Welcome to The Carpentries Etherpad!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try etherpad.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

All content is publicly available under the Creative Commons Attribution License: https://creativecommons.org/licenses/by/4.0/

Software Carpentry Code of Conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

Post-Workshop Survey: https://www.surveymonkey.com/r/swc_post_workshop_v1?workshop_id=2019-05-23-shapiro
Mailing List (for Future Workshops): https://mcommunity.umich.edu/#group:Software%20Carpentry%20Updates
Interested in Being a Helper or Instructor? https://forms.gle/Yk6KZ24incickBDf6

Instructors:

- Dave Bridges
- Mike Wolfe

Helpers

- Kelly Sovacool
- Jingqun Ma
- Morgan Oneka
- Rucheng Diao

Red Sticky: I need help
Blue Sticky: I am good to go

Before We Start

- Grab a coffee/tea if you like, initial the sign-in sheet, get one red and one blue sticky note. Fill out your name tag.
- Please do the Pre-Survey Link: https://www.surveymonkey.com/r/swc_pre_workshop_v1?workshop_id=2019-05-23-shapiro
- Download the file https://swcarpentry.github.io/shell-novice/data/data-shell.zip, move it to your desktop and unzip it.
- Download https://swcarpentry.github.io/python-novice-inflammation/data/python-novice-inflammation-data.zip and https://swcarpentry.github.io/python-novice-inflammation/code/python-novice-inflammation-

code.zip, move it to your desktop and unzip it

- Check Your Installations. We want you to do this on your own computer if possible, so make sure each of these are installed on your computer. Instructions at https://davebridges.github.io/2019-05-23-shapiro/
    - Bash Shell
        - Windows Install https://gitforwindows.org/, once complete open git-bash from the start menu, a terminal should appear
        - OSX open a terminal, close it by closing the window
    - Nano
        - From the terminal type **nano**, the screen should switch to a blank text editing page. To get back to the terminal hold CTRL-X
        - The instructors will use nano mostly, but if you want to use another text editor (vim, sublime, notepad++) that is also ok.
    - Python
        - Please follow the instructions at the bottom of https://davebridges.github.io/2019-05-23-shapiro/ to install python using anaconda. This will ensure you have all the required packages.
        - From terminal type **python.** Some information should appear in your terminal, including a new prompt (>>>). Type **exit()** and hit enter to get back to the shell
        - To open a jupyter notebook type **jupyter notebook**, a new web browser should appear, close it and then type CTRL-C at the terminal to exit the jupyter environment
        - To open an ipython session type **ipython** at the command prompt. To exit, Hit ctrl+d and it will ask you if you really want to exit. Press y and hit enter to exit
        - (Note for those **not** installing python through the recommended anaconda) You will need the following packages installed for this workshop:
            - numpy
            - matplotlib
            - pandas
    - Git
        - From the terminal type **git**. You should see a long message about the use of git
        - If you do not have an account already go to github.com and create and account

If you get anything else put up a red sticky and we will be by to help

This morning's lessons:

- **The Unix Shell**: https://swcarpentry.github.io/shell-novice/reference
- **Programming in Python**: https://swcarpentry.github.io/python-novice-inflammation/reference
- Just a little bit of **Version Control with Git**: https://swcarpentry.github.io/git-novice/reference

Tomorrow's lessons

- **Version Control with Git**: https://swcarpentry.github.io/git-novice/reference
- **Analysis and Visualization with Python** https://swcarpentry.github.io/python-novice-gapminder/reference

Command line tools
Yes* **********
8No*****

Nelles pipeline:
https://swcarpentry.github.io/ -novcdice/01-intro/index.htmlsca

Copy with Mutiple filenames (at bottom-ish of the link below)
https://swcarpentry.github.io/shell-novice/03-create/index.html

