UNIX Shell

- Why:
    - Ideal for Working with folders/files & Big Data
- **Files & Directories:**
    - Filesystem = Organizes data into files (folders)
        - Root = top directory. Indicated by /
        - Below that any subfolders ex: /users/____
    - Commands:
        - **$** = command line where to type commands
        - **whoami** = username, current user (runs small program then returns to prompt)
        - **pwd** = to know current location/path (print working directory)
        - **ls** = Looking at content w/in fs. Prints names of files & directories in alphabetical order.
        - **ls ____** = ls + foldermame
        - **ls -F** = organizes by folders
        - **ls -F desktop** = view contents of desktop
        - **ls -a** = SHOW ALL (hidden folders, ./, .//)
        - **cd ____** = (Change Directory) Change location; go to certain folder ex: cd sd-workshop
        - **cd /** = to root folder
        - **cd ~** or **cd** = to home directory
        - **cd( )** = Type first few letters then TAB does auto-complete. Shortcut to typing folder names
        - **cd .** = current working directory
        - **cd ..** = takes you back one level. Parent of current directory.
        - **mkdir** = Make directory. Make new folder
        - Creating new file:
            - Program name + space + filename
                - Ex: **notepad draft.txt**
                - *Note: Typing notepad.draft.tx & ← Allows you to edit document & still work in the terminal
        - **rm ____** = remove or delete ex: rm draft.txt
            - Only works on files, not directories
        - **rmdir ____** = removes directories
            - Cannot remove if file within
            - May use rm thesis/draft.txt
        - **mv** = rename (move)
            - Ex: mv thesis/draft.txt thesis/quotes.txt
        - **mv .** = move a file from directory it was in, into current
            - Ex: mv thesis/quotes.txt .
        - **cp** = copies file instead of moving it
            - Ex: cp quotes.txt thesis/quotations.txt

        - Show changes: Ls quotes.txt thesis/quotes.txt thesis/quotations.txt
- **wc** = wordcount
    - Ex: wc *.pdb ← asterisk tells shell to read all files with that extension
    - Outputs columns: 1 number of lines 2 wordcount 3 characters
- **wc -l** = line count only
    - Ex: wc -l *.pdb
- **wc -l >** = outputs info to new file
    - Ex: wc -l *.pdb > lengths.txt
- **wc*.txt|wc** = counts number of lines calculated after wc the files
- **>** = redirects output to new file
    - Ex: wc -l *.pdb > lengths.txt
- **cat** = concatenate. Prints contents of file 1 after another. Shows script contents.
    - Ex: cat lengths.txt
- **sort**
    - Ex: sort -n lengths.txt (n for numeric)
    - Sorts from smallest to largest
    - Save sorted list into new file:
        - sort -n lengths.txt > sorted-lengths.txt
- **head -#** = pulls first few lines within file
    - Ex: head -1 sorted-lengths.txt
    - Ex: head -n 1 (same thing)
    - Shows only the first line of the file
    - Output of sorted list here shows shortest file
- **tail** = pulls last lines within file
    - Ex: tail - 5 sorted-lengths.txt
- **|** = Pipe. Keeps each command in memory & combines them. Runs one command instead of executing one at a time. Tells shell we want to use output of command on LEFT (sort) as input to command on RIGHT (head)
    - Take original wc & sort numerically
    - Ex: wc -1 *.pdb | sort -n | head -1
- **man** = Manual command. type man then command you want more info about
    - In Windows: --help
    - Ex: wc --help
- **ls** = can be used to search within filenames as well
    - Ex: ls *Z.txt

- **Loops**
  - View first 3 lines of 2 files:
    - **For filename in basilisk.dat unicorn.dat**
    - **> do**
    - **> head -n 3 $filename**
    - **> done**
  - Print first 100 lines & last 20 lines of files
    - **For filename in *.dat**
    - **>Do**
    - **> Echo $filename**
    - **> Head -n 100 $filename | tail -n 20**
    - **>Done**
      - (Echo prints parameters to output)
  - Rename multiple file names
    - **For filename in *.dat**
    - **>Do**
    - **> Cp $filename original-$filename**
    - **>Done**
      - Output: basilisk.dat, original-basilisk.dat
  - Create loop with only certain file types desired
    - **For datafile in *[AB].txt**
    - **>do**
    - **> echo $datafile**
    - **>done**
      - Brackets indicate OR in between, not AND
      - If A then B or C, could do A[B] or A[C]
  - Prefix each with stats
    - **For datafile in *[AB].txt**
    - **>Do**
    - **> Echo $datafile stats-$datafile**
    - **>Done**
  - Running program or script that is processing files and outputs
    - **For datafile in *[AB].txt**
    - **>Do**
    - **> Echo $datafile stats-$datafile**
    - **> bash goostats**
    - **>Done**

