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/2022-06-13-uwmadison-swc/
Intro Slides: https://docs.google.com/presentation/d/10WG7GYS7Egg0v1yAjQHlHdYTuK0x7FP9iAf93rxys0Y/edit?usp=sharing
Daily Feedback Form: https://forms.gle/Bo3DXNAoEC3D5P7ZA
Pre-Workshop Survey: https://carpentries.typeform.com/to/wi32rS?slug=2022-06-13-uwmadison-swc
Post-Workshop Survey: https://carpentries.typeform.com/to/UgVdRQ?slug=2022-06-13-uwmadison-swc

Follow-Up Resources

Coding Meetup (office hours): https://datascience.wisc.edu/hub/#dropin
Data Science Hub Newsletter (upcoming workshops, seminars, jobs): https://datascience.wisc.edu/newsletter/
Workshops Via Data Science Hub: https://datascience.wisc.edu/training-resources/
Additional resources in wrap-up slides: https://docs.google.com/presentation/d/1DVWf61qHJvVhbaxKXK-xfT7d6Z1-hdi7mUEsf1vkTQI/edit?usp=sharing

Day 1

Unix Shell lesson: https://swcarpentry.github.io/shell-novice/

Sign in

Name, pronounces (optional), department/program/affiliation, describe your research/work in 1-2

sentences.

- Chris Endemann (he/him), Data Science Hub, I help researchers apply data science and machine learning tools to their research projects
- Scott Prater (he/him), UW Digital Collections Center. digital library architect, I manage projects, technical infrastructure and strategy for digital libraries
- Mary Murphy (any pronouns), Research Cyberinfrastructure, I lead the Electronic Lab Notebook SaaS and help in other research cyberinfrastructure needs
- Daven Quinn (he/him), Research scientist, Geoscience department. I am a structural geologist who builds data and software infrastructure for geoscience research, including especially https://macrostrat.org.
- Trisha Adamus (she/her), Ebling Library. I help researchs with their data needs.
- Louk (he/him) Prep research student
- Nikhil Damle (he/him) - BDS Summer Research Program
- Taylor Boldoe(she/her- BDS Summer Research Program
- Quinn White (she/her) - BDS Summer Research Program
- Nistha Panda (she/they) - BDS SROP
- Mackenzie Ray (she/her), PREP
- Trenton Mercadel (he/him) PREP
- Josselyn Muñoz (she/her) PREP
- Jiewen Chen (she/her) PREP
- Marlin Lee (he/him)
- Sarai Garcia (she/her), PREP - Incarceration and Mental Health Lab
- Lisa Padua (she/her), PREP
- Samantha Voelker (she/her), UW Health Systems - Implementation Science and Engineering Lab
- Nadeshka J. Ramirez (she/her) PREP
- Nafisa Raisa(she/her), Biomedical Data Science

I

Notes

- open git bash (windows) or terminal (mac)
- bash stands for "born again shell"

- this next step is optional, but can help with readability in bash
export PS1="$ "

- list directories
ls

cd Desktop

- download the data from: https://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip
- move the folder to desktop and unzip it (Desktop/shell-lesson-data)

- check where you are (your path) using pwd (print working directory)
pwd

- run ls to confirm that you can see your data folder, shell-lesson-data, on your desktop
ls

- use tab-complete (press tab as you start typing a folder name) to auto-complete folder names
cd shell-lesson-data/

- Run 'ls' again. You should see 'exercise-data/' and 'north-pacific-gyre' folders
ls

- move to specific data folder
cd north-pacific-gyre/
pwd
ls

- what if I want to go back to my desktop?
cd Desktop # this doesn't work

- move UP one directory
cd ..
pwd

cd ..
pwd
cd shell-lesson-data/
pwd
ls
cd north-pacific-gyre/
pwd

- move back two folders at once
cd ../..
pwd

- back from break; navigate to your Desktop folder using pwd to determine where you currently are
- use 'cd ..' to move up a directory

cd Desktop

Use -F to indicate a flag; this appends an indicator to folder and file names: * indicates executable file, / after an item indicates a folder
ls -F

view help/documentation for command
ls --help # windows
man ls # mac (use q to quit the manual view)

list items in folder with extra info about file size, creation date
ls -l

```
# human readable format
ls - lh
ls -l -h

# move up one level in directory
cd ..
pwd
G
# get info on Desktop
ls -F Desktop

# get info on shell lesson data
ls -F Desktop/shell-lesson-data
pwd

# use "full/absolute paths" to cd
cd /c/Users/yourUsername/Desktop/shell-lesson-data
pwd # in shell lesson data folder :)

# use ~ to cd to "home directory"
cd ~
```

Absolute vs. relative paths

Starting from /Users/amanda/data, which of the following commands could Amanda use to navigate to her home directory, which is /Users/amanda?

1. cd .
2. cd /
3. cd /home/amanda
4. cd ../..
5. cd ~
6. cd home
7. cd ~/data/..
8. cd
9. cd ..