Goal is to get to north-pacific-gyre/2012-07-03

calculate number of lines in every text file
wc -l *.txt
wc -l *txt
wc -l *.txt
wc -l *txt
wc -l *txt
wc -l *.txt

now save that line count as a line_number.txt and put that command below

wc -l *txt > line_number.txt
wc -l *.txt > line_number.txt
wc -l *txt > line_number.txt
wc -l *.txt > line_number.txt
wc -l *.txt > line_number.txt

now try to only calculate the line count for files marked NENE01729A.txt and NENE01729B.txt
wc -l *1729[AB].txt

wc -l NENE01729?.*
wc -l NENE01729[AB].txt
wc -l NENE01729[AB].txt
wc -l NENE01729[AB].txt
wc -l NENE01729[AB].txt

write a script that lists the file names for all files ending with A or B in the north-pacific-gyre/2012-07-03
$ ls *[AB].txt
for file in NENE*[AB].txt; do echo $file; done
for file in *[AB].txt; do; echo $file; done
for i in NENE*[AB].txt; do echo $i; done
$ for file in 2012-07-03; do ls *[AB].txt; done

bash <FILESCRIPT> should give you the same result as your forloop

IF YOUR PRINT FUNCTION IS NOT WORKING IN PYTHON 2.7 ENTER THE FOLLOWING LINE IN YOUR PYTHON INTERPRETER
from __future__ import print_function

https://swcarpentry.github.io/python-novice-inflammation/01-numpy/index.html

What is element[-1]? What is element[-2]?
[-1] n
[-2] e
n, e
n, e
n, e
n,e

https://swcarpentry.github.io/py6thon-novice-inflammation/02-loop/index.html

Put solution from 1:N here:
66
6
6 times
6
6

```
 for nat in range(1,4):
    ...:     print(nat)
```

```
x = range(1,4)
   for n in x:
   print(n)
```

```
for char in range(1,4):
   print(char)
```

```
range = range(0,4)
for i in range:
       print(i)
```

https://swcarpentry.github.io/python-novice-inflammation/03-lists/index.html

Would your solution work regardless of whether you knew beforehand the length of the string or list (e.g. if you wanted to apply the solution to a set of lists of different lengths)?

```
print (list_for_slicing [-4:])
print(string_for_slicing [-4])
list_for_slicing[-4:]
```

https://swcarpentry.github.io/python-novice-inflalmmation/05-cond/index.html

https://swcarpentry.github.io/python-novice-inflammation/06-func/index.html

```
def fence(word,edge):
   return edge + word + edge

fence('hello','*')
Out[207]: '*hello*'

def fence(original, wrapper):
...     return(wrapper + original + wrapper)
```

Notes on Morning Git section:

- **git status** - allows you to see which files are being tracked
- **git init** - initializes a git repository
    - Have to make a decision at what level you want to track things
    - Don't want to run git init at the root of your file system as you would have to track everything in your file system in one repository (which is unmanagable)
- **touch filename** - allows you to make a new file called filename
- **cat filename** - print entire file to the screen
- **git add .** - add all files in directory to be tracked
- **git add filename** - add specific file to be tracked
- **git commit -m "message"** - commit with a message "message"
- **git log** - show history of all changes in the repo
- If you're getting warning/error messages about name or e-mail address configuration, run these commands:
    - **git config --global user.name "Your Name"**
    - **git config --global user.email you@example.com**
- **git log --oneline** - puts logs for each commit on one line
- **git diff HEAD-1** - look at the differences between the most up-to-date and the commit previous
- **git checkout filename** - undo changes that have happened since the last commit *for filename*
- **git checkout hashtag filename** - undo changes back to the hash for that commit for that file
- **git checkout master**- fix a detached HEAD
- HEAD - location in the chain of commits where you currently are. I.e.
    - * --- * --- * --- * --- * -- *
    - ^ - HEAD
- You could imagine changing head to an early iteration of your repository: **git reset HEAD hashtag**
    - * --- * --- * --- * --- * -- *
    - ^ - HEAD
- But if you want to get back to the end of the chain of commits you use
- **git checkout master**
- To ignore files create a **.gitignore** file and add the names of the files you want in there. This tells git to ignore these files

