jWelcome 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/policies/coc

---

Welcome to Software Carpentry!

Links:

Workshop Website: https://uw-madison-datascience.github.io/2026-01-12-uwmadison-swc/
Setup: https://uw-madison-datascience.github.io/2026-01-12-uwmadison-swc/#setup
Intro slides: https://docs.google.com/presentation/d/1VnXIE0cE89_LYABrId4BGTxSmWP2DGtBvKIg5pKkefo/edit?usp=sharing
Feedback Form: https://forms.gle/EjWgzbn9rouMmoKZ6
Coding Meetup: https://hub.datascience.wisc.edu/consultation/#dropin
Wrap-up Slides: https://docs.google.com/presentation/d/16OMOcmoL97qaduw_PG4fGlDRVVhfE9ysJUBMJpr0LBg/edit?usp=sharing

Day 5

Sign-in

- Annika Pratt (she/her), Plant Pathology
- Peter Cruz Parrilla (he/him), Chemistry, Helper
- Tracy Reuter (she/her), DAPIR, Observer
- Sarah Stevens (she/her/her), Data Science Hub - Instructor
- Celeste Qin(She/her/her), Computer Science
- Jose Sanz, Mechanical Engineering, Energy harvesting from the sea
- Sven Sierra, Cereal Crop Genetics
- Ibrahim Momohjimoh, (He/him), Materials Science and Engineering, TEM data analysis.
- Aadhi Balaji, Computational Biology
- Kayannah Luther, Neurobiology
- Likulunga Emmanuel Likulunga, From botany working on plant-soil-microbial interactions
- Saketh Edpuganti, Data Engineering

- Elizabeth Hrycyna, Entomology
- Aidan Horn , Math/Genetics
- Suihan Zhang, Biochemistry, Protein function and sequence bioinformatics
- Dante

Questions from yesterday's feedback:
how do you know when to use () vs []?

- parentheses and brackets can have different meanings in different contexts. From what we have seen, parentheses wrap the arguments to a function and calls it, when they are placed after a function name. Parentheses alone (not next to a function name) will mean "this is a tuple", which is a different data type similar to a list. Brackets placed directly after a list will wrap the index or slice of a list. Brackets alone will mean "this is a list" and convert what's inside to a list data type.

I would like to learn more how range could be useful for indexing in a function? or in a loop? It was a good thing to mention that range could be useful and now i'm curious

- If we just have one list we want to iterate over, it is better to just iterate over the items in the list (for item in list:) rather than interating over the indices (for i in range(10):). With the first approach, you can access the current list value simply with the variable "item". With the second approach, it's more cumbersome and you have you access the value through indexing the original list, such as "list[i]".The latter, again, is more tedious to type.
- Let's say you have two lists you want to iterate over, such as the example at the end of the Conditionals lesson where we had a list of masses and a list of velocities. Since for loop only takes in one iterable, we can't iterate through both lists simultaneously. However, if we want to compare the items in two lists at the same index position in both lists (i.e. the first item in both lists, the second item in both, etc.), having the index we want to evaluate, and then getting the information we want to compare in both lists by "masses[i]" and "velocities[i]" becomes a viable approach. For this case, if we build a for loop as "for i in range(5):" (assuming both lists have length of 5), then the value of i represents the indexed positions inside the lists that we would like to access, allowing us to compare multiple, aligned values of lists simultaneously.
- Of course, there are other, more advanced ways to accomplish this, such as using python's "enumerate()" function, which is a good next step to look into! One more tip: if you want to ensure that your range produces only the indices of a list, without having to count the items yourself, you can use range(len(list)), which gets the range from the determined length of a list.

What are lambda functions?

- lambda functions are useful when we have short functions, and it is slightly more time efficient to write your function in a single line rather than defining and structuring it. For example:
    - def add_one(x):
        - return x + 1
- We could write an equivalent lambda function as
    - lambda x: x+1
- The lambda keyword starts the lambda function, then we pass the arguments (comma separated), a colon, and then the code statement. Note that for a lambda function, the colon does not require an indented code block. You usually use lambda functions as parameters to functions that take as arguments other functions. If not created as a parameter, you need to store them in a variable, such as:

    - plus_one = lambda x: x+1
- When stored in a variable, they are no different than a defined function. To use it, you have to call the variable name as a function, such as plus_one(10), to execute your function.
- Overall, they can be useful, but there is no case that a lambda function can handle that a defined function can't.