Valid answers: 5, 7, 8, 9

```
# Let's cd to our Desktop
cd ~/Desktop/shell-lesson-data/exercise-data/
pwd # check cd worked

ls # confirm you have the same subfolders Trisha is showing: animal-counts/, creatures/, numbers.txt, proteins/, and writing/

cd writing/
pwd
```

```
ls -F # two files show up: LittleWomen.txt and haiku.txt

mkdir thesis # make a folder called thesis

pwd # check directory again

# make a project/data folder and a project/results folder
# the -p flag allows us to create the project folder before we create the subfolders. Without this flag, the below command doesn't work because there isn't a 'project' folder to put the new subfolders into
mkdir -p ../project/data ../project/results

ls
cd ..
cd project/
ls # data/ and results/ can be seen

cd ~/Desktop/shell-lesson-data/exercise-data/writing

# open nano editor
nano draft.txt
# add some text to file
It's not "publish or perish" anymore.
It's "share and thrive".

# save the file
Ctrl/Cmd + O # Ctrl for windows, cmd for mac
Enter
Ctrl/Cmd + X # to exit

# touch command to create a blank file
touch my_file.txt
ls

# cd to Desktop/shell-lesson-data/
cd ~/Desktop/shell-lesson-data/exercise-data/writing

# use movefile (mv) to rename a file (draft.txt -> quotes.txt)
mv thesis/draft.txt thesis/quotes.txt
cd thesis
ls # draft.txt is now named quotes.txt

# create a copy of quotes file and name it, quotations.txt
cp quotes.txt quotations.txt
ls
nano quotations.txt
Ctrl/Cmd + X # to exit
```

Suppose that you created a plain-text file in your current directory to contain a list of the statistical tests

you will need to do to analyze your data, and named it: statstics.txt
After creating and saving this file you realize you misspelled the filename! You want to correct the mistake, which of the following commands could you use to do so?

1. cp statstics.txt statistics.txt
2. mv statstics.txt statistics.txt
3. mv statstics.txt .
4. cp statstics.txt .

1 works, but the 2nd answer it a bit easier

back from break
clear # clear console

cd to writing folder
cd ~/Desktop/shell-lesson-data/exercise-data/writing
pwd
ls

cd thesis
ls

remove quotes.txt file
rm quotes.txt
ls # file is removed!

the -i option gives you a chance to back out of a delete command
rm -i quotations.txt
n # n for "no"
Enter

pwd
cd ../..
pwd # in exercise-data folder

create a backup directory l
mkdir backup
cp creatures/minotaur.dat creatures/unicorn.dat backup/
ls # we have a backup folder
cd backup
ls

cd ..
pwd # in exercise-data folder

let's move to our proteins folder
ls
cd proteins
pwd # in proteins folder

```
ls # bunch of .pdb files

# view only pdb files
ls *.pdb

# view all files that start with p and end with .pdb
ls p*.pdb

# the ? wildcard substitues ONE character rather than a variable amount of characters like the asterisk does
ls ?ethane.pdb # methane shows up
ls *ethane.pdb # ethane and methane shows up
```

When run in the proteins directory, which ls command(s) will produce this output?
ethane.pdb methane.pdb

1. ls *t*ane.pdb
2. ls *t?ne.*
3. ls *t??ne.pdb
4. ls ethane.*

```
cd ~/Desktop/shell-lesson-data/exercise-data/proteins
pwd

# wc = word count, but it returns number of lines, number of words, and number of characters
wc cubane.pdb # number of lines, number of words, number of characters

# get line/word/character count for all pdb files
wc *.pdb

# get line counts only
wc -l *.pdb

# save line counts of pdb files to a file named lengths.txt
wc -l *.pdb > lengths.txt

cat lengths.txt # another way to view file contents

less lengths.txt
pwd

cd ..
pwd # in exercise-data folder
ls

# view contents of numbers.txt
cat numbers.txt

sort numbers.txt # sorts in alphabetical order
```

```
# sort by numbers
sort -n numbers.txt

cd proteins/
pwd
ls
cat lengths.txt

# sort lengths.txt numerically
sort -n lengths.txt > sorted-lengths.txt

# view first line of document
head -n 1 sorted-lengths.txt

# view whole file
cat sorted-lengths.txt

# use echo to print things
echo hello # displays "hello" to console
echo hello > textfiles01.txt
cat textfiles01.txt # hello is saved to textfiles01.txt

# Use two greater than signs to add content to a file
echo hello >> textfiles02.txt
cat textfiles02.txt

echo hello > textfiles01.txt
echo hello > textfiles01.txt
echo hello > textfiles01.txt
cat textfiles01.txt # single "hello" in file

echo hello >> textfiles02.txt
echo hello >> textfiles02.txt
echo hello >> textfiles02.txt
cat textfiles02.txt # hello is entered in file multiple times

# the pipe operator can be used to join multiple commands together
sort -n lengths.txt | head -n 1 # shows "9 methane.pdb"

pwd # in proteins folder
wc -l *.pdb
wc -l *.pdb | sort -n

cd north-pacific-gyre/
ls
wc -l *.txt
# look at the last 5 results of the sort
wc -l *txt | sort -n | tail -n 5
```

```
ls *z.txt

# look at data for A and B files (not Z files)
wc -l NENE*A.txt NENE*B.txt

history # shows your history
# you can run a line of history like this:
!690 # to re-run line 690 of your history output

pwd # in north-pacific-gyre
cd ..
ls
cd exercise-data
pwd
ls
cd creatures
pwd

# look at first 5 lines of a few files
head -n 5 basilisk.dat minotaur.dat unicorn.dat

# print last line of first two lines of basilisk.dat
head -n 2 basilisk.dat | tail -n 1

# for loop that will run "head -n 2 $filename | tail -n 1" on each file listed in the first line of code for a for-loop
for filename in basilisk.dat minotaur.dat unicorn.dat
do
head -n 2 $filename | tail -n 1
done

for number in  0 1 2 3 4 5
do
echo $number
done

# you can also type a for-loop in one line by separating lines with semi-colons
```

