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 https://etherpad.wikimedia.org).

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

---

Welcome to Software Carpentry!

Links:

Workshop website: https://uw-madison-datascience.github.io/2022-01-10-uwmadison-swc/
Pre-workshop survey: https://carpentries.typeform.com/to/wi32rS?slug=2022-01-10-uwmadison-swc
Provide feedback on today's session: https://forms.gle/qYyJGCqSKEjd9FK46
Intro slides: https://docs.google.com/presentation/d/1n-Q_yFJ0EVt00COKg0mSLVu3OdUJlTA-a84d0rsgVW4/edit?usp=sharing
Wrap-up slides: https://docs.google.com/presentation/d/14vZaMG3gb45qHtkGZfd-QU3f_UvY0s-kC3GmflxU-xU/edit?usp=sharing
Get one-on-one help from the data science facilitators by attending our coding-meetups: https://datascience.wisc.edu/hub/#dropin
Data Science Research Bazaar throughout February: https://datascience.wisc.edu/data-science-research-bazaar/

Day 5 - Workflows with git/py/sh

Sign-in:

Name (pronouns optional) & affiliation. One question you have from the previous sessions.

- Sarah Stevens (she/her/hers), Data Science Hub
- Erwin Lares (he). Spanish and Portuguese
- (helper) Heather Shimon (she/her), Science & Engineering Libraries, how to set up my SSH key
- (helper) Tobin Magle (they/she), DoIT,
- (attendee) Kelsey Kruger (she/her), Soil Science, in Python can you see a list of all your assigned variables?

- (helper) Chris Endemann (he/him), Data Science Hub
- (attendee) Amanda N Price, Acquisitions and collections data librarian
- (attendee) Anupreksha Jain (she/her), MS Entomology. Where can I find practice exercises?
- (attendee) Olivia Bernauer (she/her), postdoc Entomology. If I want to code in python do I have to use jupyter lab/notebook? Or can I just use terminal to do everything we've been doing the last two weeks?
- (attendee) Owen Warmuth (he/him), UW Madison Biochemistry, How to keep track of all these commands? :)
- (attendee) Mengmeng Luo. Question: how can I retrieve my past notes in jupyter, or I just download them?
- (attendee) Amanda Siebert-Evenstone (she/they), Learning Analytics Masters Program. This process is a lot of typing, whereas, my previous work has used more scripts made by others (i.e., R). How do you think about your python code in relation to R code or others? When do you use Python vs. R? Is python always initiated via terminal?
- (attendee) Paige Stork, all my questions are covered above!
- (attendee) Harshit Kothari, Industrial Engineering - UW Madison, No questions. Thanks.
- (attendee) Azi Bashiri (she/her), Postdoc at UW-Madison Dept. of Medicine

Q From feedback:
- Most of the files we saw in the unix shell were txt , can terminal also work with other types of file eg, .csv and other?
- can jupyter save the notebook?
- Why did the jupyter notebook on Wed look different from the one we used last Friday?

Review Exercise:
What is one thing you learned in a previous section?
Fereshteh - grep
Harshit - pushing code to GitHub
Paige - I've been hearing a lot about version control and now I actually understand what Git version control actually means
Amanda - more confidence in coding / software-development
Mengmeng - lots of new material! The terminal is a useful tool
Owen - reading in data and plotting in Python
Olivia - Reading in data via pandas is very useful and quick/easy to do
Anupreksha - Manipulating files via terminal / Gitbash
Amanda - Realizing redundancy across programming languages. Once you learn one language, it's not so hard to pick up another one!
Kelsey - I’m new to Python, but use R a lot, so it was fun to see ggplot in Python and other similarities to R.

 # making sure everyone has ssh keys setup

- open terminal (mac) or gitbash (windows)
ssh -T git@github.com # if this tells you you've successfully authenticated, you're set for now
ssh-keygen -t ed25519 -C "yourEmailAddress" # tells github that your computer is associated with your github account
- enter passphrase and generate key
ls -al .ssh # should see a private key (id_ed25519) and public key (id_ed25519.pub)

pbcopy < ~/.ssh/id_ed25519.pub # or "clip < ~/.ssh/id_ed25519.pub" on windows

- on Github
- Go to profile icon and click settings
- Go to SSH and GPG keys section of account settings (tab on left panel)
- Click New SSH key
- give key title (name of your computer is usually best)
- paste key into key section (ctrl+v or right click and then click paste)
- click add key

- open terminal/gitbash again
ssh -T git@github.com
- type yes
- enter in your passphrase

 # Setup project on Github

- github.com
- Click the green "New" button to create new repository OR go to plus icon in top right and click "New repository"
- Give repo name: swc-gapminder
- leave as public repo
- include README file (this is standard for most repos. The README contains useful info on code authors, contact info, instructions on how to use the code, etc.)
- add .gitignore (Also pretty standard to have in most repos. You can specify file types that you want git to ignore in this file. We will leave this unchecked since github doesn't like that option checked for some reason)
- click create repository
- click green code button
- click SSH instead of HTTPS and copy the link to the github repo

- open terminal
git clone (paste SSH URL by right-clicking)
ls
cd swc-gapminder
git log # print off commit history
git remote -v #
ls
mkdir data # make data directory
git status
copy gapminder data files into this new data folder
cp ~/Desktop/python-novice-gapminder-data/data/gapminder_*.csv data # can use tab complete to confirm your files are where you think they are

 # Back from break

ls # should see README and data folder
mkdir figs # make another directory
ls

```
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

# load data and transpose so that country names are
# the columns and their gdp data becomes the rows
data = pandas.read_csv('data/gapminder_gdp_oceania.csv', index_col = 'country').T

# create a plot of the transposed data
ax = data.plot()

# display the plot
plt.show()
```

- open jupyter lab and click Text File
jupyter lab
- paste the above pink text into your untitled text file
- right click your file in the file navigator in jupyter lab and rename to gdp_plots.py # python scripts end in .py; after changing the filename you'll see the text in your file looks more like a python script (keywords highlighted as green, character strings are read, etc.)