Notes

Activate the software carpentries virtual environment
Virtual environments are useful for installing packages, testing code with old/new packages, etc.
The first part of your prompts may look slightly different from the instructor's (% or $ or >)
Within terminal, run:
conda activate carpentries
python --version
Change directory to the swc folder on your Desktop and list .csv files in the data directory
cd Desktop/swc
ls data
miniforge - type dir instead of ls for the above line
Make a figures directory
mkdir figs
Download: https://github.com/carpentries-incubator/swc-ext-python/blob/main/scripts/gdp_plots.py
Move this .py file into your project folder (Desktop/swc) manually or with a terminal command
mv ~/Downloads/gdp_plots.py
use tab completion to write path names - this helps find typos!
Open a NEW terminal (cmd + n for MacOS opens a new window; cmd + t opens a new tab) to allow us to run code in Jupyter and in a terminal

- conda activate carpentries

Open gdp_plots.py in Jupyter Notebook
pandas is for data wrangling and matplotlib is for plotting; note that .py file syntax differs from .ipynb file (matplotlib.pyplot)
Run the .py script from terminal:
python gdp_plots.py
The .py file will not flag errors, like we saw with the .ipynb file.
 # Version control in terminal
 # !!! Windows users, please install GitBash:

- # https://git-scm.com/install/windows
- # Choose Nano as your default text editor
- # Choose Override the default branch to main
- # Configure with these commands:
- git config --global user.name “YOURNAME”
- git config --global user.email “youremail@yourdomain.edu”
- git config --global core.editor “nano -w”

Next open Terminal/GitBash, initialize git, and check git status

```
git init
git status
# Tell git what to exclude from version control
nano .gitignore
# List files to ignore, like the .csv files in the data directory:
.DS_Store
.ipynb_checkpoints/
data/
*.ipynb
# Commit the gitignore steps, check git status, and commit the first version of gdp_plots.py
git add .gitignore
git commit -m "creating git ignore file"
git status
git add gdp_plots.py
git commit -m "first draft of plotting script"
```
 # Make changes to a .py script and track changes with version control
```
# Make some edits in the gdp_plots.py file, such as:
filename = 'data/gapminder_gdp_oceania.csv'
data = pandas.read_csv(filename, index_col = 'country').T
ax = data.plot(title = filename) # Use the filename as the plot title
ax.set_xlabel("Year")
ax.set_ylabel("GDP Per Capita")
ax.set_xticks(range(len(data.index))) # set the x location
ax.set_xticklabels(data.index, rotation = 45) # set the x label
```

V2 of script:
```
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'

# read data into a pandas dataframe and transpose
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)

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

Now, switch back to the terminal now, and run
python gdp_plots.py
The plot now has axes, labels, a title, a legend. The axes are slighly offset, so let's fix that
Update the ax.set_xticklabels(data.index, rotation = 45, ha = 'right')

Commit this version to our history
git add gdp_plots.py
git commit -m "improving plot format"

We want to pass command line arguments to python.
That way we don't hard code the files.

Create a new .py file in jupyter lab.
import sys
print('sys.argv is', sys.argv)
ctrl+s will ask to rename the file.
Rename to "args_list.py"
From the terminal run, python args_list.py
Now run: python args_list.py first second third
the arguments passed through the commandline are always passed as strings
python args_list.py 1 2 3 #numbers become strings!
back in our gdp_plots.py script
import sys
filename = sys.argv[1]

```
# v3 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
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()

Back to terminal
python gdp_plots.py data/gapminder_gdp_oceania.csv
The same plot as the hard coded version shows up.
sys is a library (like pandas, matplotlib, etc) that we can import and use methods from
It contains methods to interact with your system, such as reading the arguments from the
command line.

git add gdp_plots.py args_list
git status #should show both files added
git commit -m "adding command line arguments"
show all versions with "git log --oneline"

# split() method for strings in Python
filename = 'my-data.csv'
split_name = filename.split('.') # split takes a separator as a string in its input
print(split_name)
print(split_name[0])

We'll save the figures with unique names instead of displaying them.
split_name1 = filename.split('.')[0]
split_name2 = split_name1.split('/')[1]
save_name = 'figs/' + split_name2 + '.png'
plt.savefig(save_name)

#V4 of gdp_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]

# read data into a pandas dataframe and transpose
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, ha = 'right')

save the plot with a unique name

```
# data/gapminder_gdp_XXX.csv
split_name1 = filename.split('.')[0] # data/gapminder_gdp_XXX
split_name2 = split_name1.split('/')[1] # gapmindr_gdp_XXX
save_name = 'figs/' + split_name2 + '.png'
plt.savefig(save_name)
```

We don't want to track the figs folder
In terminal:
nano .gitignore
add the two lines below and save the .gitignore file:

- *.png
- figs/

```
git add .gitignore gdp_plots.py
git commit -m "Saving plots to a unique name"
```

To make the script work on multiple files, we can 1) add a for loop in the python script, or 2) add a for loop in the Unix shell. We can create branches for our code to test both approaches.

```
git branch py-loop
git branch sh-loop

git switch py-loop

#V5 of gdp_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

for filename in sys.argv[1:]:
```

- # read data into a pandas dataframe and transpose
- 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, ha = 'right')
- # save the plot with a unique name
- # data/gapminder_gdp_XXX.csv
- split_name1 = filename.split('.')[0] # data/gapminder_gdp_XXX
- split_name2 = split_name1.split('/')[1] # gapmindr_gdp_XXX
- save_name = 'figs/' + split_name2 + '.png'
- plt.savefig(save_name)