Day 2: GitHub and Version Control

GitHub: https://carpentries-incubator.github.io/git-novice-branch-pr/
Sign-In, Name, pronounces (optional), department/program/affiliation,

- Mary Murphy (any pronouns), Research Cyberinfrastructure
- Chris Endemann (he/him), Data Science Hub
- Daven Quinn (he/him), Geoscience

- Louk (He/Him) PREP
- Nadeshka Ramirez Perez (she/her) PREP
- Taylor Boldoe (she/her) BDS Summer Program
- Yanissa rivera (she her) prep
- Scott Prater (he/him), UW Digital Collections Center, Helper
- Lisa Padua (she/her), PREP
- Mackenzie Ray (she/her), PREP
- Quinn White (she/her) BDS Summer Program
- Casey Schacher (she/her), Science & Engineering Libraries, Instructor
- Samantha Voelker (she/her), Implementation Science and Engineering Lab
- Sarai Garcia PREP
- Nikhil Damle (he/him) - BDS Summer Program
- Nistha Panda (she/they) BDS Summer Program
- Jiewen Chen (she/her) PREP
- Josselyn Muñoz (she/her) PREP
- Trenton Mercadel (he/him) PREP
- Marlin Lee (he/him)

Notes

```
# open terminal (mac) or GitBash (windows)
git --version # tells you version number of git

# update git on windows
git update-git-for-windows

# update git on mac
brew upgrade git

# some configuration steps
git config --global user.name "you name"

# config git to use your GitHub aNccount email address
git config --global user.email "yourEmailAddress"

# set color of user interface
git config --global color.ui "auto"

# run the below command if you're on windows
git config --global core.autocrlf true

# mac/linux users, run below command
git config --global core.autocrlf input

git config --global core.editor "nano -w"

git config --list # list all config options
```

go to github.com and sign in (create an account if you don't have one already)
https://github.com/

check if you have key-pairs setup already: the below command will produce an error if you don't
ls -al ~/.ssh

setup key pairs
ssh-keygen -t ed25519 -C "yourEmailAddress"

Enter file in which to save the key:
Enter # don't type anything, just hit enter

If you get a prompt asking you if you want to overwrite, type y for yes
y

Enter a passphrase that you can easily remember
yourPassword
Enter

check if you have key-pairs setup successfully: you should see something like id_ed25519 and id_ed25519.pub
ls -al ~/.ssh

head back over to github.com
- click profile icon in upper right corner
- click settings
- on the lefthand side, find "SSH and GPG keys" and click on it
- click the green "New SSH key" button
- add a title: we recommend the title reflects whichever machine you're currently working from

in gitbash/terminal, use cat to view your public key
cat ~/.ssh/id_ed25519.pub

- select the public key (output of cat) and Ctrl+C to copy it

- head back over to GitHub and Ctrl+V to paste the public key into the "Key" textbox
- Click the green "Add SSH key" button

head back over to GitBash
ssh -T git@github.com # hit Enter

you'll be prompted to enter the passphrase you recently entered in our above steps

if successful, it'll say "Hi yourName! You've successfully authenticated, but GitHub does not provide shell access."

in GitBash (windows) / terminal (mac)
clear # this will clear all past commands so GitBash looks a little cleaner

Next we're going to create a "repository" to store our code and all past versions of our code
pwd # check current/working directory

cd to home directory, then desktop
cd ~
cd Desktop
pwd

make a planets folder
mkdir planets
ls

cd into planets folder
cd planets
pwd

initialize rpository
git init

check folder/repository contents
ls -a # the -a flag shows hidden folders/files

check the status of your repository: are there any changes made to the directory, new files, etc.
git status

cd ..
git status # error: this folder is not a git repository

cd planets
pwd

open up nano
nano mars.txt

let's add some text to our mars.txt file
"Cold and dry, but everything is my favorite color."

exit file and save
Ctrl+x
y
enter

ls # we see mars.txt
cat mars.txt # we see the text we added to our file

let's check the status of our repository ("repo")
no commits yet
in red, we see an "untracked file" (our mars.txt file)

```
git status

# next, we will "stage" our new file
git add mars.txt

# check status of repo
# our new file now shows up in green because the file is now staged
git status

# commit our new file to the repo
git commit -m "Start notes on Mars."

# check status
git status # no new files show up anymore, and we no longer see a "No commits" message

# check log of commits
git log

# Let's add some more text to our mars.txt file
nano mars.txt
# add the following text
"The two moons may be a problem for Wolfman."

# save and exit nano
Ctrl+x
y
enter

# view updated file contents
cat mars.txt

# check status
git status # untracked file shows up in red

# check difference between our updated file (uncommitted) and the last version of the file that was committed to our repo
git diff

# stage our updated file
git add mars.txt

# commit the updated file
git commit -m "add concerns about effects of Mars' moons on Wolfman."

# check status
git status

# let's add another line to our file
nano mars.txt
```

add the following line
"But the Mummy will appreciate the lack of humidity."

save and exit nano
Ctrl+x
y
enter

check difference between new updates and previously committed file
git diff

Let's stage our file
git add mars.txt

run git diff again
git diff # no output because our changes are already staged

add modifier to diff to view diff between staged changes and last commit
git diff --staged

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

view log of commits
git log

add a new line to our mars.txt file
nano mars
add the following words (shown in bold below) to previous sentences
"The two moons may be a problem for the **nocturnal** Wolfman."
"But the **dry** Mummy will appreciate the lack of humidity."