- in terminal
ls # we have our gdp_plots.py file now
python gdp_plots.py # run this script via the terminal
git status
git add gdp_plots.py
git status
git add data
git status

- go back to jupyter lab and click the plus button (near top left)
- create new text file
- name it .gitignore (right click on file and rename) # you won't see the file in jupyter lab's file explorer since it begins with a period. The period indicates that the file/folder is hidden

- open file and enter the following two lines to ignore the following folders:
.ipynb_checkpoints/
figs/
- ctrl+s to save file (or file -> save)

- go back to terminal
git status
git add .gitignore
git status
git commit -m "First commit of analysis script"
python gdp_plots.py

- back to jupyter lab
# set some plot attributes
ax.set_xlabel("Year")
ax.set_ylabel("GDP Per Capital")
# set the x location and labels
x.set(x_ticks(range(len(data.index))
ax.set_xticklabels(data.index, rotatation = 45)

 # added labels and title to plot

import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

filename = 'data/gapminder_gdp_oceania.csv'

# load data and transpose so that country names are
# the columns and their gdp data becomes the rows
data = pandas.read_csv(filename, index_col = 'country').T

# create a plot of the transposed data
ax = data.plot(title = filename)

# set some plot attributes
ax.set_xlabel("Year")
ax.set_ylabel("GDP Per Capita")
# set the x location and labels
ax.set_xticks(range(len(data.index)))
ax.set_xticklabels(data.index, rotation = 45)

# display the plot
plt.show()

Let's test out our changes in terminal/gitbash
python gdp_plots.py # we have a title, rotated xtick labels, x label
git add gdp_plots.py
git commit -m "Improving plot format"

git status # shows that 2 commits have been made before syncing up with github

back in github, only have initial commit
sync github repository
git push origin main

can see changes in github
it takes the history with you when you push to github

```
# undo a git add - to undo add before committing
git reset filename

# change python code to take a file name as an input
# command line arguments

# make a tester script
# in jupyter, make a new script
# new text file, rename to argv_list.py

import sys

print('sys.argv is', sys.argv)

# in terminal/bash
python argv_list.py first second third

# update in jupyter
print('sys.argv is', sys.argv[2])

# back in gdp_plots
# add
import sys

# change line 7
filename = sys.argv[1]

# current version of the script in gpd_plots
import sys
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt
filename = sys.argv[1]

# load data and transpose so that country names are
# the columns and their gdp data becomes the rows
data = pandas.read_csv(filename, index_col = 'country').T

# create a plot of the transposed data
ax = data.plot(title = filename)

# set some plot attributes
ax.set_xlabel("Year")
ax.set_ylabel("GDP Per Capita")
# set the x locations and labels
ax.set_xticks(range(len(data.index)) )
ax.set_xticklabels(data.index, rotation = 45)
```

```
# display the plot
plt.show()

# in terminal
python gdp_plots.py

# add an argument
python gdp_plots.py data/gapminder_gdp_oceania.csv

python gdp_plots.py data/gapminder_gdp_asia.csv

# add and commit in one step
git commit -m "Adding command line arguments" gdp_plots.py

# branching
# want gdp plots to make multiple plots - once in python, once in bash
# keep the faster one

# create two different branches to work with - start with the same history as main branch

git branch python-multi
git branch bash-multi

git branch # see our new branches
git chechout python-multi

git branch # star is next to python-multi branch

# need a for loop
# in python
for filename in sys.argv[1:]: # take all of the list items starting at the first index
# need to indent the rest of the section

# new version of script

import sys
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

for filename in sys.argv[1:]:
    # load data and transpose so that country names are
    # the columns and their gdp data becomes the rows
    data = pandas.read_csv(filename, index_col = 'country').T

    # create a plot of the transposed data
    ax = data.plot(title = filename)
```

```
    # set some plot attributes
    ax.set_xlabel("Year")
    ax.set_ylabel("GDP Per Capita")
    # set the x locations and labels
    ax.set_xticks(range(len(data.index)))
    ax.set_xticklabels(data.index, rotation = 45)

    # display the plot
    plt.show()

# in bash
python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_dgp_africa.csv

# will  pop up graphs one at a time. Need to close Oceania before Africa will pop up

# add and commit
git add gdp_plots.py
git commit -m "allowing for plot generation for mulitple files at once"

# save plot to figs folder. In python change under #display the plot
plt.savefig('figs/gdp-plot.png')

# terminal - up arrow to get previous commands
python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_dgp_africa.csv

# in python change last comment
# save the plot with a unique file name
# original input name: data/gapminder_gdp_region.csv
# pull out only the parts of the name that we want to keep
# split function - keep only first part of name
split_name1 = filename.split('.')[0] # data/gapminder_gdp_region - no csv at the end of it

# don't want the data/ part as well
# split again
split_name2 = split.name1.split('/')[1] # gapminder_gdp_region

# save name
save_name = 'figs/' + split_name2 + '.png'

add to savefig call

import sys
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

for filename in sys.argv[1:]:
    # load data and transpose so that country names are
```

```
    # the columns and their gdp data becomes the rows
    data = pandas.read_csv(filename, index_col = 'country').T

    # create a plot of the transposed data
    ax = data.plot(title = filename)

    # set some plot attributes
    ax.set_xlabel("Year")
    ax.set_ylabel("GDP Per Capita")
    # set the x locations and labels
    ax.set_xticks(range(len(data.index)))
    ax.set_xticklabels(data.index, rotation = 45)

    # save the plot with a unique file name
    split_name1 = filename.split('.')[0] #data/gapminder_gdp_XXX
    split_name2 = split_name1.split('/')[1]
    save_name = 'figs/'+split_name2 + '.png'
    plt.savefig(save_name)

# in bash
python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_dgp_africa.csv

git status

# remove gd-plots
rm figs/gdp-plot.png

# make sure you are in the python-multi branch
git add gdp_plots.py
git commit -m "Saves each figure as a separate file"

git log --oneline --all --graph --decorate

# now make changes in bash
git checkout bash-multi

# in bash
touch gdp_plots.sh

# in python - gdp_plots.sh
for filename in data/gapminder_gdp_oceania.csv data/gapminder_gdp_africa.csv
do
```

- python gdp_plots.py $filename

```
done

# save file

# bash
```

```
bash gdp_plots.sh

# in python - gdp_plots.sh
# split names again and write file to the folder


import sys
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

filename = sys.argv[1]

# load data and transpose so that country names are
# the columns and their gdp data becomes the rows
data = pandas.read_csv(filename, index_col = 'country').T

# create a plot the transposed data
ax = data.plot(title = filename)

# set some plot attributes
ax.set_xlabel("Year")
ax.set_ylabel("GDP Per Capita")
# set the x locations and labels
ax.set_xticks(range(len(data.index)))
ax.set_xticklabels(data.index, rotation = 45)

# save the plot with a unique file name
split_name1 = filename.split('.')[0] #data/gapminder_gdp_XXX
split_name2 = split_name1.split('/')[1]
save_name = 'figs/'+split_name2 + '.png'
plt.savefig(save_name)

# in bash
ls figs
ls -l figs

git add gdp_plots.*
git commit -m "wrote bash script to call python plotter"

time bash gdp_plots.sh

# took around 2 sec

# test python
# go back to python branch

git checkout python-multi
```

time python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_gdp_africa.csv

git branch

take changes from python-multi to main

git checkout main # get back to main branch

bring changes to main branch
git merge python-multi

git log --oneline

More lessons here: https://carpentries-incubator.github.io/swc-ext-python/
ended during "Trying Different Methods", recommend "Running Scripts and Importing"

Day 4 - Python Part 2

Sign-in:

Name (pronouns optional) & affiliation. One thing you hope to use Python for in the future. If you don't know yet, that's okay! You'll hopefully have an answer after the end of today's workshop.

