Python: Lists and Files

Only two more classes to go before warp speed data processing

Reminder: Python Data Types

Object type	Category	Mutable?
Numbers	Numeric	No
Strings	Sequence	No
Files	Extension	N/A
Lists	Sequence	Yes
Dictionaries	Mapping	Yes
Tuples	Sequence	No

Python Lists

- Lists are one-dimensional arrays
- **Every list begins and ends with square brackets and the elements are separated by commas.**
- They can store data of different types in a sequence (strings, floats, integers, logical T/F). **They can even store lists.**
- Lists are **mutable**, meaning you can edit it directly (strings are not, you need to save a new one)
- Unlike a string, this sequence is in **convenient chunks**

```
>>> mygenes = ['BRCA1','FGF21','OR45F']
>>> data1 = ['Tigers',15,2423.333,True,'3 cubs']
>>> data1
['Tigers', 15, 2423.333, True, '3 cubs']
```

Combining numbers, strings, and lists

- The + operator is cool in Python because it can combine elements across data types
- In numbers, it adds the two values, but in lists and strings, it joins the elements together
- The – operator, unfortunately, only works for numbers

```
>>> 4 + 6
10
>>> '4' + '6'
'46'
>>> [4] + [6]
[4, 6]
```

Lists can store other lists!

- List of lists allow you to build a **multidimensional array of mixed data types**. This is big deal because all biological data spreadsheets pretty much fall under this category
- Note that some elements of the sublist are floats, some are strings. Some can even be placeholders, like the 0.0 I used.

```
[['A1BG', 'ILMN_2055271', 'cg03123289', -1.1915, -0.2185, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7025, 1.9365, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.3119, -0.2319, 1.0, 0.0, 0.0, 0.0, -0.1537,
0.1468, -1.0, 0.1352, -0.6589, 1.0, 0.0, 0.0, 0.0, -3.2305, 1.0189, -1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0974, 0.3849, 0.0, ]
['A1BG', 'ILMN_2055271', 'cg03630821', -1.1915, 0.0475, -1.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.7025, 0.5401, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.3119, 0.2522, 0.0, 0.0, 0.0, 0.0, -0.1537,
0.2327, -1.0, 0.1352, 0.4055, 0.0, 0.0, 0.0, 0.0, -3.2305, 0.9767, -1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0974, -0.059, 1.0, ]]
```

Lists can store other lists!

- **Every list within a list also begins and ends with square brackets and the elements are also separated by commas.**
- Lists-in-lists are created the same way lists are, but with a list as an element of a larger list. Usually the larger list holds together smaller lists of the same size
- Each element of a sublist represents a different column of information
- When we learn later today about pulling data off of files, lists of lists are an invaluable way to organize the information

```
>>> mydata1 = [['AGCATGGTTCA', 3, 2, 3, 3],['GGGTTATACAT', 3, 1, 3, 4]]
>>> mydata1
[['AGCATGGTTCA', 3, 2, 3, 3], ['GGGTTATACAT', 3, 1, 3, 4]]
```

Slicing Strings in Python

- Python has a 0-based numbering system, so counting goes: 0, 1, 2, 3, 4, etc.
- Indexing **[beg:end:step]**
  - Beginning: count to the letter you want starting by 0
  - End: (n-1) by the 0-based system
  - Step: 1 is every letter, 2 is every second letter, 3 is every third, etc.
- If **beg** or **end** is negative, it will count backwards from the end of the string
- If **beg** or **end** is left out, it defaults to 0 and the end, respectively
- You can **reverse** a string by adding [::-1] to the slice or string

```
>>> dna = 'CGGTTAATAGGGACTCTC'
>>> dna[4:10:2]
'TAA'
>>> dna[-2:-1]
'CT'
```

```
>>> dna[2:]
'GTTAATAGGGACTCTC'
>>> dna[2:][::-1]
'CTCTCAGGGATAATTG'
```

Indexing lists is similar to strings

- For a list of lists, the first index tells you which sublist, and the second index tells you which element of said sublist
- List slicing can do all of the things that string slicing can do (but if you want to slice a list of lists, only the last index can be a slice)
- **You can replace list elements with indexing**

```
>>> data1 = ['Tigers',15,2423.333,True,'3 cubs']
>>> data1[1:4]
[15, 2423.333, True]
>>> mydata1 = [['AGCATGGTTCA', 3, 2, 3, 3],['GGGTTATACAT', 3, 1, 3, 4]]
>>> mydata1[0][2]
2
>>> mydata1[0][2:5:2]
[2, 3]
>>> mydata1[1][0] = 'helloworld'
>>> mydata1
[['AGCATGGTTCA', 3, 3, 2, 3], ['helloworld', 3, 1, 3, 4]]
```

String Methods and Functions

- This is a method: **string.method()**
  - dna.count('A') # counts the number of adenines in your sequence
  - dna.find('GGGA') # finds exact motif and returns index (or -1)
  - dna.replace('GG', 'CC') # replaces all occurrences of 'GG' with 'CC'
  - And many, many more! (The real power of strings comes with **regular expressions** that deal with inexact matches, which will be covered later)
- This is a function: **function(string)**
  - len(dna) # gives you the length of the sequence
  - str(anything) # turns anything (like numbers) into a string
  - And many more! (I rarely use functions on strings, TBH)

