Python workshop 1, practice exercises.

Practicing flowcharts

If we didn't during the workshop, then draw a flowchart for how to make a peanut butter and jelly sandwich. For each of the programs below, draw a flowchart before you write it. Yes, they may be small and not needed here. But you really want to be able to do this once you get to bigger projects. I promise.

Temperature converter

Since I'm an American, and I use a *sensible* unit system, I don't understand your crazy foreign Celsius temperatures. So, I need you to help me. Write me a function that given a temperature in Celsius, returns the corresponding temperature in Fahrenheit. (F= (9/5)*C+32)

Example output:

```
>tempConverter(0)
32.0
>tempConverter(25)
77.0
>tempConverter(-7)
19.4
```

Modify the function so that, if I want to, I can tell it which units I am converting from, so I can convert back if I want to.

```
>tempConverter(0)
32.0
>tempConverter(0, unit=”C”)
32.0
>tempConverter(32, unit=”F”)
0.0
>tempConverter(25, unit=”C”)
17.0
>tempConverter(25, unit=”F”)
-3.88888
```

Note: you can give arguments to functions default options, that way, if you do not provide an argument, it uses the default. Also, you can specify which argument you are passing by using an '= '. For example:

```
def funct(n=2):
     return(n*2)

>funct()
4
```

```
>funct(2)
4
>funct(3)
6
>funct(n=3)
6
```

Translation

Open the file already provided (translation_simple.py), this file contains a string of DNA. You need to add a function to the program that takes that DNA and translates it into amino acids, then prints the resulting sequence. I recommend you use a dictionary for a codon table.

Next, write a function that calculates the GC content of this sequence. Print the result out to the console.

Factorial

Write a function that, given some number N, will calculate N!, and return it.

Example output:

```
>factorial(2)
2
>factorial(10)
3628800
>factorial(5)
120
```

Bonus: write this function without using ANY loops. Don't try to do this first, unless you are an experienced programmer, come back and try this after you've done a few other problems.

Fibonacci numbers

Write a function that, given some number N, will return the Nth Fibonacci number. The Fibonacci numbers follow the sequence: 0,1,1,2,3,5,8...

```
>fibonacci(1)
1
>fibonacci(10)
55
>fibonacci(15)
610
```

Bonus: write this without a loop. Again, don't try to do this first.

Processing strings

I mentioned in class that strings can be treated like lists most of the time, or you can turn a string into a list using the **split()** function.

```
myString = “Apple Bottom Jeans”
myList = myString.split()
print(myList)
```

This is useful when processing many types of text files. Say you have a gene annotation file that is formatted like this:

```
scaffold_1     200  300  IGF2 -
```

That is a chromosome name followed by the location on the chromosome where this gene starts, then the location where it ends, the name of the gene, and the directionality of the gene (is it on the positive or negative strand). So this line says that *Igf2* is on chromosome 1, starting at the 200th base from the end of the chromosome and goes to the 300$^{th.}$

The package for today contains an example file formatted this way (geneAnnotation.txt). Write a program that opens this file, reads in each line, and prints out the gene name and start position for each line in the input file.

Example output:

```
IGF2: 200
GFP: 7000
RPM1: 10000
```

Note, the input file gene names are not unique on each line, don't worry about that. Also, the gene names are not nice things like in this example, you're just getting a truncated version of one of my actual data files with gene names like “PAC:20892461”.

Processing strings 2

Now lets do something fun with this data. You've already figured out how to read in this file, and print out the gene names. Now we want to calculate the number of codons in each gene. For each line, print out the gene name followed by the number of codons:

Example output:

```
IGF2: 34
GFP: 126
RPM1: 867
```

Note, I made those numbers up, don't complain that *Igf2* in reality contains more than 34 amino acids. It's gonna be alright, take a deep breath. Also, the number of bases in each gene may not exactly be a multiple of 3, if so just round up. But before you do that make sure you've counted the number of sites correctly!

Hint: if a gene starts at position 10 and ends at position 12 how many bases are in that gene? (3) how many codons? (1)

Making code more efficient

I mentioned that loops are the second slowest part of any program (what is the first?), so as a good programmer you want to minimize the number of loops in your program. Look the the file provided (theta.py), this file calculates Watterson's theta for three categories of sites. The number of polymorphic sites is stored in the lists 'polymorphic'. The sample size for all three of these categories was the same, N = 10.

This program is broken! The correct output should be:

```
theta 1: 353.485762379
theta 2: 530.228643569
theta 3: 70.6971524758
```

First fix the program so the output is correct. Then define a function that calculates theta, instead of just using a loop. Finally, try and make this code more efficient. Hint: you should end up with only one loop.

Tracking variables through a program

Look at the file (variables.py). Feel free to run the program to see the output. Without adding print statements, write out the value stored in each variable (total, x, and i) for each run through the for loop. Having this skill will help you debug code, and find out where things are going wrong. Once you're done feel free to test things with print statements.