- (instructor) Chris Endemann (he/him), Data Science Hub
- (helper) Heather Shimon (she/her), Science & Engineering Libraries, survey analysis and plotting
- (helper) Sarah Stevens (she/her/hers), Data Science Hub I've been doing coding challenges in python. No plans for my work in python at the moment but I'd love to think of some projects.
- (helper) Erwin Lares (he) - no immediate plans to use python
- (helper) Trisha Adamus (she/her), Ebling Library
- (attendee) Owen Warmuth (he/him) UW Madison Biochemistry - Generate different visualizations of proteins given a sequence or structure
- (attendee) Amanda N Price - Acquisitions & Collection Data librarian - data analytics (gathering, collation, reporting in more automated ways) for purchasing, budgeting and collections decisions and management
- (attendee) Mengmeng Luo - Data analysis for my research
- (attendee) Kelsey Kruger (she/her) - Data analysis
- (attendee) Kate Slauson (she/her) - no plans to use Python at the moment other than maybe analyzing data related to diecast toy car jousting/crashing and creating pretty graphs/plots
- (attendee) Paige Stork - meta modeling a bulky hydrological model
- (attendee) Pema Lhamo (she/her),
- (attendee) Olivia Bernauer (she/her), statistics, models etc. with my own data

- (attendee) Nick Scheel (he/him), just trying to gain some literacy in coding, will probably usually use R
- (attendee) Anupreksha Jain (she/her), MS Entomology, data analysis and visualization
- (attendee) Amanda Siebert-Evenstone (she/they), Learning Analytics Masters Program

Notes:

Open Anaconda or Terminal

cd Desktop

Check that you have the pytong-novice-gapminder folder on your desktop

cd python-novice-gapminder-data

```
# open/launch jupyter notebook
jupyter notebook

# Plotting

import matplotlib.pyplot as plt

time = [0, 1, 2, 3]
position = [0, 100, 200, 300]

plt.plot(time, position)
plt.xlabel('Time (hr)')
plt.ylabel('Position (km)')

import pandas as pd

data = pd.read_csv('data/gapminder_gdp_oceania.csv', index_col='country')

data.head() # to check the data

years = data.columns.str.strip('gdpPercap_')

print(years)

data.columns = years.astype(int)

data.head()

data.loc['Australia'].plot()

# transpose data
```

```
data.T.plot()
plt.ylabel('GDP per capita')

# using ggplot style
plt.style.use('ggplot')
data.T.plot(kind='bar')
plt.ylabel('GDP per capita')

years = data.columns
gdp_australia = data.loc['Australia']

plt.plot(years, gdp_australia, 'g--') #g to plot a green line, -- for line type or -.

# plot multiple datasets together

gdp_australia = data.loc['Australia']
gdp_nz = data.loc['New Zealand']

# plot with different color lines/line types
plt.plot(years, gdp_australia, 'b--', label='Australia')
plt.plot(years, gdp_nz, 'g', label='New Zealand')

# create legend
plt.legend(loc='upper left')

#xlabels/ylabels
plt.xlabel('Year')
plt.ylabel('GDP per capita ($)')

# scatter plots - look at correlation between to countries' GDP
plt.scatter(gdp_australia, gdp_nz)

# transpose data
data.T.plot.scatter(x = 'Australia', y = 'New Zealand')

# Challenge
```

Fill in the blanks below to plot the minimum GDP per capita over time for all the countries in Europe. Modify it again to plot the maximum GDP per capita over time for Europe.

```
data_europe = pd.read_csv('data/gapminder_gdp_europe.csv', index_col='country')
data_europe.min().plot(label='min')
data_europe.max().plot(label='max')
plt.legend(loc='best')
plt.xticks(rotation=90)

# Challenge
```

Modify the example in the notes to create a scatter plot showing the relationship between the minimum and maximum GDP per capita among the countries in Asia (gapminder_gdp_asia.csv) for each year in the

data set. What relationship do you see (if any)?

```
data_asia = pd.read_csv('data/gapminder_gdp_asia.csv', index_col='country')

data_asia.describe().T.plot(kind='scatter', x='min', y='max')

# data does not look correlated

# saving figure
plt.savefig('myfigure.png')

# close notebook and start a new one
# New > Python 3
# call it python-lists

# create list [ ]
pressures = [.273, .275, .277, .275, .276]
print('pressures:', pressures)
print('length:', len(pressures))

# how to acces single elements of a list (list indexing)
print('first item of list', pressures[0])
print('fifth item of list', pressures[4])

# replace items in a list
pressures[0] = .276
print('pressures is now:', pressures)

# add items to a list with .append
primes = [2, 3, 5]
print('primes is initially', primes)
primes.append(7)
print('primes has become', primes)

# add two lists together with .extend
teen_primes = [11, 13, 17, 19]
middle_aged_primes = [37, 41, 43, 47]
print('primes is currently:', primes)
primes.extend(teen_primes)
print('primes has beome', primes)
primes.extend(middle_aged_primes)
print('primes has finally become', primes)

# to delete an item from a list with del
primes = [2, 3, 5, 7, 9]
print('primes before removing last item:', primes)
del primes[4]
print('primes after removing last item:', primes)
```

```
# initialize empty list
myList = []
print(myList)
myList.append(1)
print(myList)

# lists can contant multiple variable types
myList = [1, 'two', 3, 'four']

# character strings can be indexed lists
element = 'carbon'
print('0th character:', element[0])
print('3rd character:', element[3])

# to replace items in string use .replace

str.replace(element, 'c', 'C')

# output: 'Carbon'

# can't index beyond the length of a list
print('99th element of element list is:', element[99])

# Challenge
# Fill in the blanks so that the program below produces the output shown.
values = []
values.append(1)
values.append(3)
values.append(5)
print('first time:', values)
values = values[1:]
print('second time:', values)

# output:
# first time: [1, 3, 5]
# second time: [3, 5]

myList=[1, 2, 3, 4, 5]
myList=myList[0:2] # [beginIndex; endIndex], endIndex is exclusive
print(myList)
# output: [1, 2]

# indes the last item of a list
element = 'helium'
print(element[-1])
print(element[-2])

# step through certain values of a list
element = 'fluorine'
```

```
startIndex=0
endIndex=len(element)
print(element[startIndex:endIndex:2])
print(element[startIndex:endIndex:3])

# Challenge
# what values whould you use for start/end/stride to print off
# all even indices
element[startIndex:endIndex:stride]

# Solution
startIndex=1
endIndex=len(element)
stride=2
print(element[startIndex:endIndex:stride])

# copy a list (or not)
old = list('gold')
print(old)
new = old
new[0] = 'D'
print('new is', new, 'and old is', old)

# correct way to create a copy of a list
old = list('gold')
new = old[:]
new[0] = 'D'
print('new is ', new, 'and old is', old)

# exit out of notebook
# open a new notebook from home page, click New > Python 3
# label it python-for-loops

# for loops - need tab in format
for number in [2, 3, 5]: #hit enter to get tab in next line
    print(number)

# using for loop is same as
print(2)
print(3)
print(5)

primes = [2, 3, 5]
for p in primes:
    squared = p ** 2
    cubed = p **3
    print(p, squared, cubed)

# output
```

```
2 4 8
3 9 27
5 25 125

# range
for number in range(0,3):
    print(number)

# default starting index is zero, so this will also work
for number in range(0,3):
    print(number)

# use for loop to accumulate info

# sum over the first 10 integers
total = 0
for number in range(10):
    total = total + (number + 1)
print(total)

# Challenge
# Fill in the blanks in the program below so that it
# prints “nit” (the reverse of the original character string “tin”).

original = "tin"
result = ""
for char in original:
    result = char + result
print(result)

# Challenge
# Fill in the blanks in each of the programs below to
# produce the indicated result (12)
total = 0
for word in ["red", "green", "blue"]:
    total = total + len(word)
print(total)

# Challenge
# List of word lengths: ["red", "green", "blue"] => [3, 5, 4]
lengths = []
for word in ["red", "green", "blue"]:
    lengths.append(len(word)
print(lengths)

# Challenge
# Concatenate all words: ["red", "green", "blue"] => "redgreenblue"
words = ["red", "green", "blue"]
result = ""
```

```
for word in words:
    result = result + word
print(result)



# close notebook
# open a new file: New > Python 3
# call it python-conditionals

# Conditionals - if, else, elif
mass = 3.54
if mass > 3.0:
```