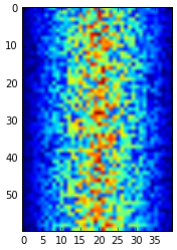
- **Exercises**
    - Question: Absolute vs relative paths
        - Starting from **/users/amanda/data**, which commands could she use to navigate to home directory, which is **/users/amanda**?
            - 5. cd ~
            - 8. cd
            - 9. cd ..

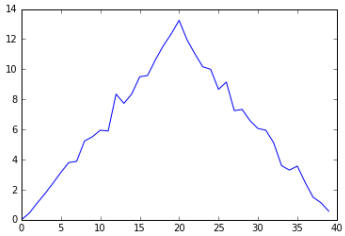
    - Question: Renaming files
        - Suppose you created a .txt file in your current dir to contain a list of stat tests named statstics.txt. You realized you mispelled & want to correct, you can use:
            - mv statstics.txt statistics.txt
            - cp statstics.txt statistics.txt ← will keep old incorrect file
        - In current dir, want to find 3 files which have least # of lines. Which command would work?
            - wc -l* | sort -n | head -n 3
                - Wordcount with lines, sort, then head 3 for top 3 lines
    - Variables in Loops:
        - Loop 1:
            - for datafile in *.dat
            - >do
            - > ls *.dat
            - >done
                - Output:
                    - basilisk.dat unicorn.dat
                    - basilisk.dat unicorn.dat
        - Loop 2:
            - for datafile in *.dat
            - >do
            - > ls $datafile
            - >done
                - Output:
                    - Basilisk.dat
                    - unicorn.dat
        - Why are these 2 different?
            - Loop 1, las *.dat processes dat files (multiple)
            - Loop 2 processes 1 at a time (only the 2)

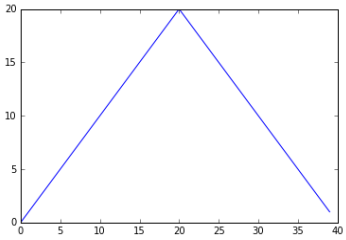
PYTHON

- **Shift + Enter** for initializing command (running cell)
- Commands:
    - **import numpy** = Import directory. Able to read in csv files
    - **numpy.loadtxt( )** = fn call. Runs loadtxt. Numpy is the thing, file is loadtxt.
        - Numpy turns data into matrix
        - Delimiter separates elements by ,
        - Ex: numpy.loadtxt(fname='inflammation-01.csv', delimiter = ',')
    - **___ = variable** = Create variables
        - Must start with letter
        - Ex: weight_kg = 55
    - **print( )** = prints variables
        - Ex: print(weight_kg)
        - Python2: NO PARENTHESIS
            - Print weight_kg
    - **___?** = help with function
        - Ex: print?
    - **print(data)**
        - matrix
    - **print(type(data))**
        - Shows type
    - **print(data.shape)**
        - Gives length
    - ATTRIBUTE
        - don't need parenthesis
        - Ex: print(data.shape)
    - PARENTHESIS
        - used when calling a function
    - Load array as variable:
        - Ex: data = numpy.loadtxt(fname = 'inflammation-01.csv', delimiter=',')
    - Indexing:
        - STARTS AT ZERO
        - Array shows UP TO but NOT EQUAL to
        - Ex: data[0:4,0:10]
            - Rows 0, 1, 2, 3
            - Columns 0 through 9
        - Use colons instead, assumes 0
            - Ex: data[:4,:10]
    - Axes:
        - 0 = down columns (col)
        - 1 = across columns (rows)