In terminal:
python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_gdp_asia.csv
ls -la figs # shows that the figures were created!
git add gdp_plots.py
git commit -m "added plot generation for multiple files"
git log --oneline # shows the py-loop and sh-loop branches have diverged.
git switch sh-loop
You may need to close and reopen your gdp_plots script in jupyter to see the changed version in the new branch (sh-loop)

Open a new text file, named "gdp_plots.sh"
for filename in data/gapminder_gdp_oceania.csv data/gapminder_gdp_asia.csv
do

- python gdp_plots.$filename

done

save and run from cmd as "bash gdp_plots.sh"
ls -la figs should show folder containing updated figures
git add gdp_plots.sh
git commit -m "wrote bash script to call python plotter"
git log --oneline --all --graph # shows both sh-loop and py-loop diverged from each other and from main branch
time bash gdp_plots.sh
user time is the clock time the function takes to run.
git switch py-loop
time python gdp_plots.py data/gapminder_gdp_oceania.csv data/gapminder_gdp_asia.csv
python loop seems faster, so let's keep that.
git switch main
git merge py-loop

Use glob to make it easier to find relevant files for our script

#V6 of gdp_plots
import sys
import glob
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt
#check for -a flag in arguments
if "-a" in sys.argv:

- filenames = glob.glob("data/*gdp*.csv")

else:

- filenames = sys.argv[1:]

for filename in filenames:

- # read data into a pandas dataframe and transpose
- data = pandas.read_csv(filename, index_col = 'country').T
- if "continent" in data.index:
    - data.drop("continent", inplace = True)
- print(filename)
- print(data.head())
- # 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, ha = 'right')
- # save the plot with a unique name
- # data/gapminder_gdp_XXX.csv
- split_name1 = filename.split('.')[0] # data/gapminder_gdp_XXX
- split_name2 = split_name1.split('/')[1] # gapmindr_gdp_XXX
- save_name = 'figs/' + split_name2 + '.png'
- plt.savefig(save_name)
- Run the new script:
- python gdp_plots.py

```
#V7 of gdp plots
import sys
import glob
import pandas
# we need to import part of matplotlib
# because we are no longer in a notebook
import matplotlib.pyplot as plt

# check for -a flag in arguments
if "-a" in sys.argv:
    filenames = glob.glob("data/*gdp*.csv")
else:
    filenames = sys.argv[1:]



for filename in filenames:
    # read data into a pandas dataframe and transpose
    data = pandas.read_csv(filename, index_col = 'country').T
    if "continent" in data.index:
        data.drop("continent", inplace = True)

    # 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, ha = 'right')

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

    # commit version 7 in terminal
    # git commit -m "adding flag to remove run on all data" gdp_plots.py

# GitBash opening a figure from command line
start figs/gapminder_gdp_oceania.png
# a **silent error** means something did not work, but we cannot see why
# In Jupyter, make sure that additional arguments/flags have been provided by the user
if len(sys.argv) == 1 # there is always at least 1 argument (the script itself)
```

- print("Not enough arguments have been provided") # print some messages to help debug
- print("Usage: python gdp_plots < FILENAMES >")
- print("Options:")
- print("-a : plot all gdp datasets in the data directory.")

```
# back in terminal, try running without the required "-a" arg:
python gdp_plots.py
# See the error messages we wrote above, and commit the change:
git commit -m "added handling for missing args"
```

SILENT ERRORS CHALLENGE

Silent errors can be difficult to anticipate. If we try to run our program from another directory with the -a flag, we still don’t see any errors, but it also doesn’t do anything. This is because when we do the -a flag here, there are no .csv files in the directory, so our filenames list is empty. Add a check for this case and an error message when the issue arises.

```
if "-a" in sys.argv:
    filenames = glob.glob("data/*gdp*.csv")
    if filenames == []
    print("No files found in this folder")

# Commit the change in terminal:
git add gdp_plots.py
git commit -m "added handling for no files in data directory"
```

 # Refactoring Code

```
# First create a branch called 'refactor' to keep the working version safely in the 'main' branch
git switch -c refactor
git status
```

Refactor code to create 4 functions:

- parse_arguments() - gets the input from argv[], returns a list of file names
- create_plot() - takes one file name as input, creates one plot and writes it to the fig folder
- create_plots() - takes a list of files as input, calls create_plot() for each element in the list
- main() - calls parse_arguments() and create_plots()

```
# outline for refactor
def parse_arguments(argv):
    """
    Parse the argument list passed from the command line
    (after the program filename is removed) and return a list
    of filenames.

    Input:
    ------
        argument list (normally sys.argv[1:])

    Returns:
    --------
        filenames: list of strings, list of files to plot
    """
# Paste in your existing code for the block "make sure additional arguments..."
# Remember to indent the code (use tab or 4 spaces)
# Change "sys.argv" to just "argv" to have proper scope
# Add a line at the very end to return the filenames:
return filenames

def create_plot(filename):
    """
    Creates a plot for the specified
    data file.

    Input:
    ------
        filename: string, path to file to plot

    Returns:
    --------
        none
    """
# Paste your existing code for the loop "read data into a pandas dataframe and transpose"...

def create_plots(filenames):
    """
    Takes in a list of filenames to plot
    and creates a plot for each file.
```

```
    Input:
    ------
        filenames: list of strings, list of files to plot

    Returns:
    --------
        none
    """
# Paste and indent your existing code:
```

