An Introduction to Python

Day 3

Renaud Dessalles
dessalles@ucla.edu
Writing Modules
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:

```python
# 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.
A function to sum values:

```python
# 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
```
# 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 = sumOfNums / len(numbers)
    return average
```

Reminder: in Python2, should put a `float` somewhere
myStats.py

def averageNums(numbers):
    sumOfNums = sumNums(numbers)
    average = sumOfNums / len(numbers)
    return average

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 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")
* 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
```
Problem when debugging a module:

Solution 1: Exit python, re-open python and import again

Solution 2: reload the module

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

* Python 2
  * `reload` is directly available

* Python 3
  * Need to import `reload` via `importlib`

```python
>>> 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

```
>>> 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 **Python 2**, they return simple lists
Iterating over dictionaries

Let’s iterate over the keys:

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

```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:

```python
call = [expression for value in oldList]
```
Reverse complement function we wrote previously 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`. 

```python
>>> 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)
'ATCGATT'T'
```
Slicing Up a List (with step)

listName[start:end:step]

- 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}
- List with values with only pairwise index
- Entire list, every value, in reverse
- 2\textsuperscript{nd} value down to the beginning
- 9\textsuperscript{th} value down to the beginning
- From beginning of list to 4\textsuperscript{th} value, in reverse
Lambda functions

An alternative way to define a function.

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

More compact, but also useful in conjunction with other functions.
The function `filter(function, list)`

```python
>>> 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

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

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

```python
>>> 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
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.
* Write in a file:

```python
>>> 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.
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.readline())
125
125
216
343
512
729
1000

>>> myFile.close()
```
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
GATTTGGGGTTCAAAGCAGTATCGATCAAAATAGTAAATCCATTGTTCACACTCACAGTTT
+
!''*((****++)&amp;&amp;++)(&amp;&amp;&amp;).1***--++')**55CCF&gt;&gt;&gt;&gt;&gt;&gt;&gt;CCCCCCC65
```
fastQ file

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

In home directory

.fastq extension
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

myFile.close()

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

decimal = 4
percentage = (countOfAdapter / totalLines) * 100
print("Percentage of reads containing the adapter: %.2f"% percentage)
```

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

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:~ qcbbcclaboratory$ 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**

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