view updated file contents
cat mars.txt

check diff
the output we see is hard to interpret — it shows which lines have changed, but it doesn't highlight new additions to existing lines (word-wise differences)
git diff mars.txt

run git diff with a modifier to see word-wise differences
git diff --color-words mars.txt

add/stage and commit all in one step by listing the updated file at the end of the commit command
git commit -m "Added a couple of random words" mars.txt

display n number of past commits
git log -1 # look at most recent commit
git log -3 # look at last 3 commits

condensed view of just the commit messages associated with each commit
git log --oneline

back from break, let's clear our GitBash
clear

check contents of mars.txt
cat mars.txt
git log --oneline

add new line to file
nano mars.txt
"An ill-considered change."

save and exit nano
Ctrl+x
y
enter

confirm change was added using cat
cat mars.txt

git diff with a HEAD modifier: compare updated file with last version committed.
git diff HEAD mars.txt # equivalent to git diff mars.txt

you can use HEAD to compare to other versions committed to your repo
git diff HEAD~1 mars.txt # compare updated file to 2nd most recent commit
git diff HEAD~2 mars.txt # compare updated file to three versions ago (3rd most recent commit)

check git log
git log --oneline # on the left you'll see unique identifiers for each commit. We can use these identifiers to compare our updated file to specific commits

git diff e3ddc76 mars.txt # you will likely have a different unique identifier than "e3ddc76". Use one of the identifier codes that show up when you run "git log --oneline"

check status
git status # mars.txt file is unstaged

let's overwrite our change

delete the last line we added (an ill considerd change)
add the following line
"We will need to manufacture our own oxygen"

save and exit nano
Ctrl+x
y
enter

```
cat mars.txt
git status

# add and commit file in one step
git commit -m "This is a change we don't want." mars.txt
cat mars.txt

# revert to previous version of our file
git checkout HEAD~1 mars.txt

cat mars.txt

# Git branches: branches are a great way to make changes to a project without impacting the previous state of the project. You can create a new branch whenever you have a series of changes you want to make. When you're done making changes, you can merge your branch into the main/master branch.

# check current branches
git branch

# create a new branch called pythondev
git branch pythondev # create a new branch
git branch # see all branches

# let's move into our pythondev branch. This branch is currently an exact copy of the work we committed to the main/master branch
git checkout pythondev # branch switch
git branch # pythondev branch now shows up green with an asterisk

# view files in branch
ls # mars.txt file shows up in this branch

# make a new python file
touch analysis.py

ls # new file can be seen

# let's add and commit the new file
git add analysis.py
git commit -m "Wrote and tested a python script"

# check status
git status

# return to master/main branch
git checkout master

# create a new branch and switch to it: the -b flag will create a branch before moving you to that branch
git checkout -b bashdev
```

```
ls

# create a analysis.sh file
touch analysis.sh
git add analysis.sh
git commit -m "Wrote and tested bash script"

git status

git checkout master

git branch

# merge branches: pythondev merge with main
git merge pythondev
ls # analysis.py shows up

# view branches
git branch

# delete our pythondev branch since we no longer need it (this branch was merged with master/main)
git branch -d pythondev

# attempt to delete bashdev branch
git branch -d bashdev # you'll get an error stating that this branch is not fully merged.

# force delete with uppercase D
git branch -D bashdev
git branch # now we just have our master branch

# take a look at mars.txt
cat mars.txt

# create a new branch called marsTemp
git branch marsTempt

nano mars.txt
# add the following line:
"I'll be able to get 40 extra minutes of beauty rest."

# save and exit nano
Ctrl+x
y
enter

# view updated file contents
cat mars.txt
```

```
# add and commit in one line
git commit -m "Add a line about the daylight on Mars." mars.txt

# switch branches
git checkout marsTemp
git branch

cat mars.txt

# add new line to our file
nano mars.txt
# new line below
"Yeti will appreciate the cold."

# save and exit
Ctrl+x
y
enter

# view file contents
cat mars.txt

# add/commit changes
git add mars.txt
git commit -m "Add a line about temperature on Mars."

# switch back to master branch
git checkout master

# attempt to merge
git checkout master # CONFLICT message shows up because we have edited our mars.txt file in both branches
cat mars.txt

# use nano to keep only the lines you want
nano mars.txt
# exit/save nano
Ctrl+x
y
enter

# add/commit
git add mars.txt
git commit -m "merged changes from marsTemp."
git status

# head back over to github.com
- under your profile icon, click "Your repositories"
- click the green "New" button to create a repository
```

- name the repo, "planets"
- click "Create repository"
- click SSH instead of HTTPS near the top of the repo window
- copy the SSH path

back in GitBash/terminal
git remote add origin PLACE_COPIED_ADDRESS_HERE

check that it worked by typing...
git remote -v
N
push your local repo to GitHub
git push origin master

enter in the passphrase you created earlier

go back to GitHub.com
- From the planets repo page, click "add file" and then "create new file"
- name the file, pullTest.txt
- scroll down and commit the file via GitHub

pull changes from GitHub to local repo
git pull origin master
enter your passphrase

ls # we now see our pullTest.txt file in our local repo

Day 3: Plotting & Programming in Python

GitHub: http://swcarpentry.github.io/python-novice-gapminder/
Sign-In, Name, pronounces (optional), department/program/affiliation