- Example:
  - **Doubledata = data * 2.0**
    - Multiplies all by 2
  - **Doubledata[:3, 36:]**
    - Indexing rows 0 to 3 & rows 36 to end
  - **Tripledata = Doubledata + data**
    - Can add matrices since they're the same size
  - **print Tripledata**
  - **print data.mean()**
    - Mean is descriptive, itself is a function you can pass values into, but here it will get the mean of ALL the values in the matrix
  - Max, Min, Standard deviation:
    - print 'maximum inflammation:', data.max()
    - print 'minimumum inflammation:', data.min()
    - print 'standard deviation', data.std()
    - print(data.max(axis=0).shape)
  - 2 ways of doing same thing:
    - **data.max()** ← data is object
    - **numpy.max(data)** ← data is argument being called
- Example (using program matplotlib.pyplot)
  - import matplotlib.pyplot
  - %matplotlib inline ← create image
  - image = matplotlib.pyplot.imshow(data)
    - Shows entire plot of data
  - ave_inflammation = data.mean(axis=0)
    - Creates variable for axis 0
  - ave_plot = matplotlib.pyplot.plot(ave_inflammation)
    - Plots subset of new variable
  - matplotlib.pyplot.show(ave_plot)
    - Shows
  - max_plot = matplotlib.pyplot.plot(data.max(axis=0))
  - min_plot = matplotlib.pyplot.plot(data.min(axis=0))
  - std_inflammation = matplotlib.pyplot.plot(data.std(axis=0))







- Slice = Section of array
  - Example:
    - element = 'oxygen'
    - print 'first three character:', element[0:3]

        - print 'last three characters:', element[3:6]
            - Output:
                - first three character: oxy
                  last three characters: gen
    - Example:
        - element = 'oxygen'
        - print 'first four character:', element[:4]
        - print 'last four characters:', element[4:]
        - print 'all characters:', element[:]
            - Output:
                - first four character: oxyg
                  last four characters: en
                  all characters: oxygen

    - Examples:
        - -# = gets back to elements
            - Ex: element[-1] = n
            - Ex: element[-2] = e
            - Ex: element[1:-1] = xyge
                - Goes from index 1 to index -1 from last
- Loops
    - Example of bad way:
        - **word = 'tin'**
        - **print word[0]**
        - **print word[1]**
        - **print word[2]**
        - **print word[3]**
            - Get ERROR since char dimension mismatch, so we use a LOOP instead

```
t
i
n
---------------------------------------------------------------------
------
IndexError                                Traceback (most recent call
last)
<ipython-input-62-7974b6cdaf14> in <module>()
      3 print word[1]
      4 print word[2]
----> 5 print word[3]

IndexError: string index out of range
```

    - Loop version:
        - **word = 'oxygen'**
        - **for char in word:**
        - **   print char**
            - char = Defining loop variable (arbitrary; could have called it anything else)

```
word = 'oxygen'
for char in word:
    print char
```

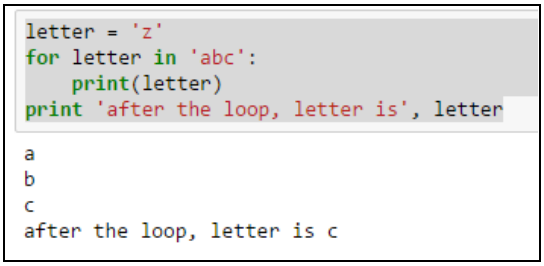
```
o
x
y
g
e
n
```

    - Example:
        - **length = 0**
        - **for vowel in 'aeiou':**
        - **   length = length + 1**
        - **print 'there are', length, 'vowels'**

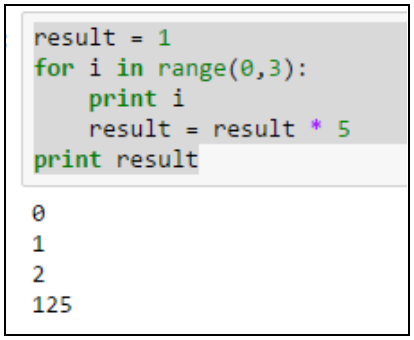
```
length = 0
for vowel in 'aeiou':
    length = length + 1
print 'there are', length, 'vowels'
```

```
there are 5 vowels
```

        - Note: Loop variable exists after loop completes
    - Example:
        - **letter = 'z'**
        - **for letter in 'abc':**
        - **   print(letter)**
        - **print 'after the loop, letter is', letter**



    - Example:
        - Using **RANGE**
            - Range can accept 1 to 3 parameters.
            - If there are 2 parameters, it indicates the range only
                - Ex: (2,5) = 2, 3, 4
            - If there are 3 parameters, it indicates 1. First value, 2. Last value, 3. Increment value
                - Ex: (3, 10, 3) = 3, 6, 9
    - Example:
        - How can we create LOOP instead of EXPONENTIATION? (5**3 = 125)
            - **result = 1**
            - **for i in range(0,3):**
            - **   print i**
            - **   result = result * 5**
            - **print result**
                -



