Functional Ideas in Python

"Gematria" is a system for assigning a number to a word by summing the numeric values of each of the letters as defined by the Mispar godol (https://en.wikipedia.org/wiki/Gematria). For English characters, we can use the ASCII table (https://en.wikipedia.org/wiki/ASCII). It is not necessary, however, to encode this table in our program as Python provides the `ord` function to convert a character to its "ordinal" (order in the ASCII table) value as well as the `chr` function to convert a number to its "character."

```
print('"{}" = "{}"'.format('A', ord('A')))
print('"{}" = "{}"'.format('a', ord('a')))
print('"{}" = "{}"'.format(88, chr(88)))
print('"{}" = "{}"'.format(112, chr(112)))
```

```
"A" = "65"
"a" = "97"
"88" = "X"
"112" = "p"
```

To implement an ASCII version of gematria in Python, we need to turn each letter into a number and add them all together. So, to start, note that Python can use a `for` loop to cycle through all the members of a list (in order):

```
for n in range(5):
    print(n)
```

```
0
1
2
3
4
```

```
for char in ['p', 'y', 't', 'h', 'o', 'n']:
    print(char)
```

```
p
y
t
h
o
n
```

A "word" is simply a list of characters, so we can iterate over it just like a list of numbers:

```
for char in "python":
    print(char)
```

```
p
```

```
y
t
h
o
n
```

Let's print the ordinal (ASCII) value instead:

```
for char in "python":
    print('"{}" = "{}"'.format(char, ord(char)))
```

```
"p" = "112"
"y" = "121"
"t" = "116"
"h" = "104"
"o" = "111"
"n" = "110"
```

Now let's create a variable to hold the running sum of the values:

```
word = "python"
total = 0
for char in word:
    total += ord(char)

print('"{}" = "{}"'.format(word, total))
```

```
"python" = "674"
```

Another way could be to create another list to hold the values and then use the `sum` function:

```
word = "python"
all = []
for char in word:
    all.append(ord(char))

print(all)
print('"{}" = "{}"'.format(word, sum(all)))
```

```
[112, 121, 116, 104, 111, 110]
"python" = "674"
```

Map

We can use a `map` function to transform all the characters via the `ord` function. This is interesting because `map` is a function that takes another function as its first argument. The second is a list of items to feed into the function. The result is the transformed list. For instance, we can use the `str.upper` function to turn

each letter (e.g., "p") into the upper-case version ("P"). NB: it's necessary to force the results into a `list`.

```
list(map(str.upper, "python"))
```

```
['P', 'Y', 'T', 'H', 'O', 'N']
```

```
list(map(ord, "python"))
```

```
[112, 121, 116, 104, 111, 110]
```

Now we can `sum` those numbers:

```
sum(map(ord, "python"))
```

```
674
```

Now let's think about how we could apply this to all the words in a file. As above, we can use a `for` loop to iterate over all the lines in a file:

```
for line in open('gettysburg.txt'):
    print(line)
```

```
Four score and seven years ago our fathers brought forth on this

continent, a new nation, conceived in Liberty, and dedicated to the

proposition that all men are created equal.
```

The original is single-spaced, so why is this printing double-spaced? The `for` loop reads each "line" which is a string of text up to and including a newline. The `print` by default adds a newline, so we either need to `print(line, end='')` to indicate we don't want anything at the end:

```
for line in open('gettysburg.txt'):
    print(line, end='')
```

```
Four score and seven years ago our fathers brought forth on this
continent, a new nation, conceived in Liberty, and dedicated to the
proposition that all men are created equal.
```

Or we need to use the `rstrip` function to "strip" whitespace off the "r"ight side of the line:

```
for line in open('gettysburg.txt'):
    print(line.rstrip())
```

```
Four score and seven years ago our fathers brought forth on this
continent, a new nation, conceived in Liberty, and dedicated to the
proposition that all men are created equal.
```

We can use the `split` function to get all the words for each line and a `for` loop to iterate over those:

```
for line in open('gettysburg.txt'):
    for word in line.rstrip().split():
        print(word)
```

```
Four
score
and
seven
years
ago
our
fathers
brought
forth
on
this
continent,
a
new
nation,
conceived
in
Liberty,
and
dedicated
to
the
proposition
that
all
men
are
created
equal.
```

We want to get rid of anything that is not character like the punctuation. There is a function in the `str` library called `isalpha` that returns `True` or `False`:

```
for char in "a8,X.b!G":
    print('"{}" = "{}"'.format(char, str.isalpha(char)))
```

```
"a" = "True"
"8" = "False"
"," = "False"
"X" = "True"
"." = "False"
"b" = "True"
"!" = "False"
```

```
"G" = "True"
```