- print(mass, 'is large')

```
mass = 2.9
if mass > 3.0:
```

- print(mass, 'is large') # this will not execute because mass is less than 3.0

```
masses = [3.54, 2.07, 9.22, 1.86, 1.71]
for m in masses:
    if m > 3.0:
        print(m, 'is large')

# more than one condition to check with else
masses = [3.54, 2.07, 9.22, 1.86, 1.71]
for m in masses:
    if m > 3.0:
        print(m, 'is large')
    else:
```

- print(m, 'is small')

```
# three or more conditionals
masses = [3.54, 2.07, 9.22, 1.86, 1.71]
for m in masses:
```

- if m > 9.0:

```
        print(m, 'is HUGE')
    elif m > 3.0:
        print(m, 'is large')
    else:
        print(m, 'is small')

# conditions are tested once, and in order
grade = 85
if grade >= 70:
    print('grade is C')
elif grade >= 80:
    print('grade is B')
```

```
elif grade >= 90:
    print('grade is A')

# output: grade is C # due to order, once one is true, the others will not execute

# to do in correct order
if grade >= 90:
    print('grade is A')
elif grade >= 80:
    print('grade is B')
elif grade >= 70:
    print('grade is C')

# using conditionals to adjust/evolve values of variables
velocity = 10.0
for i in range(5): # execute the loop 5 times
    print(i, ':', velocity)
    if velocity > 20.0:
        print('moving too fast')
        velocity = velocity - 5.0
    else:
        print('moving too slow')
        velocity = velocity + 10.0

print('final velocity:', velocity)

# output:
0 : 10.0
moving too slow
1 : 20.0
moving too slow
2 : 30.0
moving too fast
3 : 25.0
moving too fast
4 : 20.0
moving too slow
final velocity: 30.0

# how to check multiple conditions in one if statement
mass = [3.54, 2.07, 9.22, 1.86, 1.71]
velocity = [10.0, 20.0, 30.0, 25.0, 20.0]

for i in range(5):
    if mass[i] > 5 and velocity[i] > 20:
        print("Fast heavy object.  Duck!")
    if mass[i] > 5 or velocity[i] > 20:
        print("Object is either fast or heavy or both")
```

```
# exit out of notebook and open a new Python 3 file
# call it python-looping-over-data

# for loop to read multiple files at once
import pandas as pd
files =['data/gapminder_gdp_africa.csv', 'data/gapminder_gdp_asia.csv']
for filename in files:
    data = pd.read_csv(filename, index_col='country')
    print(filename, '\n', data.min())


# glob funtion
import glob
print('all csv files in data directory:', glob.glob('data/*.csv'))

# could also use for text files
print('all csv files in data directory:', glob.glob('data/*.txt'))

# read in multiple files at once
allCSVfiles = glob.glob('data/*.csv')
for filename in allCSVfiles:
    data = pd.read_csv(filename)
    print(filename, '\n', data['gdpPercap_1952'].min())


# close file, start new Python 3 file
# call it python-functions

# Functions
# syntax to create a function: def [name of function](inputs):
def print_greeting():
    print('Hello!')


print_greeting()

def print_date(year, month, day):
    joined = str(year) + '/' + str(month) + '/' + str(day)
    print(joined)

print_date(1995, 3 , 17)
# output: 1995/3/17

# order matters
print_date(3 , 1995, 17)

# output: 3/1995/17

# can use labels to get ther correct order
print_date(month=3, year=1995, day=17)
```

```
# functions may return results to their caller
def average(values):
    if len(values)==0:
        return None
    return sum(values) / len(values)

a = average([1, 3, 4])
print('avg of actual values:', a)

print('avg of actual values:', average([]))
```

```
def another_function():
    • print("Syntax errors are annoying.")
    • print("But at least python tells us about them!")
    • print("So they are usually not too hard to fix.")
```

```
# Challenge
# What does the following program print?
def report(pressure):
    print('pressure is', pressure)

print('calling', report, 22.5)

# Output: calling <function report at 0x000001F9490AB378> 22.5
# not calling function correctly. to fix...

# print('calling', report, 22.5)
# print('calling\n', report(22.5))
report(22.5)

# when you don't have output explicitly returned, function returns None

# Challenge
# Fill in the blanks to create a function that takes a single
# filename as an argument, loads the data in the file named by
# the argument, and returns the minimum value in that data.

import pandas as pd
def min_in_data(filename):
    data = pd.read_csv(filename)
    return data.min()

# Challenge
# Fill in the blanks to create a function that takes a list of
# numbers as an argument and returns the first negative value
# in the list. What does your function do if the list is empty?
# What if the list has no negative numbers?
```

```
def first_negative(values):
    for v in values:
        if v < 0:
            return v

print(first_negative([])) # returns None
```

Day 3 - Python Part 1

Sign-in:

Name (pronouns optional) & affiliation.