- Lists
    - Can change elements in list, can't do in a string.
    - Commands:
        - **append** = function that adds to end of list
            - Ex: odds.append(11)
            - Can also use **+=**
        - **delete**
            - Ex: del odds[0]
        - **reverse**
            - Ex: odds.reverse()
    - Example:
        - **names = ['Newton', 'Darwing', 'Turing']**
        - **print 'names is originally:', names**
        - **names[1] = 'Darwin'**
        - **print "final value of names:", names**
            - Names[1] indexes Darwing which is in index 1

- Example:
  - **odds = [1,3,5,7]**
  - **primes = odds**
  - **primes += [2]**
    - Adds 2 to end, shortcut append
  - **print 'primes', primes**
  - **print 'odds', odds**

```
odds = [1,3,5,7]
primes = odds
primes += [2]
print 'primes', primes
print 'odds', odds
```

```
primes [1, 3, 5, 7, 2]
odds [1, 3, 5, 7, 2]
```

- Example: How can you get the word hello to print as a list: 'h' 'e' 'l' 'l' 'o'
  - Option 1:
    - **my_list = [ ]**
    - **for char in 'hello':**
    - ** my_list += char**
    - **print my_list**
  - Option 2:
    - **my_list = [ ]**
    - **for char in 'hello':**
    - ** my_list.append(char)**
    - **print my_list**
  - Option 3:
    - **my_list = [ ]**
    - **for char in 'hello':**
    - ** my_list += list(char)**
    - **print my_list**

```
my_list = []
for char in 'hello':
    my_list += char
print my_list
```

```
['h', 'e', 'l', 'l', 'o']
```

```
my_list = []
for char in 'hello':
    my_list.append(char)
print my_list
```

```
['h', 'e', 'l', 'l', 'o']
```

```
my_list = []
for char in 'hello':
    my_list += list(char)
print my_list
```

```
['h', 'e', 'l', 'l', 'o']
```

Building programs with PYTHON part2

- Importing built-in package
  - Import glob (global object)
  - Inside is function called glob that is used for matching
    - Ex: **glob.glob('inflammation*.csv')**
      - Grabs all inflammation files
      - Outputs LIST of character strings denoted by [ ]
  - Add Loop:

```
In [3]: glob.glob('inflammation*.csv')
Out[3]: ['inflammation-01.csv',
         'inflammation-02.csv',
         'inflammation-03.csv',
         'inflammation-04.csv',
         'inflammation-05.csv',
         'inflammation-06.csv',
         'inflammation-07.csv',
         'inflammation-08.csv',
         'inflammation-09.csv',
         'inflammation-10.csv',
         'inflammation-11.csv',
         'inflammation-12.csv']
```

```
In [4]: count = 0
        for filename in glob.glob('*.csv'):
            count = count + 1
        print "number of files:", count

        number of files: 15
```