- for filename in filenames:
    - create_plot(filename)

```
def main():
    """
    main function - does all the work
    """
# All of your code should be refactored into the above functions now!
```

- # parse arguments
- files_to_plot = parse_arguments(sys.argv[1:])
- # generate plots
- create_plots(files_to_plot)

```
# call main
main()

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


def parse_arguments(argv):
    """
    Parse the argument list passed from the command line
    (after the program filename is removed) and return a list
    of filenames.

    Input:
    ------
        argument list (normally sys.argv[1:])

    Returns:
    --------
        filenames: list of strings, list of files to plot
```

```
    """
    # make sure additional arguments or flags have been provide by user
    if argv == []:
        # why the program will not continue
        print("Not enough arguments have been provided")
        # how this can be corrected
        print("Usage: python gdp_plots < FILENAMES >")
        print("Options:")
        print("-a : plot all gdp datasets in the data directory")

    # check for -a flag in arguments
    if "-a" in argv:
        filenames = glob.glob("data/*gdp*.csv")
        if filenames == []:
            # file list is empty (no files found)
            print("No Files found in this folder.")
            print("Make sure the data is localed in the data folder")
    else:
        filenames = argv
    return filenames

def create_plot(filename):
    """
    Creates a plot for the specified
    data file.

    Input:
    ------
        filename: string, path to file to plot

    Returns:
    --------
        none
    """
    # read data into a pandas dataframe and transpose
    data = pandas.read_csv(filename, index_col = 'country').T
    if "continent" in data.index:
        data.drop("continent", inplace = True)

    # 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, ha = 'right')
```

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

def create_plots(filenames):
    """
    Takes in a list of filenames to plot
    and creates a plot for each file.

    Input:
    ------
        filenames: list of strings, list of files to plot

    Returns:
    --------
        none
    """
    for filename in filenames:
        create_plot(filename)

def main():
    """
    main function - does all the work
    """
    # parse arguments
    files_to_plot = parse_arguments(sys.argv[1:])

    # generate plots
    create_plots(files_to_plot)


# call main
main()

# Commit the refactored version
git status
git add gdp_plots.py
git commit -m "refactored code into functions"

# Merge the 'refactor' branch to the 'main' branch
# Making a branch ensures that the prior working version is safe on the 'main' branch while you test out new, refactored code on a new branch (called 'refactor') and merge when you're confident it works.
git switch main
git merge refactor
```

V - import

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


def parse_arguments(argv):
    """
    Parse the argument list passed from the command line
    (after the program filename is removed) and return a list
    of filenames.

    Input:
    ------
        argument list (normally sys.argv[1:])

    Returns:
    --------
        filenames: list of strings, list of files to plot
    """
    # make sure additional arguments or flags have been provide by user
    if argv == []:
        # why the program will not continue
        print("Not enough arguments have been provided")
        # how this can be corrected
        print("Usage: python gdp_plots < FILENAMES >")
        print("Options:")
        print("-a : plot all gdp datasets in the data directory")

    # check for -a flag in arguments
    if "-a" in argv:
        filenames = glob.glob("data/*gdp*.csv")
        if filenames == []:
            # file list is empty (no files found)
            print("No Files found in this folder.")
            print("Make sure the data is localed in the data folder")
    else:
        filenames = argv
    return filenames

def create_plot(filename):
    """
    Creates a plot for the specified
    data file.

    Input:
    ------
```

```
        filename: string, path to file to plot

    Returns:
    --------
        none
    """
    # read data into a pandas dataframe and transpose
    data = pandas.read_csv(filename, index_col = 'country').T
    if "continent" in data.index:
        data.drop("continent", inplace = True)

    # 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, ha = 'right')

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

def create_plots(filenames):
    """
    Takes in a list of filenames to plot
    and creates a plot for each file.

    Input:
    ------
        filenames: list of strings, list of files to plot

    Returns:
    --------
        none
    """
    for filename in filenames:
        create_plot(filename)

def main():
    """
    main function - does all the work
    """
    # parse arguments
```

```
    files_to_plot = parse_arguments(sys.argv[1:])

    # generate plots
    create_plots(files_to_plot)

if __name__ == '__main__': # if we are running from the cmd line
    # call main
    main()
```

"Untitled.ipynb" code

```
import gdp_plots
gdp_plots.create_plot("data/gapminder_gdp_oceania.csv")
```