- (helper) Chris Endemann (he/him), Data Science Hub
- (helper) Heather Shimon (she/her). Science & Engineering Libraries
- (instructor) Trisha Adamus (she/her), Ebling Library
- (helper) Erwin Lares (he) - Spanish and Portuguese
- (attendee) Olivia Bernauer (she/her), postdoc in Entomology
- (attendee) Amanda Siebert-Evenstone (she/they), Creative Producer, Learning Analytics Masters Program
- (attendee) Kelsey Kruger (she/her), Research Specialist, Soil Science
- (attendee) Amanda N Price, Acquisitions and data librarian
- (attendee) Pema Lhamo (she/her), Educational psychology
- (attendee) Azi Bashiri (she/her), postdoc in Department of Medicine
- (attendee) Mengmeng (she/her), Soil Science
- (attendee) Nick (he/him) PhD student Freshwater and Marine Sciences
- (attendee) Anupreksha (she/her) MS Entomoloogy
- (helper) Sailendharan Sudakaran (he/him), Microbiome Hub
- (attendee) Kate Slauson (she/her), graduate student in the UW-Madison iSchool
- (attendee) Eliceo Ruiz Guzman (he/him) Graduate student in Environmental Observation and Informatics - UW-Madison

Notes:

```
# Open Anaconda
# Windows - use Anaconda Prompt
#mac - open Terminal
$ jupyter lab

# open Notebook - Python 3 - will have ipynb extension
```

```
# Shift enter to run a cell and insert below

# Help tab is available

# Math in python

# Challenge
# 1. Type 7*3 into a cell and then shift + return.
# 2. Type 2+1 into a cell and then shift + return.
# 3. Type 7*3 into a cell then type 2+1 into the same cell and shift return. What output do you get and why?

# only processing last line of code: 2 + 1 = 3

# Variables - store values
# use = to assign value to a variable

# Comments (notes that python does not run) - use # to comment in code

# use meaningful names when creating variables - but not so long for retyping
# do not use spaces in variable names. Do not start with an underscore or a number.
# Variable names are case sensitive

age = 42
first_name = 'Arthur'

# shift + enter to run

# Functions - built in code in python to do certain things - will show up in green font

# using print fuction - calls your code so you can see the output
print(first_name, 'is', age, 'years old')

# output: Arthur is 42 years old

# order matters!

print(last_name) #gives error, need to give a value to last_name

# Error messages are helpful in python

first_name = 'Trisha' # reassigns value of first name to Trisha

print(first_name, 'is', age, 'years old')

# output: Trisha is 42 years old

# if re-run line 5 of code, it becomes line 10
```

```
# returning to line 9 of code, and running it again, first_name is reassigned to Arthur

# ctrl + enter to run line of code without creating a new line

# check number next to line of code to see if it has been re-run

# variables and calculations

age = age + 3 #reassignes the variable age to 42 + 3

print('Age in three years:', age)

# output: Age in three years: 45

# indexing with [ ] - ask for a position in the string
# when indexing, first value is at zero (0)

atom_name = 'helium'
print(atom_name[0])

# output: h

# slicing - ask for multiple positions within the string
# [start:stop] - stops before the position of the last value, and does not inlcude it
atom_name = 'helium'
print(atom_name[0:3])

# output: hel

# Len function
print(len('helium')) # same as: print(len(atom_name))

# output: 6

# Challenge
# What is the final value of position in the program below? (Try to predict the value without running the program, then check your prediction.)
initial = 'left'
position = initial
initial = 'right'

# solution: left, position was set to 'left', and did not change when initial was reassigned to 'right'

# Challenge
# If you assign a = 123, what happens if you try to get the second digit of a via a[1]?
a = 123
print(a[1])

# get an error message, 123  is a number and not a string, would need to convert the number to a string to
```

be able to index it

# Challenge
# Which is a better variable name, m, min, or minutes? Why? Hint: think about which code you would rather inherit from someone who is leaving the lab:
ts = m * 60 + s
tot_sec = min * 60 + sec
total_seconds = minutes * 60 + seconds

minutes makes the best variable. min could mean minimum and is a reserved keyword in python

data types and type conversion

Can use + in the tool menu to add a cell

Data Types and Type Conversion # make this a title by converting it to Markdown, can change it in the tool menu - change Code to Markdown

add ### to make it a header (text will change to blue)

number of hashtags changes the size of the font

can also insert cells using the + in the tool menu

add heading +
Variables and Assignments
esc + m will also change text to Markdown

every value has a type #esc + m to make Markdown

# integer
print(type(52))
# output <class 'int'>

# float
print(type(3.14))
# output <class 'float'>

# string
fitness = 'average'
print(type(fitness))
# output <class 'str'>

print(5-3)
# output: 2

print('hello' - 'h')
# error message: can't do this operation with a string

```
full_name = 'Arthur' + ' ' + 'Dent'
print(full_name)

dog = 'Bark' * 10
print(dog)
# output: BarkBarkBarkBarkBarkBarkBarkBarkBarkBark

# strings have a length but numbers don't #esc + m to change to Markdown

print(len(full_name))
#output: 11

print(len(52))
#output is an error. Can't use length function on an integer

print(1 + '2')
# output is an error. Can't add int and str together

# Converting strings and integers
print(1 + int('2'))
#output: 3

print(str(1) + '2')
#output: 12 - concatenation, brings 1 and 2 together

# mixing integers and floats is okay
# converts integers to floats automatically
print(1 / 2.0)
# output: 0.5

print(3.0 ** 2) # ** will square 3.0
# output: 9.0

#Challenge

# What type of value is 3.4? How can you find out?
print(type(3.4))
#output: <class 'float'>

# Division types
print('5 // 3:', 5 // 3)
# output: 5 // 3: 1

print('5 / 3:', 5 / 3) #floating point division
# output: 5 / 3: 1.6666666666666667

print('5 % 3:', 5 % 3)
# output: 5 % 3: 2
```

```
# strings to numbers
print('string of numbers to float:', float('3.4'))
# output: string of numbers to float: 3.4

print('float to int:', int(3.4))
# output: float to int: 3

print('string of letters to float:', float('Hello world'))
# output error message - this is not possible to convert a string of letters to a float

# every function returns something

result = print('example')
print('result of print is', result)
# output:
example
result of print is None # printed something - None

# Commonly used built-in functions: min, max, round

# min and max functions
print(max(1, 2, 3))
# output: 3

print(min('a', 'A', '0'))
# output: 0 # orders 0-9, A-Z, a-z

print(min('a', 'A'))
# output: A

print(max('a', 'A'))
#output: a

# round function
round(3.712)
#output: 4

# add arguments
round(3.712, 1) # to round to first decimal place
# output:  3.7

# help function
help(round)

round(371.2, -1)
#output 370.0

# Challenge
# Predict what each of the print statements in the program below will print.
```

```
# Does max(len(rich), poor) run or produce an error message? If it runs, does its result make any sense?
easy_string = "abc"
print(max(easy_string))
rich = "gold"
poor = "tin"
print(max(rich, poor))

# output:
c
tin # t comes after g in alpha order



## Libraries
calling libraries module_name.thing_name

import math # import math library

print('pi is', math.pi)
print('cos(pi) is', math.cos(math.pi))

help(math) # gives you info on math library

# import only specific items from library
from math import cos, pi
print('cos(pi) is', cos(pi))

# creating aliases
import math as m
print('cos(pi) is)', m.cos(m.pi))

# Fill in the blanks so that the program below prints 90.0.
# import math as m
# angle = m.degrees(m.pi / 2)
# print(angle)

# Fill in the blanks so that the program below prints 90.0.
# from math import pi, degrees
# angle = degrees(pi / 2)
# print(angle)

# download data and place on desktop
http://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip

# start a new notebook

## Reading Tabular Data into Dataframes
import pandas as pd

data = pd.read_csv('data/gapminder_gdp_oceania.csv')
```

```
print(data)

data = pd.read_csv('data/gapminder_gdp_oceania.csv', index_col = 'country')

data.info()

print(data.columns)

# DataFrame.T to transpose a dataframe
print(data.T)

# Read the data in gapminder_gdp_americas.csv into a variable called americas and display its summary statistics.
americas = pd.read_csv('data/gapminder_gdp_americas.csv', index_col='country')
americas.describe()

# head command for first rows
americas.head(n=3)

# tail command - for last rows
america.tail(n=3)

# data to csv
americas.to_csv('processed.csv')

# iloc. I like iloc
import pandas as pd
data = pd.read_csv('data/gapminder_gdp_europe.csv', index_col='country')

print(data.iloc[0,0]) # print first row, first col

data.head(n=1)

# loc by row and column names
print(data.loc['Albania', 'gdpPercap_1952'])

print(data.loc['Albania', :]) # all columns for albania

print(data.loc[:, 'gdpPercap_1952'])

# name slicing
print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'])

# functions applied to slices
print(data.loc['Italy' : 'Poland', 'gdpPercap_1962' : 'gdpPercap_1972'].max())

# Use a subset of data to keep output readable
subset = data.loc['Italy' : 'Poland', 'gdpPercap_1962' : 'gdpPercap_1972']
```

```
print('Subset of data:\n', subset) # \n indicates that you want to insert a new line

# which values are greater than 10,000?
print('\nWhere are values large?\n', subset > 10000)

# Boolean mask
mask = subset > 10000
print(subset[mask])

print(subset[mask].describe())
```