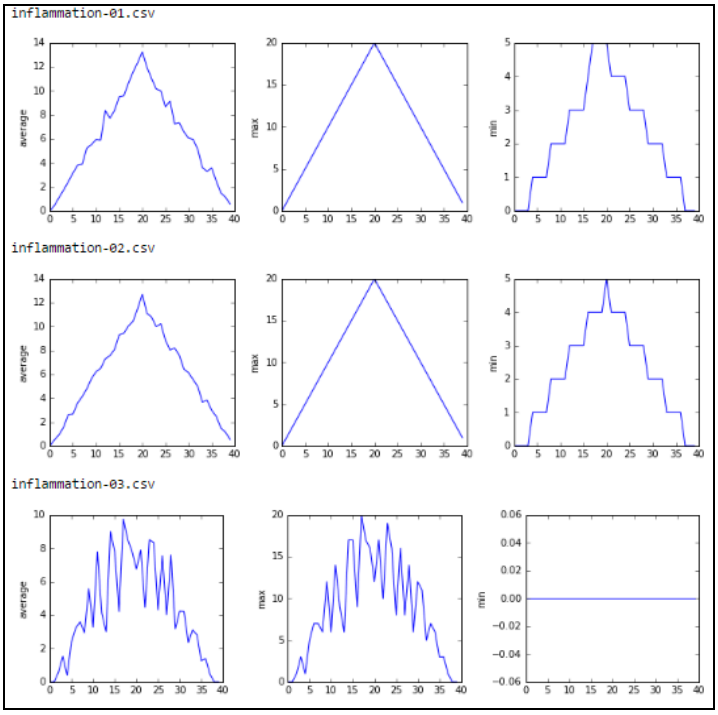
Data analysis

- Example:
  - **import numpy**
  -
  - **count = 0**
  - **for filename in glob.glob('inflammation*.csv'):**
  - **data = numpy.loadtxt(fname = filename, delimiter=',')**
  - **print filename, "mean is:", data.mean()**
  - **count += 1**
  - **print "number of files:", count**
    - Gives MEAN across entire matrix for ea file
    - Numpy program requires 2 input arguments: filename, delimiter

```
In [7]: count = 0
        for filename in glob.glob('inflammation*.csv'):
            data = numpy.loadtxt(fname = filename, delimiter=',')
            print filename, "mean is:", data.mean()
            count += 1
        print "number of files:", count

        inflammation-01.csv mean is: 6.14875
        inflammation-02.csv mean is: 5.99083333333
        inflammation-03.csv mean is: 4.20458333333
        inflammation-04.csv mean is: 6.10958333333
        inflammation-05.csv mean is: 6.11833333333
        inflammation-06.csv mean is: 6.04333333333
        inflammation-07.csv mean is: 6.01958333333
        inflammation-08.csv mean is: 4.20458333333
        inflammation-09.csv mean is: 6.03291666667
        inflammation-10.csv mean is: 6.0525
        inflammation-11.csv mean is: 4.20458333333
        inflammation-12.csv mean is: 6.06166666667
        number of files: 12
```

- Example:
  - import matplotlib.pyplot
  - %matplotlib inline

```
count = 0
for filename in glob.glob('inflammation*.csv')[0:3]:
    count += 1
    print(filename)

    data = numpy.loadtxt(fname = filename, delimiter = ',')

    fig = matplotlib.pyplot.figure(figsize = (10.0,3.0))

    #defining subplot in each figure space. 1 row, 3 plots in figspace
    #fig1 labels are arbitrary
    fig1 = fig.add_subplot(1,3,1)
    fig2 = fig.add_subplot(1,3,2)
    fig3 = fig.add_subplot(1,3,3)

    #set properties of graphs
    fig1.set_ylabel('average')
    fig1.plot(data.mean(axis=0))

    fig2.set_ylabel('max')
    fig2.plot(data.max(axis=0))

    fig3.set_ylabel('min')
    fig3.plot(data.min(axis=0))

    #figtight is cosmetic to automatically adjust spacing
    fig.tight_layout()
    matplotlib.pyplot.show(fig)
```



- Save image add:
  - **fig = matplotlib.pyplot.savefig(filename + '.png')** -or-
  - **fig = matplotlib.pyplot.savefig(filename.replace('csv','png'))**
- Add count to bottom:
  - **print "number of files:", count**

- Use Python to make CHOICES/decisions
  - conditionals
  - logical
  - elseif
    - **elif** = shorthand for elseif

```
num = 37
if num > 100:
    print 'greater'
else:
    print 'not greater'
```

```
not greater
```

```
num = 53
print 'before conditional ...'
if num > 100:
    print '53 is greater than 100'
print '... after conditional'
```

```
before conditional ...
... after conditional
```

```
num = -3
if num > 0:
    print num, 'is positive'
elif num == 0:
    print num, 'is zero'
else:
    print num, 'is negative'
```

```
-3 is negative
```

```
if (1 > 0) and (-1 > 0):
    print 'both parts are true'
else:
    print 'at least one part is false'
```

```
at least one part is false
```

```
if (1 < 0) or (-1 < 0):
    print 'at least one test is true'
```

```
at least one test is true
```

- Check for suspicious looking data:

```
data = numpy.loadtxt(fname = 'inflammation-01.csv', delimiter= ',')

if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
    print 'suspicious looking max'
elif data.min(axis = 0).sum() == 0:
    print 'minima add up to zero!'
else:
    print 'seems ok'
```

```
suspicious looking max
```

*Make loop to check all files!