How to see function descriptor message?

```
help(gdp_plots.create_plot)
```

Day 4

Sign-in

- Annika Pratt (she/her), Plant Pathology
- Peter Cruz Parrilla (he/him), Chemistry - Instructor
- Tracy Reuter (she/her), DAPIR - Helper
- Aadhi Balaji, Computational Biology
- Ibrahim Momohjimoh (He/him), Materials Science and Engineering, TEM data analysis.
- Jose Sanz, Mechanical Engineering, Energy harvesting from the sea
- Sven Sierra, Cereal Crop Genetics
- Sydney MCauslin, Plant Pathology
- Mari Shishido MS Library Studies
- Elizabeth Hrycyna, Entomology
- Saketh Edpuganti, DE
- Celeste Qin, Computer Science
- Trisha Adamus (she/her), Ebling Library - helper
- Kayannah Luther, Neurobiology
- Likulunga Emmanuel Likulunga, From botany working on plant-soil-microbial interactions
- Hanna Yu, MS Information
- Juan Valderrama, Nuclear Engineering
- Suihan Zhang, Biochemistry, Protein function and sequence bioinformatics

Questions from yesterday's feedback:

- What do you do if you can't figure out what a command can do from the help command? Is it easily googlable? suggestions?
    - (Sarah) The manual pages, what you see from the help command, can take a bit to learn more about how to read them, but you will get better at this with practice. You can also google most things and for a lot of the popular libraries/tools you will probably find tutorials that are good examples of how to use the comamnds. Asking a Large Language Model (LLM aka genAI) can also be helpful for this sometimes. I don't recommend relying too much on LLMs in coding generally, as you need to be able to interpret the code

and understand the organization of your code as well and LLMs can make terrible code if you don't know how to read it yourself. However, they can be useful for explaining how something is working or giving examples of how the code might look that you can play with. It is also usually free to try things! You can see what happens as you try it. With rare exceptions, you are unlikely to mess things up so trying it and see what happens can be a great way to learn. If your worried, watch out for deleting or overwriting files as that is the most common way you might mess something up that could make it harder for you. You can also come to coding meetup and get help! https://hub.datascience.wisc.edu/consultation/#dropin

- Can you transform datasets easily as in R?
    - (Sarah) I'm not quite sure what kind of transformations you are thinking about but yes, you can do pretty much the same things in python as you can in R. Since R is a statiscal programming language, it has more built in features for working with data/stats where in python you might need to load a library to do those things more easily. Both are good tools for doing this type of work. I always recommend picking one language and learning it well, then you can do pretty much anything you want in it. Howeve R and python are both high level languages (written more like human language - English - than like computer code - binary) so they are fairly similar. I picked up R pretty easily after learning several languages, including python, but since they are so close I sometimes have to look up the arguments/parameters for functions or function names to remember which one goes with which language that do roughly the same thing.
    - (Tracy) Just adding to this - If you already know R, then Python may seem familiar. The dplyr library in R is similar to the pandas library in Python. You can also use an AI code converter to translate snippets of code from R to Python or vice versa.
- Remembering every command
    - (Sarah) You don't have to remember every command. With practice, you will remember the common ones you need and you can otherwise look up what you need. Programmers don't have it all memorized, even if they code every day, they know what is possible and how to look things up or get help when they need it. See my answer to the first question today for some advice on getting help.
- We learned a lot of how to work with data but now how to make it meaningful.
    - (Sarah) So true! This workshop is more about giving you the basics of software engineering you need as as researcher, rather than trying to teach you interpretation. You as the researcher bring the expertise on your data and how to interpret it.
- Quesiton about Jupyter rearraraging the the lines if needed.
    - (Sarah) As far as I know, Jupyter can't rearrange the lines itself. It is not that "smart" of a tool. You can however run the cells in different order. This is really something to watch out for. Typically you want to write your code to run top down, but occasionally you might find yourself running cells in a different order. You then might want to reorder them so you can run it top down if you come back to it again. Please let me know if I misintepreted this question, happy to answer follow-ups.
- What is miniforge3, what is conda, and how to set an (virtual) environment?
    - conda is a package manager (and not just for python! You can install R and other tools this way too). It helps you install python (different versions of python too!) and packages/libraries. Miniforge is a version of conda that you can install to help you manage your packages. Virtual environments are where you can have multiple install setups on the same computer. So for example, you might need pandas and matplot lib for one project and for another you need a bunch of bioinformatics tools and an older version of pandas. You can keep both as separate virtual environments on your computer and then activate

one when you need it and the other when you need the other one. Otherwise you can't have two versions of pandas installed on your computer easily. Also, sometimes libraries can conflict with each other so it is good to install them seperately in different environments.

