cs4_Section5

March 5, 2019

```
In [1]: from functools import reduce
```

Filter, map, and reduce are a set of higher order functions that we can use to speed up list operations without having to write repetetive commands like looping through a list. See below for examples. Remember to import reduce from functools before using it.

```
In [12]: def isVowel(x):
             return x in 'AEIOUaeiou'

         def double(x):
             return x*2

         def add(x,y):
             return x + y
```

```
In [6]: #Map examples
        answer=map(double, [1,2,3])
        #If we print answer, we get a map object
        print(answer)
        #Need to wrap - 'cast'- answer to list
        print(list(answer))
        #We can use map on functions that take multiple parameters- each list becomes the next p
        answer=list(map(add, [1,2,3],[10,10,10]))
        print(answer)
```

```
<map object at 0x11210fc88>
[2, 4, 6]
[11, 12, 13]
```

```
In [11]: #Filter example
         answer = filter(isVowel, 'steam')
         #Same as above- we get a filter object, if we don't wrap
         print(answer)
         print(list(answer))
         #Remember- these functions always return a list for ANY iterable type: strings, lists,
```

```
<filter object at 0x11218d550>
['e', 'a']
```

```
In [14]: #Reduce examples
         answer=reduce(add, [1,2,3])
         #This time, printing works as expected because we return a single value
         print(answer)
```

```
6
```

```
In [15]: #Lets practice
         def double_vowels(word):
             '''
             Doubles each vowel in the word.
             'hello' -> 'eeoo'
             '''
             # First, we filter vowels, double each vowel, and then add them together to form a
             return reduce(add, map(double, filter(isVowel, word)))
```

```
In [16]: double_vowels('hello')
```

```
Out[16]: 'eeoo'
```

On your homework, we ask you to write a cipher. To do so, you must convert characters into integers and vice versa. This can be done using the ord(character) and chr(integer) commands. These get converted based on ASCII, which is found here. The cipher that we want you to implement is a rotation cipher. This means that every letter moves forward by a certain integer, n. For instance, if n=4, then cipher('a') should produce 'e'. **Case must be preserved**. So how do we do this? We can add n to the ASCII value for each character, chr(ord(character)+n)=answer. However, what about at the end of the alphabet? For instance, with n=4, cipher('z') should produce 'd'. How do we wrap around? *Use the modulus operator*. In general, if a range of acceptable values are 0 <= n < c, for some integer c, we can use the modulus to keep the acceptable values in that amount. Thus, for this problem, the value of c is 26, so we want to include modulus 25. Work out an example if this is confusing to you. There's one more step to this question: you need to reindex this for both *upper and lower case*.

```
In [ ]:
```

```
In [ ]:
```