Create Functions

- Temperature Conversion (F to K)
  - def far_to_kelvin(temp)
    - Def = definition
    - ( ) = where arguments go
  - return ((temp - 32) * (5/9)) + 273.15
    - Example:
      - far_to_kelvin(82)
      - = 273.15

```
In [40]: def far_to_kelvin(temp):
             return ((temp - 32) * (5/9)) + 273.15

In [41]: far_to_kelvin(82)
Out[41]: 273.15
```

```
def far_to_kelvin(temp):
    kel = ((temp - 32) * (5/9)) + 273.15
    return kel
```

```
In [44]: def far_to_kelvin(temp):
             kel = ((temp - 32) * (5/9)) + 273.15
             return kel

In [45]: print 'freezing point of water', far_to_kelvin(32)
         print 'boiling point of water', far_to_kelvin(212)

         freezing point of water 273.15
         boiling point of water 273.15

In [46]: def kelvin_to_celsius(temp_k):
             return temp_k - 273.15

In [47]: print 'absolute zero in Celsius', kelvin_to_celsius(0.0)

         absolute zero in Celsius -273.15

In [48]: def far_to_celsius(temp_f):
             temp_k = far_to_kelvin(temp_f)
             result = kelvin_to_celsius(temp_k)
             return result

In [50]: print('freezing point of water in celsius:', far_to_celsius(32.0))

         ('freezing point of water in celsius:', 0.0)
```

- Challenge: Write a function called fence that takes 2 parameters & returns new string w/ wrapper as beginning & end
    - def fence(original,wrapper):
    - return wrapper + original + wrapper

```
In [94]: def fence(original,wrapper):
             return wrapper + original + wrapper

In [95]: print(fence('face','*'))

         *face*
```

- Analyze Function:

def analyze(filename):

#print(filename)

data = numpy.loadtxt(fname = filename, delimiter = ',')

fig = matplotlib.pyplot.figure(figsize = (10.0,3.0))

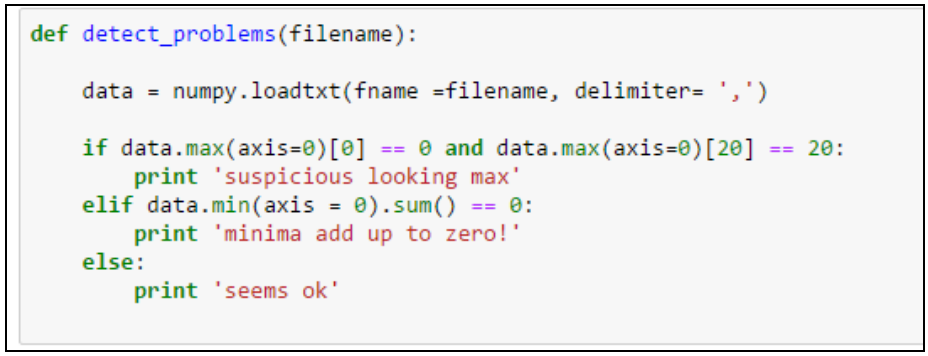
#defining subplot in each figure space. 1 row, 3 plots in figspace
#fig1 labels are arbitrary

```
fig1 = fig.add_subplot(1,3,1)
fig2 = fig.add_subplot(1,3,2)
fig3 = fig.add_subplot(1,3,3)

#set properties of graphs
fig1.set_ylabel('average')
fig1.plot(data.mean(axis=0))

fig2.set_ylabel('max')
fig2.plot(data.max(axis=0))

fig3.set_ylabel('min')
fig3.plot(data.min(axis=0))

#figtight is cosmetic to automatically adjust spacing
fig.tight_layout()

#matplotlib.pyplot.show(fig) will populate new window
fig = matplotlib.pyplot.savefig(filename + '.png')
```

```
def analyze(filename):

    #print(filename)

    data = numpy.loadtxt(fname = filename, delimiter = ',')

    fig = matplotlib.pyplot.figure(figsize = (10.0,3.0))

    #defining subplot in each figure space. 1 row, 3 plots in figspace
    #fig1 labels are arbitrary
    fig1 = fig.add_subplot(1,3,1)
    fig2 = fig.add_subplot(1,3,2)
    fig3 = fig.add_subplot(1,3,3)

    #set properties of graphs
    fig1.set_ylabel('average')
    fig1.plot(data.mean(axis=0))

    fig2.set_ylabel('max')
    fig2.plot(data.max(axis=0))

    fig3.set_ylabel('min')
    fig3.plot(data.min(axis=0))

    #figtight is cosmetic to automatically adjust spacing
    fig.tight_layout()

    #matplotlib.pyplot.show(fig) will populate new window
    fig = matplotlib.pyplot.savefig(filename + '.png')
```