Reminders before you take off for the day!

1) Please fill out this brief feedback form on today's session: https://forms.gle/5fovmeA54cWTjUJm9

2) The Data Science Hub is here to support you! You can join our coding-meetups Tuesdays/Thursdays from 2:30-4:30pm to get help from one of the data science facilitators. Info here on how to join for coding meetup sessions: https://datascience.wisc.edu/hub/#dropin

Day 2 - Version Control With Git/GitHub

Sign-in:

Name (pronouns optional) & affiliation.

- (helper) Chris Endemann (he/him), Data Science Hub
- (instructor) Casey Schacher (she/her), Science & Engineering Libraries
- (helper) Erwin Lares (he), Research Cyberinfrastructure Group
- (attendee) Kate Slauson (she/her), graduate student in the UW-Madison iSchool
- (helper) Trisha Adamus (she/her), Ebling Library
- (attendee) Amanda Siebert-Evenstone (she/they) Creative producer for the Learning Analytics Masters Program in the Dept of Educational Psychology

- (attendee) Harshit Kothari (he/him), graduate student in the department of Industrial Engineering.
- (attendee) Eliceo Ruiz Guzman (he/him), Graduate Student - Environmental Observation and Informatics-UW Madison
- (attendee) Kelsey Kruger (she/her), Research Specialist - Soil Science
- (attendee) Owen Warmuth (he/him) UW Madison Biochemistry
- (Attendee) Paige Stork, UW Madison Freshwater and Marine Sciences
- (attendee) Pema Lhamo (she/her), Educational psychology
- (attendee) Shaoxuan Wang (she/her). I am a master's student in UW-Madison information School.
- (attendee) Nick Scheel (he/him). PhD student- Freshwater and Marine Sciences UW Madison
- (helper) Heather Shimon (she/her), Science & Engineering Libraries
- (attendee) Anupreksha Jain (she/her), MS Entomology student
- (attendee) Amanda N Price
- (attendee) Olivia Bernauer (she/her), postdoc Entomology
- (attendee) Azi Bashiri (she/her), postdoc UW Madison

Notes:

Open GitBash (Windows) and Terminal (Mac)

to copy and paste - right click (ctrl/cmd V or C will not work)

Set git to your information - examples - only have to set once
```
$ git config --global user.name "Vlad Dracula" #use your name
$ git config --global user.email "vlad@tran.sylvan.ia" #use your email
$ git config --global color.ui "auto"
```

And on Windows:
```
$ git config --global core.autocrlf true
```

on Mac/Linux:
```
$ git config --global core.autocrlf input
```

# Configuring for text editor (nano)
$ git config --global core.editor "nano -w"

Help
```
$ git config -h
$ git config --help
```

# For a list of all of the configurations that you've already set
$ git config --list
(on a mac, might need to enter through some other commands to get list)

Creating a Repository - builds history of your project over time
find where you are first - pwd

create new project folder

$ mkdir planets

# change directory - cd
$ cd planets

# Initialize the repository
$ git init

# to see the directory
$ ls -a

# check that everything is set up correctly - changes that haven't been committed will show up with this command
$ git status

don't nest repositories inside of repositories

# create a file called mars.txt
$ nano mars.txt

Enter text:
Cold and dry, but everything is my favorite color

ctrl + x to save > Y > hit enter to save

# print the contents of the file
$ cat mars.txt

# Check for untracked changes
$ git status

# Tracking a change takes two steps: 1. add the change to a staging area; 2. commit the change
# step 1: adding to a staging area
$ git add mars.txt

# check status again
$ git status

# step 2: commit the change with a message using -m
$ git commit -m "Start notes on Mars"
# keep messages short but helpful for remembering history

# can check status again
$ git status

# See project history
$ git log

Open nano again

$ nano mars.txt

add text [use arrows to navigate around lines]
The two moons maybe a problem for Wolfman.

ctrl + x to save > Y > hit enter to save

# check status to see changes
$ git status

# To see difference of uncommitted state and current state - what has not committed will come up in green
$ git diff

1. The first line tells us that Git is producing output similar to the Unix diff command comparing the old and new versions of the file.
2. The second line tells exactly which versions of the file Git is comparing; df0654a and 315bf3a are unique computer-generated labels for those versions.
3. The third and fourth lines once again show the name of the file being changed.
4. The remaining lines are the most interesting, they show us the actual differences and the lines on which they occur. In particular, the + marker in the first column shows where we added a line.

