While loops

I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
I will not chew gum in CS 5 Green!
Naming homework files and functions

In this problem, you'll write several helper functions for manipulating DNA sequences. These functions will be useful in subsequent problems where you'll analyze DNA data in various ways. All of your code for this problem should be in a file called `dna.py`.

**Base complements**

Write a function `compBase(N)` that takes a string that is a single DNA base "A", "G", "T", or "C" as input and returns the base that is complementary to it. Here are some examples:

```python
>>> compBase("A")
'T'
>>> compBase("G")
'C'
```

- Case sensitive
- One solution: cut and paste function name when you define it
Editor and shell revisited

(There are many different text editors, you can use whichever you like.)
while loops

def mystery(n):
    k = 1
    while k < n:
        k = k * 2
    return k

>>> mystery(1)

>>> mystery(5)

>>> mystery(10)
while loops

def mystery(n):
    start = 0
    while start <= n:
        print(start)
        start = start + 1
while loops

def bart():
    while True:
        print("I will not chew gum in CS5 Green.")
>>> bart(4)
1 I will not chew gum in CS5 Green.
2 I will not chew gum in CS5 Green.
3 I will not chew gum in CS5 Green.
4 I will not chew gum in CS5 Green.

def bart(times):
    for line in range(1,times+1):
        print(line,"I will not chew gum in CS5 Green.")

def bart(times):
    line = 1
    while line <= times:
        print(line,"I will not chew gum in CS5 Green.")
        line = line + 1
while loops

def collatz(n):
    '''Applies the collatz function to the number'''
    if n % 2 == 0:
        return n/2
    else:
        return 3*n + 1

def howManyTimes(n):
    ''' Determines the number of times that collatz must
    be applied to n before we get 1 '''
    counter = 0

>>> 5 == 5
True

>>> 5 != 6
True

>>> howManyTimes(5)
5

5  ->  16  ->  8  ->  4  ->  2  ->  1
while loops

def collatz(n):
    '''Applies the collatz function to the number'''
    if n % 2 == 0:
        return n/2
    else:
        return 3*n + 1

def howManyTimes(n):
    '''Determines the number of times that collatz must be applied to n before we get 1'''
    counter = 0
    while n != 1:
        n = collatz(n)
        counter += 1
    return counter

>>> howManyTimes(5)

DEMO!
A Prime Example of Looping!

```python
>>> thisManyPrimes(5)
[2, 3, 5, 7, 11]
```

def prime(k):
    '''Returns True if k is prime and False otherwise'''
    for d in range(2, k):
        if k % d == 0:
            return False
    return True

def thisManyPrimes(n):
    '''Returns a list of the first n primes. '''
A Prime Example of Looping!

```python
>>> thisManyPrimes(5)
[2, 3, 5, 7, 11]

def prime(k):
    ''' Returns True if k is prime and False otherwise '''
    for d in range(2, k):
        if k % d == 0:
            return False
    return True

def thisManyPrimes(n):
    ''' Returns a list of the first n primes. '''
    k=2
    primeList=[]
    while len(primeList) < n:
        if prime(k):
            primeList.append(k)
        k += 1
    return primeList
```

Worksheet
Homework: gene finding in a region of DNA unique to salmonella

Salmonella pathogenicity island 1
The central dogma in a nutshell

DNA  →  RNA  →  Protein

Promoter  →  S  →  TAG

ATG
Finding open reading frames (ORFs): check all 3 reading frames
Genes can occur on either strand

ATG = start codon
TGA, TAG, TAA = Stop codons

Gene 1: coding strand is on top

5' – ATGCCGTGCTTTGTAAGCTAGGCTTAGATCGTCTATGGG – 3'

3' – TTACGGCACGAACATCTGCATCCGAATCTAGCA GTACCC – 5'

Gene 2: coding strand is on bottom
Noncoding sequence between Salmonella genes

Its orfully strange to find this here!