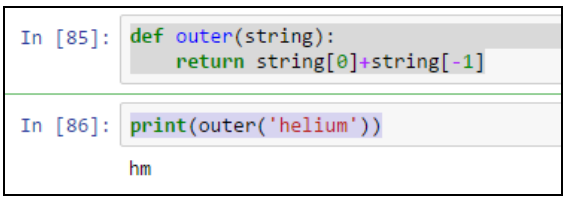
- Detect problems function:

```
def detect_problems(filename):

    data = numpy.loadtxt(fname =filename, delimiter= ',')

    if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
        print 'suspicious looking max'
    elif data.min(axis = 0).sum() == 0:
        print 'minima add up to zero!'
    else:
        print 'seems ok'
```



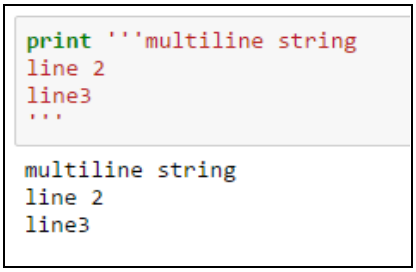
```
def detect_problems(filename):

    data = numpy.loadtxt(fname =filename, delimiter= ',')

    if data.max(axis=0)[0] == 0 and data.max(axis=0)[20] == 20:
        print 'suspicious looking max'
    elif data.min(axis = 0).sum() == 0:
        print 'minima add up to zero!'
    else:
        print 'seems ok'
```

- Challenge: Write a function called outer that returns a string made up of just the FIRST & LAST characters of its input.
  - Function:
    - **def outer(string):**
    - **return string[0]+string[-1]**
  - Call:
    - **print(outer('helium'))**
  - Output:
    - hm



```
In [85]: def outer(string):
             return string[0]+string[-1]

In [86]: print(outer('helium'))
         hm
```

- ‘’’ = **Docstring.** Triple quotes. Used for multiple line strings



```
print '''multiline string
line 2
line3
'''
```

```
multiline string
line 2
line3
```

PANDAS

- **import pandas** = import program
- **pandas.read_csv('A1_mosquito_data.csv')** = tell which csv file to read in
    - pandas.read + tab = can see all the ways to read in data

```
In [1]: import pandas
```

```
In [19]: pandas.read_csv('A1_mosquito_data.csv')
```

Out[19]:

	year	temperature	rainfall	mosquitos
0	2001	80	157	150
1	2002	85	252	217
2	2003	86	154	153
3	2004	87	159	158
4	2005	74	292	243
5	2006	75	283	237
6	2007	80	214	190
7	2008	85	197	181
8	2009	74	231	200
9	2010	74	207	184

```
In [5]: data = pandas.read_csv('A1_mosquito_data.csv')
```

```
In [6]: print(data)
```

```
   year  temperature  rainfall  mosquitos
0  2001           80       157        150
1  2002           85       252        217
2  2003           86       154        153
3  2004           87       159        158
4  2005           74       292        243
5  2006           75       283        237
6  2007           80       214        190
7  2008           85       197        181
8  2009           74       231        200
9  2010           74       207        184
```

```
In [7]: print(type(data))
<class 'pandas.core.frame.DataFrame'>
```

```
In [8]: print(data['year'])
0    2001
1    2002
2    2003
3    2004
4    2005
5    2006
6    2007
7    2008
8    2009
9    2010
Name: year, dtype: int64
```

```
In [9]: print(data[['rainfall','temperature']])
# must include the double brackets
   rainfall  temperature
0       157           80
1       252           85
2       154           86
3       159           87
4       292           74
5       283           75
6       214           80
7       197           85
8       231           74
9       207           74
```

```
In [15]: d2 = data[['rainfall','temperature']]
```

```
In [17]: print(d2)
   rainfall  temperature
0       157           80
1       252           85
2       154           86
3       159           87
4       292           74
5       283           75
6       214           80
7       197           85
8       231           74
9       207           74
```

```
In [10]: print(data[0:2])
   year  temperature  rainfall  mosquitos
0  2001           80       157        150
1  2002           85       252        217
```

```
In [13]: print(data[1:2])
   year  temperature  rainfall  mosquitos
1  2002           85       252        217
```

```
In [20]: data.iloc[1]
# indexes locations
Out[20]: year           2002
         temperature      85
         rainfall        252
         mosquitos       217
         Name: 1, dtype: int64
```

In [21]:
```
print(data['temperature'][data['year'] > 2005])
```

```
5    75
6    80
7    85
8    74
9    74
Name: temperature, dtype: int64
```

In [22]:
```
print(data['temperature'][data['year'] > 2005])
# Prints only temps in years ABOVE 2005
```

```
5    75
6    80
7    85
8    74
9    74
Name: temperature, dtype: int64
```