Notes

anaconda can be used for educational purposes but not research - using anaconda for research is breaking the license

For tomorrow: make sure you can run python in terminal (miniforge or anaconda for Windows users)
conda activate carpentries

Open miniforge prompt
change into the directory that has your data folder from yesterday
```
cd Desktop
cd swc
conda activate carpentries
jupyter lab
```

jupyter lab should open
make sure you are in the same folder as your data folder (you should see the data folder on the left)
make a new Notebook (click Python3 under notebook)

```
Dog = "bark"
address = 10.0
len(Dog)
```

we are going to learn lists to store multiple things

 # Lists
what are lists?
how do we use them?

```
pressures = [0.273, 0.275, 0.277, 0.275, 0.276]
print('pressures:', pressures)
print('length:', len(pressures))
```

```
# indexing lists
print('zeroth pressure in list:', pressures[0])
print('fourth pressure in list:', pressures[4])
```

```
# reassigning values
pressures[0] = 0.265
print('the pressures are now:', pressures)
```

```
# length of lists vs strings
# strings are made up of characters, len will count the characters
# lists are made up of things (can be any data type), len of a list will count those things

# appending to a list
primes[2, 3, 5]
print('primes are initially:', primes)

# adding another number to the list
primes.append(7)
print('primes are now:', primes)

help(list)

# to get rid of the help document, you can add a # at the front to comment the code and rerun the cell

teen_primes = [11, 13, 17, 19]
middle_aged_primes = [37, 41, 43, 47]

print('primes are initially:', primes)

primes.extend(teen_primes)
print('primes are now:', primes)

primes.append(middle_aged_primes)
print('primes are finally:', primes)

# append adds one thing to the list (you could end up with a list within a list)
# extend will add more (you get one clean list)

# del is a keyword for delete

del primes[8]
print(primes)

# tables are lists of lists, so we might want nested data in some cases

# empty list
empty = []
print('empty list:', empty)

goals = [1, "create lists.", 2, "extract items from lists.", 3, "modify lists."]

# comparing strings to lists
element = 'carbon'
print('zeroth char:', element[0])
print('fourth char:', element[4])

# immutable - can't be changed after creation
```

```
# mutable - can be modified

element = 'carbon'
element[0] = 'C' # this throws an error because strings are immutable and cannot be changed

print('99th element of element is:', element[99]) # throws an error because there are not 99 chars in the string

# type reassignment
# turn string into a list
list(element)

listed_element = list(element)
print(listed_element)

# to go from list to string
''.join([listed_element])
','.join([listed_element])
```

 # Challenge

Fill in the Blanks

Fill in the blanks so that the program below produces the output shown.

PYTHON

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

OUTPUT

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

Answer

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

print('second time:', values)

slicing - taking part of en entire thing (cut out one piece of the pie)

 # For loops

for number in [2,3,5]:

- print(number)

iterable is the list that the loop iterates through, in this case 2,3,5
in this example, number is the variable that changes in each iteration of the for loop
in this example, print(number) is the code block - the code you want run on the variable

# indentation is important! Needs to be consistent.
for number in [2,3,5]:

- print(number)
- print(number + 10)

# variable name can be anything, but PLEASE try to make it descriptive
for kitten in [2,3,5]:

- print(kitten)

# in python ** means 'to the power of'
primes = [2, 3, 5]
for p in primes:

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

print('a range is not a list:', range(3))

# range doesn't print like a list, but you can still use in a loop or index a range
for number in range(0, 3):

- print(number)

# accumulator pattern
total = 0
for number in range(10):

- total = total + (number + 1)

print(total) # note - if you indent this line of code, it will print the total for each iteration of the loop

# slightly different way of doing the same thing
for number in range(10):

- total += (number + 1)

print(total)

 # Conditionals

# one condition
mass = 3.54
if mass > 3.0:

- print(mass, 'is large')

mass = 2.07
if mass > 3.0:

- print (mass, 'is large')

# several conditions
grade = 85
if grade >= 90:

- print('grade is A')

elif grade >= 80:

- print('grade is B')

elif grade >= 70:

- print('grade is C')

# two conditions
velocity = 10.0
if velocity > 20.0:

- print('moving too fast')

else:

- print('adjusting velocity')
- velocity = 50.0

# conditions in a loop
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)

# we can combine conditions using boolean operators: and/or
masses = [3.54, 2.07, 9.22, 1.86, 1.71]
velocities = [10, 20, 30, 25, 20]
for i in range(5):

- if masses[i] > 5 and velocities[i] > 20:
    - print('Fast heavy object. Duck!')
- elif masses[i] > 2 and masses[i] <=5 and velocities[i] <= 20:
    - print('Normal traffic')
- elif masses[i] <= 2 and velocities[i] <= 20:
    - print('slow light object.')
- else:
    - print("Something's wrong with the data...")

