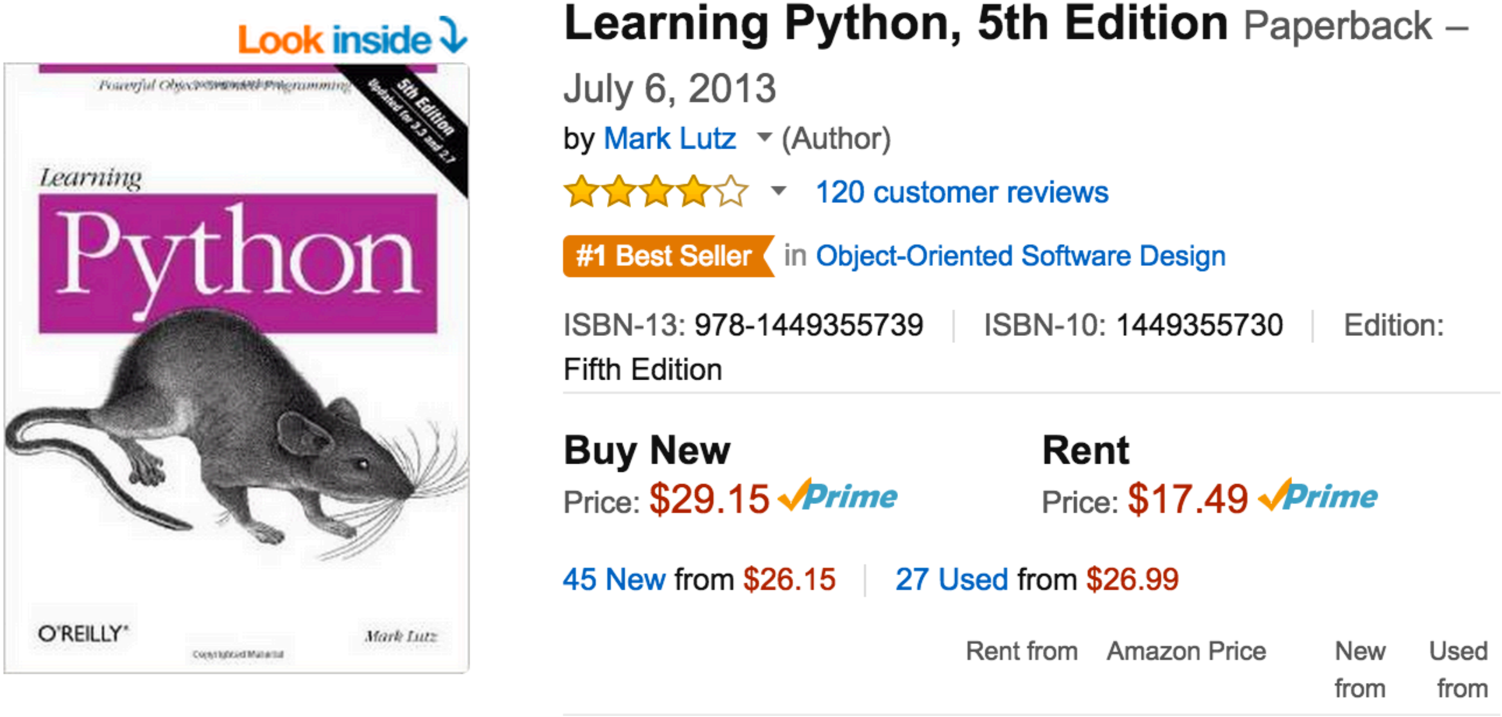


```
myFile=open("example.fastq","r")
adapterSequence='GCCAAT'
totalLines=0
countOfAdapter=0
for line in myFile:
        if line[0]=='N':
                if adapterSequence in line:
                        countOfAdapter+=1
                totalLines+=1

print("Total Lines:%.0f" %  totalLines)
print("Count of adapter:%.0f" % countOfAdapter)

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



```
SiMac:~ simon$ python fastQAdapter.py
Total Lines:25
Count of adapter:9
Percentage of reads containing the adapter:36.00
```

Continued learning



Learning Python, 5th Edition Paperback – July 6, 2013

by Mark Lutz (Author)

★★★★☆ 120 customer reviews

#1 Best Seller in Object-Oriented Software Design

ISBN-13: 978-1449355739 | ISBN-10: 1449355730 | Edition: Fifth Edition

Buy New
Price: $29.15 ✓Prime

Rent
Price: $17.49 ✓Prime

45 New from $26.15 | 27 Used from $26.99

Rent from Amazon Price New from Used from

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/PythonOct2016