Git after morning break

- GitHub hosts repositories for you

        - Can be public
        - or private
    - To create a new repository click on green new button on upper left
    - You can choose a license with your code
    - Choose https and follow the "push from an existing repository" instructions
    - **git push -u origin master** - push your changes to a remote location (our github repository)
        - **master**: the branch you are working on
        - **origin**: where we are pushing our changes to (our github repository)
    - **git branch -** see all branches
    - **git remote -v** - see remote branches
    - **git branch newthing** - create a new branch called new thing
    - **git checkout newthing** - switch over to the new branch
    - Pushing a new branch to github
        - **git push origin newthing** - adds the new branch called newthing to github
    - **git merge newthing** - merges branch newthing into master (MUST BE ON MASTER before running this)
    - **git branch -d the_local_branch** - deletes a branch
    - Fork the other persons repository (makes a copy for you on the git website)
    - Clone or download and copy the link
    - Make a new directory for their repository NOT YOUR DATA-SHELL directory
    - **git clone copied_link** - copies the repository to your local repository
    - pull-request - the repository version of git merge
    - On the website there is a place to create a pull request
    - right hand side should be the changed repository, left hand side should be the original
    - Owner of the repository decides whether or not to accept the pull request

What kind of thing should we make a recipe for
salsa
guacamole**
Cake**********
jambalaya**
Pizza********
peanut butter and jelly - pbj*

Exercise: Make two or three commits on branch newthing

Exercise: Pair up and share each others repository links

Put up blue stickies when can see partner changes on repository

Exercise: switch places and submit pull request

Python Afternoon session:

We will start in the data/ directory from the analyzing patient data files we used yesterday.

Later we will use a different set of data. Please download and unzip the following directory that we will use for later on in the session:

https://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip

To start a jupyter notebook, navigate to the directory you would like to start it from and type:

jupyter notebook

This will open up a page on your web browser

For some of you, when you installed anaconda it never activated the anaconda "environment". If you are having issues getting jupyter notebook to work and you don't see **(base)** next to your command prompt type:

conda activate

And press enter. This should activate anaconda and allow you to use the version of python you installed with it.

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

http://swcarpentry.github.io/python-novice-inflammation/01-numpy/index.html

```
std_val = numpy.std(data,axis=0) #calculates std. dev over each column
std_plot = matplotlib.pyplot.plot(std_val)

std_over_patients = numpy.std(data, axis=0)
std_plot = matplotlib.pyplot.plot(std_over_patients)

std_patient = numpy.std(data, axis=0)
std_plot = matplotlib.pyplot.plot(std_patient)

std_over_patients = numpy.std(data, axis=0)
std_plot = matplotlib.pyplot.plot(std_over_patients)


std_over_patients= numpy.std(data, axis=0)
std_plot= matplotlib.pyplot.plot(std_over_patients)
```

Making a figure with multiple plots:

```
axes1 = fig.add_subplot(1, 3, 1)
axes2 = fig.add_subplot(1, 3, 2)
axes3 = fig.add_subplot(1, 3, 3)

axes1.set_ylabel('average')
axes1.plot(numpy.mean(data, axis=0))
```

```
axes2.set_ylabel('max')
axes2.plot(numpy.max(data, axis=0))

axes3.set_ylabel('min')
axes3.plot(numpy.min(data, axis=0))

fig.tight_layout()
```

Define a function:

- def plot_my_data(my_data):
    - fig = matplotlib.pyplot.figure(figsize = [10.0, 3.0])
    - axes1 = fig.add_subplot(1, 3, 1)
    - axes2 = fig.add_subplot(1, 3, 2)
    - axes3 = fig.add_subplot(1, 3, 2)
    - axes1.set_ylabel('average')
    - axes1.plot(numpy.mean(my_data, axis=0))
    - axes2.set_ylabel('max')
    - axes2.plot(numpy.max(my_data, axis=0))
    - axes3.set_ylabel('min')
    - axes3.plot(numpy.min(my_data, axis=0))
    - fig.tight_layout()

Use the function:

plot_my_data(data2)

Use wildcards in Python & for loop to make plots:

```
import glob
my_files = sorted(glob.glob("*.csv"))
for this_file in my_files:
    my_data = numpy.loadtxt(fname=this_file, delimiter=',')
    print(this_file)
    plot_my_data(my_data)
```