List methods and functions

- As you would expect, lists also use functions and methods the exact same way as strings, but they do different things
- In fact, lists and strings, because they are both sequences, share a lot of similar functions and methods. Give some of them a try!
- One really neat function for lists is range()

```
>>> mydata1
[['AGCATGGTTCA', 3, 2, 3, 3], ['GGGTTATACAT', 3, 1, 3, 4]]
>>> mydata1[1].index(1)
2
>>> mydata1[1].count(3)
2
>>> mydata1[1][0].count('T')
4
>>> range(9)
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

List methods and functions

- Some important (new) methods for lists:
  - **List.pop(i)** will “pop” out a list item, and you can specify the index. This is permanent.
  - **List.insert(pos, ele)** will insert the element at the given position. This is permanent.
  - **List.join('delim')** will join the elements of the list together if they are all strings by a delimiter (its opposite is string.split())
  - **List.append(ele)** will append an element to the end of the specified list. This is permanent.

List methods and functions

```
>>> #we make a new string with some commas separating sequences
>>> mydata2 = 'AGCATGGTTCA,TTTGACGACG,TTCGCAGCTCA'
>>> mydata3 = mydata2.split(',') # split will create a list where every element is what came before or after the delimiter (in this case, a comma)
>>> mydata3
['AGCATGGTTCA', 'TTTGACGACG', 'TTCGCAGCTCA']
>> #how do I reverse it but change the delimiter? Use join()
>>> ' '.join(mydata3)
'AGCATGGTTCA TTTGACGACG TTCGCAGCTCA'
>>> mydata3
['AGCATGGTTCA', 'TTTGACGACG', 'TTCGCAGCTCA']
>>> mydata2
'AGCATGGTTCA,TTTGACGACG,TTCGCAGCTCA'
>>> mydata3.append('helloworld')
>>> mydata3
['AGCATGGTTCA', 'TTTGACGACG', 'TTCGCAGCTCA', 'helloworld']
>>> #append() changes it in-place, so the change to the list is permanent
```

List methods and functions

```
>>> mydata1
[['AGCATGGTTCA', 3, 2, 3, 3], ['GGGTTATACAT', 3, 1, 3, 4]]
>>> mydatacp = mydata1[:] #make a copy of your list array
>>> mydata1.pop()
['GGGTTATACAT', 3, 1, 3, 4] #pop removes the first item of the list by default
>>> mydata1
[['AGCATGGTTCA', 3, 2, 3, 3]] #it changes the original list permanently
>>> mydata1 = mydatacp #refresh with your copy
>>> mydata1[0].pop(3) #use indexing to make it more specific
3
>>> mydata1
[['AGCATGGTTCA', 3, 2, 3], ['GGGTTATACAT', 3, 1, 3, 4]]
>>> mydata1[0].insert(1,3) #insert does the opposite of pop
>>> mydata1 #a 3 should be inserted at position 1 of sublist 0
[['AGCATGGTTCA', 3, 3, 2, 3], ['GGGTTATACAT', 3, 1, 3, 4]]
>>> #of course now the data is wrong information :P
```

Problem Solving

- Go to **sacklerpython.wordpress.com** and download the file '**tumorvsnormal.csv**' into your **myPython** folder (your active folder in Python should still be this folder)
- Now in your Python interpreter, type the following to open the file:
  - After typing the colon, hit Enter and then Tab before typing 'dna' to indent the second line (it's a long story, I'll explain later)

```
>>> with open('tumorvsnormal.csv', 'r') as file:
        data = file.read().splitlines()
>>> data
['chr1\t14907\tA\tG\t689\t574', 'chr1\t14930\tA\tG\t790\t668',
'chr1\t14933\tG\tA\t1009\t443']
```

Problem Solving

Remember that everything we trudge through today is easily automated, but you need to understand the process

- Problem 1: Create an empty list and name it **tdata**. An empty list is two brackets with nothing in between (Python ignores spaces)
- Problem 2: Assign the first string of **data** to a variable called **temp**. Using one specific method mentioned earlier, turn the string with the '\t' delimiters into a list. '\t' is the programming shorthand for a Tab. Remember to assign what you do to a variable.
- Problem 3: Append that to your empty list. Do not assign this one to a variable because appending changes things permanently.
- Problem 4: Do this for the other two list items.

Problem Solving

```
>>> with open('tumorvsnormal.csv', 'r') as file:
...     data = file.read().splitlines()
...
>>> data
['chr1\t14907\tA\tG\t689\t574', 'chr1\t14930\tA\tG\t790\t668',
'chr1\t14933\tG\tA\t1009\t443']
>>> tdata = []
>>> temp = data[0]
>>> temp = temp.split('\t')
>>> tdata.append(temp)
>>> temp = data[1]
>>> temp = temp.split('\t')
>>> tdata.append(temp)
>>> temp = data[2]
>>> temp = temp.split('\t')
>>> tdata.append(temp)
>>> tdata
[['chr1', '14907', 'A', 'G', '689', '574'], ['chr1', '14930', 'A', 'G', '790', '668'], ['chr1',
'14933', 'G', 'A', '1009', '443']]
>>>
```