Each `char` in the loop is itself a string, so we can call the method directly on the variable:

```
for char in "a8,X.b!G":
    print('"{}" = "{}"'.format(char, char.isalpha()))
```

```
"a" = "True"
"8" = "False"
"," = "False"
"X" = "True"
"." = "False"
"b" = "True"
"!" = "False"
"G" = "True"
```

Filter

Similar to what we saw above with the `map` function, we can use `filter` to find all the characters in a string which are `True` for `isalpha`. `filter` is another "higher-order function" that takes another function for its first argument (called the "predicate") and a list as the second argument. Whereas `map` returns *all* the elements of the list transformed by the function, `filter` returns *only those for which the predicate is true*.

```
list(filter(str.isalpha, "a8,X.b!G"))
```

```
['a', 'X', 'b', 'G']
```

The first argument for `map` and `filter` is called the "lambda," and sometimes you will see it written out explicitly like so:

```
list(filter(lambda char: char.isalpha(), "a8,X.b!G"))
```

```
['a', 'X', 'b', 'G']
```

Here is a way to find only even numbers:

```
list(filter(lambda x: x % 2 == 0, range(10)))
```

```
[0, 2, 4, 6, 8]
```

Let's turn that list of characters back into a word with the `join` function:

```
''.join(filter(str.isalpha, "a8,X.b!G"))
```

```
'aXbG'
```

So, going back to our Gettysburg example, here is a list of all the words without punctuation:

```
for line in open('gettysburg.txt'):
    for word in line.rstrip().split():
        print(''.join(filter(str.isalpha, word)))
```

```
Four
score
and
seven
years
ago
our
fathers
brought
forth
on
this
continent
a
new
nation
conceived
in
Liberty
and
dedicated
to
the
proposition
that
all
men
are
created
equal
```

Now, rather let's print the `sum` of the `chr` values for each cleaned up word:

```
for line in open('gettysburg.txt'):
    for word in line.rstrip().split():
        clean = ''.join(filter(str.isalpha, word))
        print('"{}" = "{}"'.format(clean, sum(map(ord, clean))))
```

```
"Four" = "412"
"score" = "540"
"and" = "307"
"seven" = "545"
"years" = "548"
"ago" = "311"
```

```
"our" = "342"
"fathers" = "749"
"brought" = "763"
"forth" = "547"
"on" = "221"
"this" = "440"
"continent" = "978"
"a" = "97"
"new" = "330"
"nation" = "649"
"conceived" = "944"
"in" = "215"
"Liberty" = "731"
"and" = "307"
"dedicated" = "919"
"to" = "227"
"the" = "321"
"proposition" = "1222"
"that" = "433"
"all" = "313"
"men" = "320"
"are" = "312"
"created" = "728"
"equal" = "536"
```

Notice that we are calling `rstrip` for every line, so we could easily move that into a `map`, and the "cleaning" code can likewise be moved into a `map`:

```
for line in map(str.rstrip, open('gettysburg.txt')):
    for word in map(lambda w: ''.join(filter(str.isalpha, w)), line.split()):
        print('"{}" = "{}"'.format(word, sum(map(ord, word))))
```

```
"Four" = "412"
"score" = "540"
"and" = "307"
"seven" = "545"
"years" = "548"
"ago" = "311"
"our" = "342"
"fathers" = "749"
"brought" = "763"
"forth" = "547"
"on" = "221"
"this" = "440"
"continent" = "978"
"a" = "97"
"new" = "330"
```

```
"nation" = "649"
"conceived" = "944"
"in" = "215"
"Liberty" = "731"
"and" = "307"
"dedicated" = "919"
"to" = "227"
"the" = "321"
"proposition" = "1222"
"that" = "433"
"all" = "313"
"men" = "320"
"are" = "312"
"created" = "728"
"equal" = "536"
```

At this point, we have arguably sacrificed readability for the sake of using `map` and `filter` – another instance of "just because you can doesn't mean you should!"