GCTATCTCACTCGTCAGCCCCAAATCCTGCCAGTGCTCACACAAACGCAGCGCGTTTTGAACGTCCGTAA
GGACGGCCCCGTAGGGGTAGCTTCGCAGATCACATCTTACGTACTTTCAACGAATTCTTAA
ATTATTTTGGGTGTTGTAGGCCGGATAAGCAGCTGC
The unfortunate truth

- Not every ATG is a start codon
- Not every ORF is a gene

How can we separate gene ORFs from ORFs due to random chance?
A simple gene finding strategy

A sequence of interest

AATGGGCGGACCACAAGGCGACATAGACGCGAATCGGACCAGACGCCGGCTCACCTGTTCATCTACCTTTCTGCGTTGGCGCTAAAAGTTAACGATCGGGCCCTGCGCCGAAACGAAACGTCAGGAATCGACAAATACCAAGTATCTAAGCTACGGGATAAGCCCCCCCTCGCGAGAGAGGGGAAGGGGTCAATATTTCCCTGGCCGACTGACAA
TGGAGTGTACTTACCGGTATACAGTTTGTACTCTACAGCCATCGCTGCTTTACGACGTATTCGGGGCATTTCAACATGCTGTCTCTCAGGAGTTTTCGCGCGC

ORF: 318 nucs
A simple gene finding strategy

A sequence of interest

ATGGGGCCGACCAAAGGCGACATAGACGCGAATCGGACCAGACGCGGCTCACCTGTTCATCTACCTTTTCTG
CGTTGGGCTAAAAAGTTAACGATCGGGCCCTGCGCCGAAACGAAACGTACGGAATCGACAAATACCAAGTA
TCTAAGCTACGGGATAAGCCCCCCTCGCGAGAGAGGGGAAGGGGTCAATATTTCCCTGGCCGACTGACAA
TGGAGTGTACTTACCCGTATACAGTTTTGTCTACTCTACAGCATCGCTGTCTTACGACGTATTCGGGGCATTT
CAACATGCTGTCTCTCAGGAGTTCCTCGGCACGCTGAAGAACTCCCATCTAAACCCTG

Randomly shuffled versions of this sequence

ORF: 318 nucs

Longest ORF: 153 nucs

Don't forget to look at the reverse complements!
Modules and the import statement

```python
>>> L=['A','C','G','G','T','C','A']

>>> L
['A', 'C', 'G', 'G', 'T', 'C', 'A']

>>> import random

>>> random.shuffle(L)

>>> L
['C', 'T', 'A', 'A', 'C', 'G', 'G']
```
Finding extremz!

dictionary = ['abdomen', 'abdominal', 'abduct', 'abduction', 'aberration', 'abet', 'abhor', 'abhorrence', 'abhorrent', 'abide', 'abiding', 'ability', 'abject', 'ablaze', ...
...
etc.
...
...]

def z(input):
    '''Count z's in a string'''
    counter = 0
    for symbol in input:
        if symbol=='z':
            counter = counter +1
    return(counter)

http://www.flickr.com/photos/thepartycow/247212277/
def extremz(wordList):
    '''Find and return the word with the most z's'''
def extremz(wordList):
    '''Find and return the word with the most z's'''
    bestCount=0
    bestWord=""
    for word in wordList:
        count=z(word)
        if count>bestCount:
            bestCount=count
            bestWord=word
    return bestWord

This is a common “programming motif”.

Homework bonus: look and say

- The look-and-say sequence...

1
11
21
1211
111221
Look-And-Say Sequences (aka “Read-It-And-Weep”)

Number of digits in the $n^{th}$ term of the sequence is given by:

$$C \lambda^n$$

$C = 1.567…$

$\lambda = 1.30357726034296…$

Conway’s Constant
Homework bonus: average length to ATG vs. AAA

'CGAGGCAGGATATCTGGTTTACCAGTGACATACATACATTGATGTGTA...'
Professor P.I. Pette: calculate median genome size in intracellular pathogens vs. free livers