Moving Plots Around

```
#####Defining a function#######

def plot_my_data(my_data):

    #putting 3 plots together in one plot
    #makes an empty figure in matplot live with dimmensions 10 x 3
    fig = matplotlib.pyplot.figure(figsize = (4.0,10.0))
    matplotlib.pyplot.tight_layout()

    #make axes for your 3 plots
    axes1 = fig.add_subplot(3, 1, 1)
    axes2 = fig.add_subplot(3, 1, 2)
    axes3 = fig.add_subplot(3, 1, 3)

    #working with the axis itself not one plot
    axes1.set_ylabel('Average')
    axes1.plot(numpy.mean(my_data, axis=0))

    axes2.set_ylabel('Maximum')
    axes2.plot(numpy.max(my_data, axis=0))

    axes3.set_ylabel('Minimum')
    axes3.plot(numpy.min(my_data, axis=0))

    #keeps the figures together and no axes overlap
    fig.tight_layout()

plot_my_data(data)
```

Plot on same plot:

- def plot_my_data(my_data):
    - # plot 1
    - matplotlib.pyplot.plot(numpy.mean(my_data, axis=0), label='mean')
    - # plot 2
    - matplotlib.pyplot.plot(numpy.max(my_data, axis=0), label='max')
    - # plot 3
    - matplotlib.pyplot.plot(numpy.min(my_data, axis=0), label='min')
    - # add a legend
    - matplotlib.pyplot.legend()
- plot_my_data(data)

Full code for print_the_means.py

```
import numpy as np
import sys
```

```
def read_in_data(filename):
    my_data = np.loadtxt(filename, delimiter = ",")
    return my_data

def get_mean_and_print(data_array):
    my_means = np.mean(data_array, axis=0)
    for value in my_means:
        print(value)

def get_mean_and_print(data_array):
    my_means = np.mean(data_array, axis=0)
    for value in my_means:
        print(value)

def main():
    # pull in arguments from command line as list
    # 1st element of list is the name of the script
    # 2nd element on is the arguments that I am passing with script
    list_of_args = sys.argv

    # pulling the filename from the list of arguments
    my_filename = list_of_args[1]
    # reading in the data from the file into an array
    this_data = read_in_data(my_filename)
    # printing the average values for each column one at a time
    get_mean_and_print(this_data)
```

PRINT THE ARGS SCRIPT

```
import sys

if __name__ == '__main__':
    print(sys.argv)
```

Write a python script that takes two command line arguments. Combines then and prints the result

for example:

python my_script.py test1 test2

Should result in

test1test2
as output

Go through and comment print_the_means.py. Put any lines you didn't know what to write here in the etherpad:

if __name__ == "__main__": # if we run this program from the command line

- main() # run the function called "main"

plot the data script

```
import numpy
import matplotlib.pyplot
import sys

def plot_my_data(my_data):
    # plot 1
    matplotlib.pyplot.plot(numpy.mean(my_data, axis=0), label='mean')
    # plot 2
    matplotlib.pyplot.plot(numpy.max(my_data, axis=0), label='max')
    # plot 3
    matplotlib.pyplot.plot(numpy.min(my_data, axis=0), label='min')
    # add a legend
    matplotlib.pyplot.legend()

if __name__ == "__main__":

    infile = sys.argv[1]
    outfile = sys.argv[2]
    my_data = numpy.loadtxt(infile, delimiter=",")
    plot_my_data(my_data)
    matplotlib.pyplot.savefig(outfile)
```

---

^ edit this script to take an output file as well so you can specify the name of the output image

PLEASE FILL THIS OUT BEFORE YOU LEAVE
Post-Workshop Survey: https://www.surveymonkey.com/r/swc_post_workshop_v1?workshop_id=2019-05-23-shapiro
PLEASE LET US KNOW IF YOU ARE INTERESTED IN BEING AN INSTRUCTOR
Mailing List (for Future Workshops): https://mcommunity.umich.edu/#group:Software%20Carpentry%20Updates
Interested in Being a Helper or Instructor? https://forms.gle/Yk6KZ24incickBDf6