# Commit in one step - add file to the end of the commit command
$ git commit -m "Adding information about Wolfman." mars.txt

Challenge
Add the following sentence to mars.txt - "But the Mummy will appreciate the lack of humidity."
And commit it with the following message: "Mars climate for Mummy"

# open nano with
$ nano mars.txt

add text: "But the Mummy will appreciate the lack of humidity."
ctrl + x to save > Y > hit enter

cat mars.txt # to check if the changes were saved

git commit -m "Discuss concerns about Mars' climate for Mummy" mars.txt

git status # should show "clean working tree"

git log # check the changes we've committed
now have 3 commits to this repository

how can we see which parts of a line are changed instead of the whole lines?

nano mars.txt
Add text "awesome" to second line between "two" and "moons"
Also added "red" to end of the first sentance

ctrl + x to save > Y > hit enter to save

git diff # what is the difference between the current version of the file and the on last committed to the repo
shows highlighted first and 2nd lines
modify the git diff command to hightlight the word differences
git diff --color-words mars.txt
now we see the additions of the words we added

commit those changes
git commit -m "Added random words." mars.txt

you don't need to do this
mkdir moons # made a moon directory
git status # still working tree clean - git tracks files not folders

git log
we have 4 commits now, starting to get a bit bigger

see the most recent commit
git log -1

git log --oneline
see a shortened summary of each commit

Challenge
Which command(s) below would save the changes of myfile.txt to my lcoal git repository?
1) git commit -m "my recent changes"
2) git init myfile.txt ....AND THEN... git commit -m "my recent changes"
3) git add myfile.txt ....AND THEN... git commit -m "my recent changes"
4) git commit -m myfile.txt "my recent changes"

1) didn't add the file to staging area
2) confused git init with git add
3) this works!
4) the filename is in the wrong place, between the -m flag and the comment

nano mars.txt
add new line to the bottom of the file: An ill-considered change
ctrl + x to save > Y > hit enter to save

cat mars.txt # see the current version of the file

how do we get rid of this ill-considered change?

compare the difference between our most recent version and the most recent commit
git diff HEAD mars.txt

we can compare older versions in the repository

```
git diff HEAD~1 mars.txt # looks at the difference between most recent saved copy and the 2nd to last commit
git diff HEAD~1 mars.txt # looks at the difference between most recent saved copy and the 3rd to last commit

# see the differences introduced from a specific commit
git show HEAD~2 mars.txt

# using HEAD~X can get long (if you are looking 100+ commits back for example)
# instead you can use the unique identifer

# look at previous commits
git log --oneline
# copy unique identifier for the commit you are interested in

git diff <PASTE/TYPE UNIQUE ID HERE> mars.txt
# shows the difference betweent the current version and the commit specified

cat mars.txt
# see the most version of mars.txt

# we want to get rid of our ill-considered change
git checkout HEAD mars.txt
# this makes the current version of mars.txt the last committed version
# removes the ill-considered change line (which we never committed to the repository)

cat mars.txt
# no longer has the ill considered change

# could use this same checkout command to with HEAD~X or the unique ID to checkout older versions of the file
git checkout HEAD~3 mars.txt # revert to the 4th most recent commit

cat mars.txt
# show that we are back that the first commit we made (one line in mars.txt)

git status # shows that mars.txt was modified (and added to the stage!!)
# haven't commited the revert back to previous version to repository.

git checkout HEAD mars.txt # get back most recent commit

cat mars.txt
# back to the full 3 lines

git checkout HEAD~1 mars.txt

git status # see that something is different
# our most recent version file is different from the last committed version
```

```
git checkout HEAD mars.txt

# if you forget the file name when checking out an old commit ( you can get a "detached head" state)
git checkout HEAD~1 # will get a detached head
# this is sort of confusing for git as it doesn't know how to move forward
# often times this happens because you forgot to put the filename so it isn't quite what you wanted
# to fix it
git checkout master # if you check out the branch name, it will fix it and you can then re-run your checkout



# Branches!!!
# making changes in parallel, keeping changes separate
# reduces chance of messing up your main copy by accident
# default branch is called "master" (much of the git community has moved to calling this branch "main" instead)
# can change things in the experimental branches without affecting the main/master branch
# can then test and merge in changes from the experimental branches into the main/master branch

git branch
# shows the branches that you have already with a * next to the branch you are currently on

git branch pythondev # creates a new branch (from the current version of the master branch) called pythondev
# still on the master branch though

git checkout pythondev
# this is a second use of the checkout command, instead of checkout an old version of a file it moves us to the new branch

git branch
# see that we are on the pythondev branch now (note the * next to the pythondev branch)

touch analysis.py # creates an empty file called analysis.py

git add mars.txt
git commit -m "Wrote and tested python analysis script"
# can't use the single git commit line because this file has not been previously added to the repo

git status # everything is commited / working tree clean

# switch back to master branch
git checkout master

ls # see that no analysis.py exists in the master branch

git checkout -b bashdev # creates and switches over to the branch in one step

touch analysis.sh # makes empty bash script
```

note there is no analysis.py because we created the bashdev branch from the master which didn't have the python script

add/commit new file to bashdev branch
git add analysis.sh
git commit -m "Wrote and tested bash analysis script"

git checkout master
git branch
ls

merge files from pythondev into master branch
git merge pythondev

ls

git branch -d pythondev
git branch
git branch -d bashdev # error since bashdev has changes that aren't in master

git branch -D bashdev # forces delete

git branch # branch was successfully deleted

cat mars.txt

nano mars.txt

add line of text, e.g.,
"I'll be able to get 40 extra minutes of beauty rest."

Ctrl+x, then y to return to command line

cat mars.txt # view file contents

git commit -m "Add a line about the daylight on Mars." mars.txt

git log --oneline # remind yourself of past commits

create a new branch
git branch marsTemp
git branch
cat mars.txt

nano mars.txt
"This is a new line that will not be in the experimental branch."
Ctrl+x, then y to return to command line

git commit -m "adding a new line" mars.txt

git checkout marsTemp # move to this branch

git branch
cat mars.txt

nano mars.txt
"Yeti will appreciate the cold."
Ctrl+x, then y to return to command line

git commit -m "Add a line about temperature on Mars." mars.txt

master branch and marsTemp branch both have different new lines. Let's create a conflict
git checkout master
git branch
git merge marsTemp # conflict, merge failed

cat mars.txt # view conflict

nano mars.txt
delete Git messages indicating conflict and just keep both of the new sentences
Ctrl+x, then y, then enter

now that we've fixed the conflict, we can commit the file
git commit -m "Merged changes from MarsTemp." mars.txt

git add mars.txt
git commit -m "Merged changes from MarsTemp."
git status

....
Remotes in GitHub: https://swcarpentry.github.io/git-novice/07-github/index.html

- copy ssh address from GitHub repo
git remote add origin git@github.com:yourUserName/planets.git
git remote -v

ls -al ~/.ssh
ssh-keygen -t yourUniqueIDcode -C "yourGitHubEmail" # type your password after the email
ls -al ~/.ssh

cat ~/.ssh/id_ed25519.pub
copy the output: right click and select all, then select just the output line

- go to GitHub and right click on your profile icon. Click settings
- find the SSH and GPG keys section in the settings
- New SSH key

- Git it a title that indicates the name of the machine you are on
- paste key into the Key section
- Add SSH key

Reminders

1) If you had any problems verifying your python setup, please come to Friday's workshop at 8:30 a.m. so we can help you troubleshoot. To verify everything is setup for Friday's session, open up Terminal (Mac/Linux) or GitBash (Windows) and enter the following commands:

python --version # will tell you the version number. You should have version 3.7.X or higher (where X can vary)

python -c “import pandas” # this line should not produce any output. If you get an error, please come to Friday's session at 8:30am.