- Taylor Boldoe she/her BDS Summer Program
- Daven Quinn (he/him), Research scientist, Geoscience (Macrostrat lab)
- Lisa Padua (she/her), PREP
- Sarai Garcia (she/her, PREP)
- Casey Schacher (she/her), Science & Engineering Libraries - helper
- Samantha Voelker (she/her), Implementation Science and Engineering Lab
- Nikhil Damle (he/him) - BDS Summer Program
- Nistha Panda (she/they) - BDS summer Program
- Josselyn Muñoz (she/her) - PREP
- Mackenzie Ray (she/her), PREP
- Nadeshka Ramirez Perez (she/her) PREP
- Trisha Adamus (she/her), Ebling Library - helper
- Yanissa Rivera(she her) PREP
- Jiewen Chen (she/her) PREP
- Quinn White (she/her) - BDS Summer Program

- Marlin Lee (he/him)
- Nafisa Raisa
- Louk (he/him) Prep
- Trenton Mercadel (he/him) PREP

Notes

#Note: if you don't have the data downloaded yet, you can get it from here:
http://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip

Open Terminal (mac) or GitBash (windows)jupyter
cd into your data folder. If it's stored on your Desktop...
cd ~/Desktop/data/

jupyter lab # opens jupyter lab

File -> new -> notebook

Use the plus button to add more cells

To run a cell, use Shift+Enter. You can also use Ctrl+Enter to run a cell without advancing to the next cell

You can select different cell types: Markdown or Code. Markdown is good for leaving sections of text in your notebook.

Use asterisks to create bulleted lists in markdown
can indent to create sublists

You can use hashtags, #, to add headings. The more hashtags you use, the smaller the heading...
Largest heading
Large heading
Smaller heading
Even smaller heading

Markdown guide: https://www.markdownguide.org/basic-syntax/

Restarting a kernel clears the notebook of what was run previously — it doesn't remove any code; just resets the environment. Kernel -> Restart Kernel. It is also good practice to use Kernel -> Restart Kernel and Run All Cells when your script is complete. This guarantees that all cells will be run in order.