<table>
<thead>
<tr>
<th>speciesList</th>
<th>habitatList</th>
<th>genomeSizeList</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Agrobacterium tumefaciens C58&quot;</td>
<td>[&quot;free living&quot;]</td>
<td>5100</td>
</tr>
<tr>
<td>&quot;Chlamidia psittaci 1H&quot;</td>
<td>[&quot;intracellular&quot;]</td>
<td>1450</td>
</tr>
<tr>
<td>&quot;Salmonella typhimurium&quot;</td>
<td>[&quot;intracellular&quot;, &quot;free living&quot;]</td>
<td>4780</td>
</tr>
<tr>
<td>&quot;Lactococcus lactis C2&quot;</td>
<td>[&quot;free living&quot;]</td>
<td>2500</td>
</tr>
<tr>
<td>&quot;Escherichia coli K12&quot;</td>
<td>[&quot;free living&quot;, &quot;commensal&quot;]</td>
<td>4670</td>
</tr>
</tbody>
</table>

http://www.genomesize.com/prokaryotes/table1/
The data (in Python)

```python
habitatList = [['free living'],
                ['intracellular'],
                ['intracellular', 'free living'],
                ['free living'],
                ['free living', 'commensal'],
                ['intracellular'],
                ['free living', 'commensal'],
                ['free living'],
                ['free living'],
                ['free living'],
                ['free living', 'commensal'],
                ['free living'],
                ['free living'],
                ['intracellular'],
                ['free living', 'commensal'],
                ['free living', 'commensal']]

genomeSizeList = [5100, 1450, 4780, 2500, 4670, 1600, 4140, 4330, 6400, 6700, 4592, 3100, 1120, 7490, 4200]
```
Nick’s code (calculates median genome size in intracellular pathogens vs. free livers)

```python
from bacGenData import *

def gsize(habitatList, genomeSizeList):
a=[]
    for i in range(len(genomeSizeList)):
        if 'intracellular' in habitatList[i]:
            a.append(genomeSizeList[i])
a.sort()
    if len(a) % 2 != 0:
        midInd = int(((len(a) - 1) / 2)
        print('intracellular', 1.0 * a[midInd])
    else:
        y = int(((len(a) / 2) -1)
        b = int(len(a) / 2)
        print('intracellular', (a[y] + a[b]) / 2.0)

z=[]
for i in range(len(genomeSizeList)):
    if 'free living' in habitatList[i]:
        z.append(genomeSizeList[i])
z.sort()
    if len(z) % 2 != 0:
        midInd = int(((len(z) - 1) / 2)
        print('free living', 1.0 * z[midInd])
    else:
        y = int(((len(z) / 2) -1)
        b = int(len(z) / 2)
        print('free living', (z[y] + z[b]) / 2.0)
```
Prof Pette now wants to look at exclusively intracellular or free living species. But Nick is out of town...
def gsize(habitatList, genomeSizeList):
    '''Calculate and print median genome size in intracellular pathogens and also in free livers.'''

    intraSizes = findSizes("intracellular", habitatList, genomeSizeList)
    freeSizes = findSizes("free living", habitatList, genomeSizeList)

    print("intracellular", median(intraSizes))
    print("free living", median(freeSizes))
def median(L):
    '''Calculates the median of list L.''
    L.sort()
    if len(L) % 2 != 0:
        # odd num elements, take middle one
        midInd = int((len(L) - 1) / 2)
        return 1.0 * L[midInd]
    else:
        # even num elements
        midInd1 = int((len(L) / 2) - 1)
        midInd2 = int(len(L) / 2)
        return (L[midInd1] + L[midInd2]) / 2.0

[2,7,18,23,500]          [2,7,18,23]

>>> median([500,18,2,23,7])
18.0
def findSizes(habitat, habitatList, genomeSizeList):
    '''Find all species which live in habitat, and return a list of their genome sizes.'''
    outList = []
    return outList

>>> findSizes("commensal", habitatList, genomeSizeList)
[4670, 4140, 4592, 7490, 4200]
def findSizes(habitat, habitatList, genomeSizeList):
    '''Find all species which live in habitat, and return a list of their genome sizes.''

    outList = []

    for i in range(len(genomeSizeList)):
        if habitat in habitatList[i]:
            outList.append(genomeSizeList[i])

    return outList

>>> findSizes("commensal", habitatList, genomeSizeList)
[4670, 4140, 4592, 7490, 4200]

This does what Nick’s code did...
def findSizes(habitat, habitatList, genomeSizeList):
    '''Find all species which live in habitat, and return a list of their genome sizes.'''

    outList = []

    for i in range(len(genomeSizeList)):
        if habitat == habitatList[i][0]:
            outList.append(genomeSizeList[i])

    return outList

This does the modification Prof Pette asked for...

>>> findSizes("commensal", habitatList, genomeSizeList)
[4670, 4140, 4592, 7490, 4200]