# another way to write the line above
elif (masses[i] > 2 and masses[i] <= 5) and velocities[i] <= 20:
elif 2 < masses[i] <= 5 and velocities[i] <= 20:

True and True # TRUE
False and True # FALSE
False and False # FALSE
both conditions need to be true for true

 # Challenge

Tracing Execution
What does this program print?

```
pressure = 71.9
if pressure > 50.0:
   pressure = 25.0
elif pressure <= 50.0:
   pressure = 0.0
print(pressure)
```

 # Challenge
Trimming Values
Fill in the blanks so that this program creates a new list containing zeroes where the original list’s values were negative and ones where the original list’s values were positive.

```
original = [-1.5, 0.2, 0.4, 0.0, -1.3, 0.4]
result = ____
for value in original:
   if ____:
      result.append(0)
   else:
      ____
print(result)
```

OUTPUT

[0, 1, 1, 1, 0, 1]

 # Looping over datasets

import pandas as pd
for filename in ["data/gapminder_gdp_africa.csv", "data/gapminder_gdp_asia.csv"]:

- data = pd.read_csv(filename, index_col='country')
- print(filename, data.min())

for loops can take a list or a tuple
tuple - a way to store items with different data types

globbing - pattern matching (like wildcards in Unix Shell)

import glob

# print all csv files in the data directory
print('all csv files in data directory are:', glob.glob('data/*.csv'))
# how many csv files are in the data directory?
print('all csv files in data directory are:', len(glob.glob('data/*.csv')))

# this will produce an empty list because we have no .pdb files in our current working fir
print('all .pdb files:', glob.glob('*.pdb'))

for filename in glob.glob('data/gapminder*.csv'):

- data = pd.read_csv(filename)
- print(filename, data['gdpPercap_1952'].min())

 # Challenge

Determining Matches
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

remember: * matches 0 or more characters!

 # Writing Functions

# def is a keyword in python used to define functions
# parentheses are empty because our function will not take any arguments
# everything we write in the function is local to the function (you can't use the variables outside of the function)
def print_greeting():

- print("Hello!")

- print("The weather is nice today.")
- print("Right?")

print_greeting()

def print_date(year, month, day):

- joined = str(year) + '/' str(month) + '/' + str(day)
- print(joined)

print_date(1871, 3, 19)

if you have functions that you want to access in different workbooks or share with collaborators, you can make a library

# exactly the same output as above, just a different way to run the function
# either follow the parameter order (year, month, day) or explicitly write the parameter name
print_date(day=19, year=1871, month=3)

def average(values):

- if len(values) == 0:
    - return None
- return sum(values)/len(values)

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

print(average([])

functions default to None unless a return value is specified

 # Variable scopes

# pressure is a global variable, it can be used outside of the function
# temperature is local to the function, you cannot use it outside of the function
pressure = 103.9
def adjust(t):

- temperature = t * 1.43 / pressure
- return temperature

 # Programming Style

Read Here: https://swcarpentry.github.io/python-novice-gapminder/18-style.html
use assertions to check for errors
when writing functions, add docstrings to provide builtin help

def average(values):

- *"Return average of values, or None if no values are supplied."* # this line will print in the help
- **if** len(values) == 0:
    - **return** None

- **return** sum(values) / len(values)help(average)

Day 3

Sign-in

- Trisha Adamus (she/her), Ebling Library - Instructor
- Peter Cruz Parrilla, (he/him), Chemistry, - Helper
- Sarah Stevens (she/her/hers), Data Science Hub - Host/Helper
- Tracy Reuter (she/her), DAPIR - Observer
- Sydney McCauslin (she/her), Plant Pathology
- Sven Sierra, Cereal Crop Genetics
- Ibrahim Momohjimoh (He/him), Materials Science and Engineering, TEM data analysis.
- Aadhi Balaji, Computational Biology
- Celeste Qin, Computer Science
- Annika Pratt (she/her/hers), Plant Pathology
- Mari Shishido MS Library Studies
- Jose Sanz, Mechanical Engineering, Energy harvesting from the sea
- Likulunga Emmanuel Likulunga, From botany working on plant-soil-microbial interactions
- Aidan Horn (he/him) undergrad, math/genetics
- Kayannah Luther, Neurobiology
- Suihan Zhang, Biochemistry, Protein function and sequence bioinformatics
- Hanna Yu, MS Information
- Elizabeth Hrycyna, ento
- Juan Miguel Valderrama (he/him), Nuclear Engineering & Engineering Physics
- Yuetong Chang CBML

Questions from yesterday's feedback:
- Want to learn about ssh keys, tokens (authentication)

- - (Sarah) Here's a link to the GitHub docs to learn a bit more - https://docs.github.com/en/authentication

- How do we run code with different versions in the unix shell

- - (Sarah) we will learn how to run python scripts in the unix shell and version control them as we edit on Friday

- Can I drag an existing R file into the repository (on my desktop) and then edit it through terminal/nano? or only create new files?

- - (Sarah) Yes, you can edit any text file in nano on the terminal, including R scripts. You can also run R scripts from the terminal by typing `Rscript myfavscript.R` and it will run as if you are sourcing it in RStudio. You can move files with finder(mac)/file explorer(win) into a git repo and version control them.

- Can you retroactively edit commit comments?