# in second cell
age = 42 # assign 42 to the variable age
Age = 43 # also valid
AGE = 44 # also valid
first_name = "Ralf" # assign a string to the variable first_name

```
prtin(first_name) # print the variable contents

# print multiple things using commas
print(first_name, "is", age, "years old")

# restarting kernel and running all cells would produce an error with the below code
print(last_name)
last_name = "Kotulla"

# instead, we want to define last_name prior to printing it
last_name = "Kotulla"
print(last_name)

# adding value to a variable
age = age + 3
print("age in three years", age)

atom_name = "Helium"
print(atom_name)
atom_name = 'Helium' # you can use single-quotes or double-quotes. Try to stick with one for consistency.
print(atom_name)

# print on multiple lines using three single/double quotes
atom_name = """Helium
is an atom"""
print(atom_name)

# print just first character of atom_name
atom_name = "Helium"
print(atom_name[0])

# print multiple characters of atom_name
atom_name = "Sodium"
print(atom_name[0:3]) # print first three characters

# print length of string (number of characters)
print(len(atom_name))
print(len(Atom_name)) # produces an error because we have a typo (A in atom shouldn't be capitalized)

# Try to use meaningful variable names. The below code works, but it uses meaningless variable names, and it is hard for the average human to interpret at a glance. Use names that reflect the data you are storing in a variable (e.g., age and first_name in this case).
flabadad = 42
kshd = "Ralf"
print(kshd, 'is', flabadad, 'years old')

# print type
print(type(52)) # 52 is an integer ('int')
```

```
print(type("some text")) # string type

print(type(first_name)) # string type

print(type(52.4)) # float type (number with decimal)

print(5 - 3) # prints 2

print('hello' - 'h') # can't subtract strings

# you can add strings
full_name = "Ralf" + " " + "Kotulla"
print(full_name)

# you can also multiple a string with an iteger
separator = "=" * 25
print(separator)

# print length of full_name))
print(len(full_name))

# numbers do not have length
print(len(52)) # produces an error

# you can't add strings with integers
print(1 + '2') # error

# you can "cast" a string to an integer using int()
print(1 + int('2')) # adds the integers, 1 and 2

# you can also do the reverse: cast integer to a string
print(str(1) + '2') # 12

# you can print both strings and numbers together
print("half is ", 1/2.0)

# use two asterisks to raise a number to a power, e.g. 3 squared below
print("three squared is", 3.0 ** 2)

variable_one = 1
variable_two = 5 * variable_one
variable_one = 2
print(variable_one, variable_two) # 2 5

# you can add comments to python by using hashtags

# this sentence is a comment, and not executed by python
adjustment = 0.5 # this is also ignored
```

```
print("before") # print is a function, and "before" is the argument we pass to this function
print() # prints nothing / newline
print("after")

# functions return things — the print functoin returns None
result = print("example")
print("result of print is", result) # None is the default output of a function

# the max function
print(max(1,2,3)) # 3
print(max('a', 'f', '0')) # f

print(round(3.14)) # round to nearest integer
print(round(3.14, 1)) # round to first decimal place

my_string = "Hello World!"
print(len(my_string))
print(my_string.swapcase()) # the string class has a set of functions associated with it

# other string functions
print(my_string.isupper()) # checks whether all characters (letters) of the string are uppercase.
print(my_string.upper()) # convert string to upper case
print(my_string.upper().isupper()) # convert string to uppercase and then check if entire string is
uppercase

# get help
help(str) # see functions available for strings
help(round) # see documentation for round function

# common errors
name = 'Ralf # Syntax Error: EOL (end of line) while scanning string literal; you need to add a single
quote to the end of Ralf to make this line work

age = = 42 # invalid syntax error. Python does not know how to interpret two equals signs that are
separated by a space

print("hello World" # Syntax Error: unexpected EOF (end of file). You need to close parentheses to fix
this

age = 52
remaining = 100 - AGe # NameError: name 'AGe' is not defined. Check your variable name spelling
when you see this kind of error. AGe should be spelled as "age". There is no variable AGe defined in our
script.

age = 52
 remaining = 100 - age # unexpected indent

# let's import the math library/package
```

```
import math

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

# get help on math: you can only do this after importing the math library
help(math)

# import just two functions from the math library
# careful not to use cos and pi as variable or function names in your script. If you do, you will override the math library's cos and pi functions.
from math import cos, pi
print("cos(pi) is ", cos(pi))

# you can give libraries a nickname
import math as m # with this syntax, we reference m. when we want to use the math library
print("cos(pi) is ", m.cos(m.pi))

# Intro to the pandas library. First, we'll import the library
import pandas as pd

# read in a csv file
data = pd.read_csv('data/gapminder_oceania.csv')
print(data)

# run a shell command in python!
!ls
!pwd

# specify index column to control which variable should be displayed on the rows of the data table
data = pd.read_csv('data/gapminder_gdp_oceania.csv', index_col="country")
print(data)

# we can get more info on our data with .info()
data.info()

# print columns of your data
print(data.columns)

# print a transposed version (swap rows and columns) of your data
print(data.T)

# get some descriptive stats on your data (mean, std, min, max, count, etc.
print(data.describe)

# let's read in a different file
data = pd.read_csv("data/gapminder_gdp_europe.csv", index_col="country")
print(data)
```

```
# use iloc to print value in first row, first col
# Note: python indexing starts at 0 (unlike R, which starts at 1)
print(data.iloc[0, 0])

# use loc to index based on row and column names
# Typically, you'll want to use .loc instead of .iloc because it is more stable. That is, even if your csv file was reorganized, the code would still work when using .loc.
print(data.loc['Albania', 'gdpPercap_1952'])

# print all columns for albania by using a colon to index all columns
print(data.loc['Albania', :])

# print all rows (countries) for a specific column (gdpPercap_1952)
print(data.loc[:, 'gdpPercap_1952'])

# print range of rows and columns
print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'])

# get max of a subset of data
print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'].max())

# get min of a subset of data
print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'].min())

# save subset of data into a new variable
subset = data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972']
# Note: the \n in the print statement below adds a newline
print("subset of data:\n", subset)

# return trues and falses for a specific condition. In this example, we check where in our data gdp is > 10000
print("large gdp values:\n", subset > 10000)

# generate a mask of True/False values. Use this mask to display values of gdp that are > 10000
mask = subset > 10000
print(subset[mask]) # Note: our mask is a matrix, and so we don't need to specify both rows and columns when using mask as an index

# describe data that has a gdp > 10000
print(subset[mask].describe())

# let's create another mask that will be True for all values that are greater than the mean of gdp for a specific year
# Note: when you sum over "Boolean data" (i.e., data containing True and False values), the Trues are treated as 1's, and the Falses are treated as 0's.

mask_higher = data > data.mean()
wealth_score = mask_higher.aggregate('sum', axis=1) # axis=1 means we will sum across all rows instead of columns (axis=0)
```

```
print(wealth_score)

# import plotting library
import matplotlib.pyplot as plt

time = [0, 1, 2, 3]
position = [0, 100, 200, 300]
plt.plot(time, position)
plt.xlabel("Time [hours]")
plt.ylabel("Position [km]")

# if you're working in a different python IDE, you may need to run the following to get your plot to show up
plt.show()

# plot using pandas
import pandas as pd
data = pd.read_csv("data/gapminder_gdp_oceania.csv", index_col="country")

years = data.columns.str.strip('gdpPercap_')
# Convert year values to integers, saving results back to dataframe
data.columns = years.astype(int)
data.loc['Australia'].plot()

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

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

# use plt.plot() to plot
years = data.columnsgdp_australia = data.loc['Australia']
plt.plot(years, gdp_australia, 'g--')

# plot multiple datasets together
# Select two countries' worth of data.
gdp_australia = data.loc['Australia']
gdp_nz = data.loc['New Zealand']
# Plot with differently-colored markers.
plt.plot(years, gdp_australia, 'b-', label='Australia')
plt.plot(years, gdp_nz, 'g-', label='New Zealand')
# Create legend.
plt.legend(loc='upper left')
# add axis labels
plt.xlabel('Year')
plt.ylabel('GDP per capita ($)'
```

Day 4: Plotting & Programming in Python Continued

GitHub: http://swcarpentry.github.io/python-novice-gapminder/
Sign-In, Name, pronounces (optional), department/program/affiliation

- Chris Endemann (he/him), Data Science Hub
- Daven Quinn (he/him), research scientist in Geoscience
- Nadeshka Ramirez (she/her) PREP
- Lisa Padua (she/her) PREP
- Nikhil Damle (he/him) - BDS Summer Program
- Taylor Boldoe she/her BDS Summer Program
- Casey Schacher (she/her), Science & Engineering Libraries - helper
- Louk (he/him) PREP
- Nistha Panda (she/they) BDS Summer Program
- Samantha Voelker (she/her), Implementation Science and Engineering Lab
- Jiewen Chen (sher/her) PREP
- Mackenzie Ray (she/her), PREP
- Sarai Garcia (she/her), PREP
- Trenton Mercadel (he/him) PREP
- Marlin Lee (he/him)
- Trisha Adamus (she/her), Ebling Library
- Quinn White (she/her) BDS Summer Program
- Josselyn Muñoz (she/her) PREP
- yanissa rivera (she/her) PREP

Notes

First, open JupyterLab:

in Terminal/Git Bash shell
cd ~/Desktop
cd python-novice-gapminlder-data # or whatever you called your data folder from yesterday
ls
jupyter lab # opens jupyterlab

In JupyterLab, create a new notebook and change its name to "lists"
(Pick "Notebook">"Python 3" (ipykernel))
Once in new notebook, right-click on title to rename

We're creating lists!