2) Please fill out this brief feedback form on today's session: https://forms.gle/5fovmeA54cWTjUJm9

3) The Data Science Hub is here to support you! You can join our coding-meetups Tuesdays/Thursdays from 2:30-4:30pm to get help from one of the data science facilitators. Info here on how to join for coding meetup sessions: https://datascience.wisc.edu/hub/#dropin

Day 1 - Unix Shell

Sign-in:

Name (pronouns optional) & affiliation. Please describe your research in a couple of sentences.

- (instructor) Dorothea Salo (she/her), UW-Madison Information School. I am currently studying undergraduate perceptions of privacy vis-a-vis learning analytics in the Data Doubles project (https://datadoubles.org/).
- (helper) Tobin Magle (they/she) DoIT research cyberinfrastructure group, I help researchers use cyberinfrastructure provided by campus
- (helper) Chris Endemann (he/him), Data Science Hub. I help researchers apply data science and machine learning tools to their research projects.
- (helper) Clare Michaud (she/her), Data Science Hub,
- (helper) Erwin Lares (he) Spanish and Portuguese. I don't know what's going on.
- (attendee) Amanda Siebert-Evenstone (she/they); I'm a creative producer in the Learning

Analytics Masters Program so I teach intro to qual and quant research and support the creation of dynamic lecture videos. I study how to measure and model complex thinking.
- (helper) Trisha Adamus (she/her), Ebling Library
- (attendee) Kate Slauson (she/her), UW-Madison Information School. I am a masters student in the iSchool and work as an archivist for the Center for Limnology.
- (attendee) Paige Stork (she/her), Freshwater & Marine Sciences Masters student, studying flooding and flood resilience in Wisconsin's Driftless Area
- (attendee) Mengmeng Luo (she/her), Department of Soil Science. I am a graduate student researching the impact of repeated fires on PyOM in the boreal forest system.
- (attendee) Harshit Kothari (he/him), Department of Industrial Engineering. I am a graduate student working on stochastic optimization. Most of my work is done in python.
- (attendee) Eliceo Ruiz Guzman (he/him) UW-Madison- Environmental Observation and Informatics
- (attendee) Olivia Bernauer (she/her) UW-Madison Department of Entomology. I am a postdoc researcher studying how elevated CO2 may impact the nutritional value of pollen and nectar for pollinators like bumble bees.
- (attendee) Pema Lhamo(she/her), Educational Psychology
- (attendee) Owen Warmuth (he/him), UW-Madison Biochemistry. I am a Ph.D. student developing high-throughput methods for processing NMR and cryo-EM data.
- (attendee) Kelsey Kruger (she/her), Soil Science.
- (attendee) Azi Bashiri (she/her), Department of Medicine
- (attendee) Amanda N Price, Texas State University Libraries, Acquisitions. Position changing to be data analytics heavy for collections, program and purchasing decisions.
- (attendee) Nick Scheel PhD student Freshwater and Marine Sciences

Notes:

GUI = graphical user interface

ls = listing

- ls -F
- ls -help

cd = change directory

- cd : change to home directory
- cd / : change to root directory

Get help on bash commands
ls **--help**
man ls

ls -t # list files from oldest to newest
ls -tr # list files from newest to oldest

ls -F Desktop
ls -F Desktop/shell-lesson-data/

```
cd Desktop
cd shell-lesson-data/
cd data/

# back from break. cd into data directory if you're not already there
cd Desktop/shell-lesson-data/
ls -F

mkdir thesis

mkdir -p project/data # create project/data folder inside shell-lesson-data folder
mkdir -p project/results # create project/results folder inside shell-lesson-data folder
ls -F project
pwd
cd thesis
pwd

touch my_file.txt
nano draft.txt
# type this into nano file
It's not "publish or perish" any more,
it's "share and thrive".

# exit nano
Ctrl+x
ls -l
ls -al

cd ..
mv thesis/draft.txt these/quotes.txt # rename from draft.txt to quotes.txt
ls -F thesis

mv thesis/quotes.txt ./ # move quotes.txt file from thesis directory to current directory (".")

rm -i solar.pdf  # remove a file, and ask for permission

cd molecules
ls
wc cubane.pdb # line count, word count, character count of file

wc *.pdb # line count, word count, character count of all pdb files

wc -l *.pdb # just line counts

wc -w *.pdb # just word counts

wc -m *.pdb # just character counts
```

wc -l *.pdb > lengths.txt # output line counts to file called lengths.txt

cat lengths.txt # view file contents

cd ...
ls
cat numbers.txt
sort numbers.txt # sort alphanumerically
sort -n numbers.txt # sort numerically from smallest to largest

sort -n lengths.txt
sort -n lengths.txt > sorted-lengths.txt
head -n 1 sorted-lengths.txt
head -n 3 sorted-lengths.txt
head -n -1 sorted-lengths.txt
tail -n 1 sorted-lengths.txt

sort -n lengths.txt | head -n 1 # run two commands together by using the "pipe" operator, |

wc -l *.pdb | sort -n |

wc -l *.pdb | sort -n | head -n 1 # get file with smallest number of lines

Back from break
wc -l *.txt | sort -n | head -n 5 # four of the first five files have 300 lines, one file has 240

cd ../.. # cd back up to shell-lesson-data
ls # verify you see data, molecules, etc
cd writing
pwd
ls
cat haiku.txt
grep not haiku.txt # use grep; first argument is what you're looking for, second argument is which files to check. Finds the word "not" in haiku.txt. Also note — grep is case sensitive. grep Not haiku.txt would display no output since "Not" doesn't appear in haiku.txt
grep -w The haiku.txt # find only full word "The"
grep "is not" haiku.txt # search for the phrase "is not" in the file Haiku.txt
grep -n "it" haiku.txt # report line numbers that "it" appears
grep -wn "it" haiku.txt # report line numbers where the complete word, "it", appears
grep -w -n "it" haiku.txt # same result as above command
grep -wni "the" haiku.txt # same result as above command
grep -n -w -v "the" haiku.txt # search for lines that do not contain the word "the"
grep -r Yesterday . # which directories/files contain the text, "Yesterday"

find . # report all files/folders in current folder

find . -type d # find all directories in current directory

find . -type f # find all files in current directory as well as in folders inside current directory

```
find . -name *.txt # find all txt files in current directory (ignore subfolders of current directory)

wc -l $(find . -name "*.txt") # $() has the same functionality as the pipe, |, command

cd creatures
cat basilisk.dat
head -n 3 basilisk.dat

# for loop in bash
for filename in basilisk.dat minotaur.dat unicorn.dat
do
head -n 2 $filename | tail -n 1
done
```