In [ ]:
```

```

GIT & GITHUB

- Version Control
  - In general, is the “lab notebook for the digital world”
  - version control sys
  - Useful working in TEAM, but also alone
  - Used instead of having many copies of files, get to track changes
  - Any file type
  - **Divergent** = multiple people working on same doc
  - **Convergent** = eventually **merge** back to same doc
- Terms
  - **Commit** = New changes
  - **Repository** = Storage area where tem stores full history, diff b/w files. Will only see the one file, but can see changes within.
  - VCS = Version control system - CVS, Subversion, RCS
    - Using central server
  - DVCS = Distributed Version control system - Git, Mercurial
    - Distributed - no central server anymore. Repository created on your laptop is the entire database. No dependencies - no internet or connection to server needed.

GIT

- <u>Configuration:</u>
  - **git config --global user.name “ “** = Tell username
    - Global - allows to use config for all projects
    - (unless you want separate for work/personal/etc)
  - **git config --list or git config -l**
    - View settings
  - **git config --global user.email “ “** = Tell email address
  - **git config --global core.editor** = Tell git which editor you want to use
    - Doesnt bind you
    - Create commit - if you dont enter a message, it will use this editor to open & have you enter a message
    - Any time git needs you to add input, but can use any text editor wanted
  - **git config --global core.autocrlf true** = for Windows. Able to collaborate with mac/windows
- <u>Create Repository:</u>
  - **Mkdir**
  - **git init** = Initializes for git use. If just created directory. Initialize empty git repository
  - **ls -a** = to see path & hidden files
  - **ls -la** = to see files in list view, top to bottom

Git will recognize we created something new, and will tell us when they're not committed. Must add those files.

- STEPS:
    - 1) Edit file
    - 2) Add file to staging area (shopping cart)
        - **Git add (filename)** = Adds file
    - 3) Commit
        - **Git commit -m** "start notes on Mars as a base"
            - -m passes a message, want detailed, 50 character for good practice
- General Commands:
    - **git status** = can run anytime to see whats going on, where you are, hints as to what to do next,
    - **git log** = See commits/changes done so far
    - **git diff** = looks at changes, recently committed & current file, shows difference
        - **Git diff (filename)** = changes only within that file
    - **git reset (filename)** = reset before all changes, use with caution
    - **git checkout (filename)** = checks out MOST RECENTLY edited version, can go back to others if specified
        - **HEAD** = most recent commit. DEFAULT.
        - **~** = Tilda is number of commits behind the head
        - **Git diff HEAD~ (filename)** or **Git diff HEAD~2 (filename)**
        - **Git checkout HEAD mars.txt**
    - **rmdir .git = removes directory from git**
        - If full, might have to force:
        - Rm -rf .git
    - **touch** = quick way to generate files if none found under that name
        - Ex: touch a.dat b.dat c.dat results/a.out results/b.out
            - Created 3 files in current dir & 2 files in results
    - **notepad .gitignore** = creates files that tells git to ignore
        - *Important to be in ROOT of projectx`x`
        - Ex: Be in planets folder
    - **git status --ignored** = shows everything being ignored w/o having to open the txt file
    - **!** = exception. Ignore just one file but keep all others
        - Ex: within .gitignore file
        - *.dat
        - !(filename) ← ignore all EXCEPT this one
    - **#** = comments

Challenges

- Which command(s) would save changes of myfile.txt to local Git repo?
    - Git add myfile.txt

    - Git commit -m “my recent changes”
- How would you ignore a subdirectory?
    - results/data
    - results/plots
        - Want to ignore just plots:
            - **.gitignore** file with **results/plots***

GITHUB

- Create repository
- Next screen: List of options to connect locally
    - Able to dl GUI interface
    - HTTPS - able to copy link
    - Can push existing code (which we set up)
    - Copy + paste code into command window
- Or from command line:
    - Cd into directory
    - Follow commands:

```
echo “My resume” >> README.md
git init
git add .
git commit - “first commit”
git config --global user.email “your email”
git config --global user.name “your name”
git remote add origin your-repository-url
git push -u origin master
```

        -
- Pushing repository to github
    - **Git remote -v** ← checks that connected
    - **Git push origin master** ← sends contents of repository to github
        - Inverse: if you have done this, someone else made changes, git origin push & you dont have those changes, you can PULL
        - **Git pull origin master**
    - ***Good practice to PULL before you PUSH***
- Pushing vs Committing
    - Push = Globally
    - Commit = Locally