Run this code in your Python notebook using Shift+Enter

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

```
print('first item of pressures:', pressures[0])
print('last item of pressures:', pressures[4])

# also can get the last item of pressures using
pressures[len(pressures)-1]
OR
pressures[-1]

--

# Lists' values can be replaced by assigning to them

pressures[0] = .265
print(pressures)

# Adding items to a list

# incorrect way
primes = [2, 3, 5]
print(primes)
primes = primes + [7]
# This works but not the best way to do it

# Why avoid this? Adding items to a list in this way forces python
# to temporarily store multiple lists in your computer's memory:

    # 1) initial primes list
    # 2) [7]
    # 3) new list that has a length = len(primes) + 1

# After the 3rd new list is created, the contents of primes and [7]
# is transferred to this new list.

# This might not sound like a big deal,
# but when working with large datasets, creating copies of
# large lists can eat up your computer's memory very quickly.

# Instead, we're going to use what's called an "in-place" operation

# # An in-place operation is an operation that changes the
# content of a variable directly — this avoids having to make a copy
# of our list when we want to add data to it.

primes = [2, 3, 5]
primes.append(7)
# This operation directly changes the value of primes
print(primes) # [2, 3, 5, 7]
```

```
# Combine two lists

teen_primes = [11, 13, 17]
middle_aged_primes = [37, 41, 43]

primes.extend(teen_primes)
primes.extend(middle_aged_primes)

# primes will now contain all primes in order

# remove items from a list

primes = [2, 3, 5, 7, 9]
del primes[4]
print(primes) # [2, 3, 5, 7]

# initialize an empty list

my_list = []
my_list.append(1)
print(my_list) # [1]

# lists can contain different types
goals = [1, 'Create lists', 2, 'Extract items from lists']
# This list contains both strings and integers

# You can treat strings like lists!

element = "carbon"
print(element[0]) # "c"

# ...but you can't replace/edit items like in a list (Strings are immutable)
element[0] = "C" # Returns a TypeError

# instead, you can use a string modification function
element = str.replace(element, 'c', 'C')

# common error : Indexing error
print('99th item of element:', element[98]) # Returns IndexingError: string index out of range
```

Challenge exercise: Fill in the blanks so that the program below produces the following output...

```
first time: [1, 3, 5]

second time: [3, 5]
values = ____
values.____(1)
values.____(3)
values.____(5)
```

```
print('first time:', values)
values = values[____]
print('second time:', values)

# Answers

values = []
values.append(1)
values.append(3)
values.append(5)
print('first time:', values)
values = values[1:]
print('second time:', values)

# Cast a string to a list

print('string to list:', list('tin'))
print('list to string:', "".join(["g", "o", "l", "d"]))

# Output:
# string to list: ['t', 'i', 'n']
# list to string: gold

# print the last letter:

element = 'helium'
print(element[-1])
# equivalent to print(element[len(element)-1])

# step through a list using slices

element = 'fluorine'
begin_index = 0
end_index = len(element)
stride = 2

print(element[begin_index:end_index:stride]) # "furn"

# if starting and ending at beginning and end, respectively, you can write
print(element[::stride]) # "furn"

# How to print 'lre' from element
element[1::3] # lre

# Sorting lists

result = sorted(numbers)

# or, using an in-place operation
```

result = numbers.sort()

result will be 'None', but numbers will contain the sorted list

making a copy of a list (or not)

old = list('gold')
new = old
new[0] = 'D'
# Both new and old are ["D", "o", "l", "d"]!
# only a "reference" to the data is copied

# if you copy with a slice, you make a true copy
new = old[:]

Now create a new notebook called 'For loops'

for loops allow you to execute code for each value of a list

my_list = [2, 3, 5]

# print each item
for item in my_list:

- print(item)

tab before print is important

can include multiple lines of code

primes = [2, 3, 5]
for p in primes:

- squared = p ** 2
- cubed = p ** 3
- print(p, squared, cubed)

# use range to loop through a range of values
for number in range(0, 3):

- print(number)

# if starting at 0, you can omit first argument
for number in range(3):

- print(number)

# use for loops to sum up values in a collection
total = 0 # accumulator variable
for number in range(1, 11):

- print(number)
- total = total + number

```
print(total) # 55
# In iPython notebook, hit Shift+tab to get out of for loop

# can also loop through strings
total = 0
for char in "tin":
    print(char)
    total = total + 1
print(total) # 3
```

Exercises - go to 10:30

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

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

# New notebook: rename it to "conditionals"

# running code based on some condition to be met (if-statements)

mass = 3.54
if mass > 3.0: # don't forget colon!
    # if condition is true, this will execute
    print(mass, "is large")
# 3.54 is large

# conditionals are often used inside of loops

masses = [3.0, 3.7, 9.2, 1.8, 1.7]
for m in masses:
    if m > 3.0:
        print("m is large")



