Basic Workflow Example:

- Skills applied:
    - Creating a basic text file with mock data
    - Reading in this data
    - Manipulating input
    - Creating an output file with new data

On Monday we began covering some skills that are very frequently used in data processing and manipulation and so I thought it was a good time to practice a basic workflow with some of these skills (and a few new ones) to show you how we often take in generated data, say from some basic pipeline, and manipulate it to tell us something new.

- To begin, open a new text file on your terminal using a text editor (ex: vim or nano), name the file '`mock_data.txt`'
- Inside of this file, lets create a few rows and columns of mock data:

```
gene_one   1500  1600  1700
gene_two   550   349   475
gene_three 10    27    59
```

    - In this example we have gene names in the first column, followed by 3 columns of expression values from different samples across this gene. Everything in this table is tab separated.
- Open a new file with your chosen text editor, this file will house our function, name this file '`gene_expression.py`'
- Inside of this file, create a function called `expression()` that will take two input parameters: your input file and the name you want for your output file
- In your function, create a variable that will make your output_filename a file that we can open and write information into. For this function the 'w' stands for write, 'r' would be to read in the file, and 'a' is to append the file if we just want to add information to what is already in the file.

```
out = open(output_filename 'w')
```

- Inside of your function, open your input file and read in each line like this:

```
for line in open(input_file):
```

- Now we will take the data in each line and split it apart into its different aspects using strip, split and slicing.
    - We will take each line and strip it of any white space characters on either end using .strip().
    - Next we will split the values in the line by tab spaces using '\t', if our values were separated by a single space we could use .split(' ') or for comma separation .split(',').
    - Now the variable information is a list of all values on this line. So through slicing, we can assign the first value at index 0 to the gene_name, and then the remaining values, starting at index 1, as gene expression values. Our function should now look like this:

```
def expression(input_file, output_filename):
      out = open(output_filename, 'w')
      for line in open(input_file):
            information = line.strip().split('\t')
            gene_name = information[0]
            expression_values = information[1:]
```

- Now we will take the average of our expression values to create our new output.
    - We need to change all of the values in our list to floats, we can convert the data type of an entire list using the map() function, which then needs to be cast to a list().
    - Next we will take the average of our list, either by using the numpy.mean function (look this up and see how you can import numpy into your script to use this) **OR** by adding all values in our list using sum() and dividing by the length of our list using len().

```
def expression(input_file, output_filename):
      out = open(output_filename, 'w')
      for line in open(input_file):
            information = line.strip().split('\t')
            gene_name = information[0]
            expression_values = information[1:]
            expression_values =
      list(map(float,expression_values))
            expression_avg =
            sum(expression_values)/len(expression_values)
```

OR

```
def expression(input_file, output_filename):
      import numpy as np
      out = open(output_filename, 'w')
      for line in open(input_file):
            information = line.strip().split('\t')
            gene_name = information[0]
            expression_values = information[1:]
            expression_values =
      list(map(float,expression_values))
            expression_avg = np.mean(expression_values)
```

- Next we want to take the gene name and average expression of this gene and write it to our output file.

- We can do this using the .join() function to put our data together in a desired format, here we will be joining our data with tab spaces in between them using '\t', again, you can join using commas ',' or a single space ' ', this is really a personal preference.
- The data will be written using .write(). We need to make sure to add a new line character to the end of every line in our new file, or no white space will be added.
- Once this is done and we have looped through every line in our input code, we will want to close the output file using .close().

```
def expression(input_file, output_filename):
    out = open(output_filename, 'w')
    for line in open(input_file):
        information = line.strip().split('\t')
        gene_name = information[0]
        expression_values = information[1:]
        expression_values =
        list(map(float,expression_values))
        expression_avg =
        sum(expression_values)/len(expression_values)
        new_data = [gene_name, str(expression_avg)]
        out.write('\t'.join(new_data) + '\n')
    out.close()
```

- Finally, when calling our function in the script, we need to tell the script where to find our data file. In this case, make sure that your script and data file are in the same directory.
    - Import os and sys to use the functions we need to build a path to our file.
    - Next we will create a variable which will point our script to our file by combining the path we feed it as well as the name of the data file. In this case, we are using 'sys.path[0]' to tell it our file is in the same directory.

```
def expression(input_file, output_filename):
    out = open(output_filename, 'w')
    for line in open(input_file):
        information = line.strip().split('\t')
        gene_name = information[0]
        expression_values = information[1:]
        expression_values =
        list(map(float,expression_values))
        expression_avg =
        sum(expression_values)/len(expression_values)
        new_data = [gene_name, str(expression_avg)]
        out.write('\t'.join(new_data) + '\n')
        out.close()
```

```
import os,sys
data = os.path.join(sys.path[0], 'mock_data.txt')
convert_data = expression(data, 'mock_avg.txt')
```

- The way we built our script, no print statements will be returned, so check your directory for our output file and see if everything looks good within this new file. If you are unsure about your script add print statements as you go to check your work, I will add some comments below to show places you might want to add print statements in the beginning to check your code:

```
def expression(input_file, output_filename):
      out = open(output_filename, 'w')
      for line in open(input_file):
            information = line.strip().split('\t')
            gene_name = information[0]
            #print(gene_name)
            expression_values = information[1:]
            expression_values =
            list(map(float,expression_values))
            #print(expression_values)
            Expression_avg =
            sum(expression_values)/len(expression_values)
            #print(expression_avg)
            new_data = [gene_name, str(expression_avg)]
            #print(new_data)
            out.write('\t'.join(new_data) + '\n')
      out.close()

import os,sys
data = os.path.join(sys.path[0], 'mock_data.txt')
convert_data = expression(data, 'mock_avg.txt')
```