An Introduction to Python

Day 3
Simon Mitchell
Simon.Mitchell@ucla.edu
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.
In a text editor:

```
#Functions to analyse values in a list
#print the numbers in the list
def printNNums(numbers):
    for num in numbers:
        print num
```

Comment your code well so you remember what it does when you look at it again.
* A function to sum values:

```python
#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
```
* A function to average numbers:

```python
#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
```
#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

```python
#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")
```
* Test it

```python
>>> 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

```python
>>> 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”:

```python
>>> 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:

```python
>>> 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

```python
>>> 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

To create a list this way:

newList = [expression for value in oldList]
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**.
listName[start:end:stride]

From 1\textsuperscript{st} value to 6\textsuperscript{th}, choosing every 3\textsuperscript{rd} value.

From 2\textsuperscript{nd} value to 9\textsuperscript{th} value, choosing every 4\textsuperscript{th}

Entire list, every other value
Entire list, every value, in reverse

2\textsuperscript{nd} value to 1\textsuperscript{st}, don’t skip any

9\textsuperscript{th} value to end of list, in reverse
From beginning of list to 4\textsuperscript{th} value, in reverse
Lambda functions

An alternative way to define a function.

```python
>>> 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)

```python
>>> 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!)

```
>>> 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.

```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
```
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.

```python
>>> 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.
Always close() Files

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

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

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

```python
>>> 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
GATTTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTGTTCAACTCACAATTTT
+
!''*((((****++))))(%%% .1***+-'''))))**55CCF>>>>>>CCCCCCCC65
```
fastQ file

www.signalingsystems.ucla.edu/users/Simon/example.fastq
Code to find which reads contain an adapter sequence

```python
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!

```python
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


Buy New
Price: $29.15 Prime

Rent
Price: $17.49 Prime

45 New from $26.15 | 27 Used from $26.99

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

surveymonkey.com/r/PythonOct2016