# can also use an 'else' statement
```

```
masses = [3.0, 3.7, 9.2, 1.8, 1.7]
for m in masses:
    if m > 3.0:
        print("m is large")
    else:
        print("m is small")

# checking multiple conditions

masses = [3.0, 3.7, 9.2, 1.8, 1.7]
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")

# Once a condition is met all other conditions are ignored

grade = 85
if grade >= 70:
    print('grade is C')
elif grade >= 80:
    print('grade is B')
elif grade >= 90
    print('grade is A')

# This will print 'grade is C' for everything, ignoring later cases

# can "evolve" the value of variables in a for loop
velocity = 10.0
for i in range(5):
    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)

# check multiple conditions in one if statement

# check multiple conditions in one 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):
    print('object' + str(i) + ': mass=' +
        str(mass[i]) + ', velocity=' + str(velocity[i]))
    # check conditions
    if mass[i] > 5 and velocity[i] > 20:
        print("Fast heavy object. Duck!")

    if mass[i] <= 2 or velocity[i] <=20:
        print('object is either light, slow, or both')

#### Exercise
# Modify this program so that it only processes files with fewer than 50 records/countries.

import glob
import pandas as pd
for filename in glob.glob('data/*.csv'):
    contents = pd.read_csv(filename)
    if len(contents) < 50:
        print(filename, len(contents))



# Create a new notebook called "loop-through-data"

import pandas as pd

files = ['data/gapminder_gdp_africa.csv', 'data/gapminder_gdp_asia.csv']
for filename in files:
```

- # read in our data
- data = pd.read_csv(filename, index_col='country')
- print(filename)
- print(data.min())

```
# use glob.glob to find sets of files matching a pattern
import glob
print(glob.glob("data/*.csv"))

# if there are no text files, you end up with an empty list
print(glob.glob("data/*.csv"))
# result: []

# use glob to process a batch of files
for filename in glob.glob("data/*.csv"):
```

- data = pd.read_csv(filename)
- # print the min GDP across all countries in a file for the year 1952
- print(filename, data['gdpPercap_1952'].min())

```
# Result:
```

gapminder_gdp_americas.csv 1397.717137

gapminder_gdp_europe.csv 973.5331948
gapminder_all.csv 298.8462121
gapminder_gdp_oceania.csv 10039.59564
gapminder_gdp_africa.csv 298.8462121
gapminder_gdp_asia.csv 331.0

Exercise
Which of these files is not matched by the expression
glob.glob('data/*as*.csv')?

1. data/gapminder_gdp_africa.csv
2. data/gapminder_gdp_americas.csv
3. data/gapminder_gdp_asia.csv

Answer: 1

Exercise
Modify this program so that it prints the number of records in the file that has the fewest records.

```
import glob
import pandas as pd
fewest = float('Inf')
for filename in glob.glob('data/*.csv'):
    dataframe = pd.read_csv(filename)
    fewest = min(fewest, dataframe.shape[0])
print('smallest file has', fewest, 'records')
```

Functions

Functions allow us to:
- easily re-use code without having to re-write it
- break up complicated software into small, smanageable components

```
#create function
def print_greeting():
    print('Hello!')

#call function
print_greeting()

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

#call function using positional arguments
print_date(1871, 3, 19)

#call function using named arguments
print_date(month=3, day=19, year=1871)
```

Functions can also return outputs

#create function
```
def average(values):
    if len(values) == 0:
        return None
    return sum(values) / len(values)
```

#call function
```
a = average([1, 3, 4])
```
print('average of actual values:', round(a, 2))

Challenge
a = average([])
print('average of empty list =', a)
ANSWER: Returns: average of empty list = None

Every function returns something - even if there is no explicity return statement

Ex:
```
result = print_date(1871, 3, 19)
```

print('result of call is:', result)

Exercise: Identifying Syntax Errors
1. Read the code below and try to identify what the errors are without running it.
2. Run the code and read the error message. Is it a SyntaxError or an IndentationError?
3. Fix the error.
4. Repeat steps 2 and 3 until you have fixed all the errors.

```
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.")
```

What's wrong with this example?
result = print_time(11, 37, 59)

```
def print_time(hour, minute, second):
    time_string = str(hour) + ':' + str(minute) + ':' + str(second)
    print(time_string)
```

Exercise

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(____):
    data = ____
    return ____
```

Exercise
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 ____:
        if ____:
            return ____
```

#scope of variables

```
pressure = 103.9 #this is a global variable

def adjust(t):
    '''Adjust calculates temperature based on pressure'''  #this is a documentation string (i.e. docstring)
    temperature = t * 1.43 / pressure #this is a local variable
    return temperature

print('adjusted:', adjust(0.9))
```
print('temperature after call:', temperature) #this will return an error because you can't reference a loca variables at the global scope

#this will print the docstring
help(adjust)

Programming style
Use assertions to check for internal errors. Good practice to use if there assumptions being made in your code (ex: a value is greater than 0. or a value is a string. etc)

#create function
```
def calc_bulk_density(mass, volume):
    '''Return dry bulk density = powder mass / powder volume.'''
    assert volume > 0   #checks if the assumption that volume > 0 is true
    return mass / volume
```

#call function
calc_bulk_density(10, -1) #since volumn < 0, you get an error

#Wrap up
Daily Feedback Form: https://forms.gle/Bo3DXNAoEC3D5P7ZA

Coding Meetup (office hours): https://datascience.wisc.edu/hub/#dropin

Data Science Hub Newsletter (upcoming workshops, seminars, jobs):
https://datascience.wisc.edu/newsletter/