We can improve readability, however, by creating our own functions with informative names:

```
def onlychars(word):
    return ''.join(filter(str.isalpha, word))

def word2num(word):
    return sum(map(ord, word))

for line in map(str.rstrip, open('gettysburg.txt')):
    for word in map(onlychars, line.split()):
        print('"{}" = "{}"'.format(word, word2num(word)))
```

```
"Four" = "412"
"score" = "540"
"and" = "307"
"seven" = "545"
"years" = "548"
"ago" = "311"
"our" = "342"
"fathers" = "749"
"brought" = "763"
"forth" = "547"
"on" = "221"
"this" = "440"
"continent" = "978"
"a" = "97"
"new" = "330"
"nation" = "649"
```

```
"conceived" = "944"
"in" = "215"
"Liberty" = "731"
"and" = "307"
"dedicated" = "919"
"to" = "227"
"the" = "321"
"proposition" = "1222"
"that" = "433"
"all" = "313"
"men" = "320"
"are" = "312"
"created" = "728"
"equal" = "536"
```

Here's a streamlined version that combines `open`, `read`, and `split` to read the entire file into a list of words which are `map`d into `word2num`.

NB: This version assumes you have enough memory to read an *entire file* and split it. The versions above which read and process each line consume only as much memory as any one line needs!

```
def onlychars(word):
    return ''.join(filter(str.isalpha, word))

def word2num(word):
    return str(sum(map(ord, word)))

print(' '.join(map(word2num,
                   map(onlychars,
                       open('gettysburg.txt').read().split()))))
```

```
412 540 307 545 548 311 342 749 763 547 221 440 978 97 330 649 944 215 731 307 919 227 321 1
```

To mimic the above output:

```
def onlychars(word):
    return ''.join(filter(str.isalpha, word))

def word2num(word):
    return str(sum(map(ord, onlychars(word))))

print('\n'.join(map(lambda word: '"{}" = "{}"'.format(word, word2num(word)),
                    map(onlychars,
                        open('gettysburg.txt').read().split()))))
```

```
"Four" = "412"
"score" = "540"
"and" = "307"
```

```
"seven" = "545"
"years" = "548"
"ago" = "311"
"our" = "342"
"fathers" = "749"
"brought" = "763"
"forth" = "547"
"on" = "221"
"this" = "440"
"continent" = "978"
"a" = "97"
"new" = "330"
"nation" = "649"
"conceived" = "944"
"in" = "215"
"Liberty" = "731"
"and" = "307"
"dedicated" = "919"
"to" = "227"
"the" = "321"
"proposition" = "1222"
"that" = "433"
"all" = "313"
"men" = "320"
"are" = "312"
"created" = "728"
"equal" = "536"
```

With this, I hope you're now understand what is meant by a "higher-order function" (functions that take other functions as arguments) and how they can streamline your code.

Exercise

Read your local dictionary (e.g., "/usr/share/dict/words") and find how many words share the same numeric representation. Which ones have the value "666"?

```
from collections import defaultdict

def onlychars(word):
    return ''.join(filter(str.isalpha, word))

file = '/usr/share/dict/words'
num2word = defaultdict(list)
for line in map(str.rstrip, open(file)):
    for word in map(onlychars, line.split()):
```

```
        num = sum(map(ord, word))
        num2word[num].append(word)

satan = '666'
if satan in num2word:
    print('Satan =', num2word[satan])
else:
    print('No Satan!')

count_per_n = []
for n, wordlist in num2word.items():
    count_per_n.append((len(wordlist), n))

top10 = list(reversed(sorted(count_per_n)))[:10]
for num_of_words, n in top10:
    print('"{}" = {}'.format(n, ', '.join(num2word[n])))
```

```
No Satan!
"973" = Actaeaceae, activator, actorship, aculeolus, admixtion, admonitor, advertent, aeropa
"969" = abrotanum, acclivous, acidulous, adnexitis, adoringly, aerobious, afterpart, afterso
"965" = abhorrent, acoumeter, acronymic, admissive, advisedly, aegrotant, aerophore, afterta
"855" = abuseful, acanthus, acronych, aerodyne, akamatsu, akinesis, algidity, alkylate, altr
"861" = Absyrtus, acaulous, adjuvant, airwards, akroasis, alarmist, alastrim, albronze, aleu
"856" = abrastol, accismus, acervose, acromion, acrostic, actinism, adorally, aerolith, agno
"971" = aburabozu, acropathy, acuteness, aftercost, afterglow, afterroll, albatross, aleuron
"974" = ablastous, absolvent, abysmally, acoustics, acrospore, adjunctly, adventure, affixtu
"972" = accessory, acropolis, acutiator, adderwort, adrostral, aerocurve, affluxion, afterno
"1078" = absentness, acrogenous, actinozoan, acuclosure, adipopexis, agonizedly, alimentary
```