- - (Sarah) yes! It is easiest to do if you haven't pushed it to GitHub yet and if it is the most recent commit. You can edit older commits as well but again easier if you haven't pushed those commits

to GitHub yet. If you have pushed it to GitHub you have to "force push" to amend it and that is typically discouraged so be careful with it - More detail on "ammending" a commit - https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/changing-a-commit-message

- Can you make a git repo just for racking the changes in the commit comments of another repo?

- - (Sarah) I'm not sure about this one. Mostly, I'm not sure about the use case. Why you would want to double track the commit messages? The commit messages are being stored in your git repo, why would you need or want another repo to track them too?

- Will `git merge` check all the files in both branches and see if they are all identical? How does this work for binary files?

- - Yes! When you merge branches it merges all the history/files from both branches. It will try to give you conflicts for all the files, including the binary files. Usually you would use the command line to keep just one of the binary files as it won't be possible to manually fix the conflict between them in a text editor. You will need to keep one and then edit it again.

Notes

To search for packages https://pypi.org/

Test your install:
activate the python environment
conda activate carpentries
test python from the command line (which we will need on Friday!)
python --version
test jupter lab
jupyter lab
this should launch jupter lab in your browser and will be running a bunch of code (from little web server) in your terminal
if you get something like command not found, please let us know or try to follow the setup directions for python again - https://uw-madison-datascience.github.io/2026-01-12-uwmadison-swc/#python

IDE - integrative development environment (Jupyter Notebook)
can do python in command line instead of using IDE

- # Jupyter makes it more accessible and easy to learn

anaconda free for educational purposes but not research purposes
Data Download: https://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip

open miniforge/anaconda prompt (win) or terminal (mac)
in the prompt type:
conda activate carpentries
jupyter lab

a web browser should open for you

alternative: search for anaconda navigator and click on the app (it will take some time to open)
click "launch" on Jupyter Notebook app
you should reach the landing page with files on the left and a tab called Launcher on the right

 # In Jupyter Notebook

create a new notebook (click Python 3 under the Notebook heading in the Launcher tab) and title 2026-01-14-swcpythonday1.ipynb
running a cell - shift + enter or press the run button

- # cmd + enter for macs (macs can also use shift+return)

markdown is text, code is for code
Esc + m changes a cell to markdown
pound symbol can change font size (different headers)

Challenge - python as a calculator
1 Type 7*3 into a cell and then shift return
2 Type 2+1 into a cella nd then shift return
#3 Type 7*3 into a cell and then type 2+1 into the same cell and shift return?
What output do you get and why?

7*3
prints 21 as output
2 + 1 # spaces don't matter!
prints 3 as output
7*3
2+1
prints only the 3 as the output! - jupyter lab only prints the output from the last thing executed (it still ran the other code but didn't print the output to the screen

 # Variables and Assignment

use meaningful names
single quotes indicates a string of text, double quotes work too, just be consistent

```
age = 42
first_name = 'Ahmed'
```

print function (after typing the next line, press shift + enter to run)

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

order matters, you can reassign a variable
if you have a variable called name and you give it the value 'Annika', then later you change it to 'Trisha' it will change
be careful reassigning variables in scripts, every instance of the name variable between name = 'Annika' and name = 'Trisha' will be Annika. Once the name = 'Trisha' code is run, all instances of the variable name will be Trisha

```
# characters are numbers starting with 0
atom_name = 'helium'
print(atom_name[0]) # prints 'h' because this is the first letter of the atom name variable
# indexing starts at 0 in python, so the first character is 0, the 2nd character is 1, etc..

# slicing
# 0:3 in python means 'up to but not including the third character'
print(atom_name[0:3])# prints hel

# length (len) function - returns the length of whatever you give it (can be a variable or a string)
len('helium')

len(atom_name)

# Challenge
# what is the final value in the program below?
initial = 'left'
position = initial
initial = 'right'
print(position)

# if you assign a = 123, what happend if you try to get the second digit of a[1]?
a = 123
print(a[1]) # this prints an error because a is a number (integer)

# to fix it, you have to change a from an integer to a string
a = str(123)
print(a[1])

# Data Types and Type Conversion

# every value has a type
# integer
print(type(52))

# float - any number with a decimal point
print(type(3.14159))

# string
fitness = 'average'
print(type(fitness))

print(5-3) # this works, you can subtract integers

print('hello' - 'h') # this throws an error, you can't subtract strings

# you can use + and * on strings
full_name = 'Ahmed' + 'Walsh'
print(full_name)
```

```
full_name = 'Ahmed' + ' ' + 'Walsh'

dog = 'Bark'*10
print(dog)

# can't use len function on an integer
print(len(52)) # throws an error

# can't combine types - you must convert one so they are the same type
print(1 + "2")

# convert a string to integer
print(1 + int('2'))

# convert an integer to string
print(str(1) + '2')

# can combine integer and float
print(1/2.0)
print(3.0**2) # ** is to the power of

# challenge
# What type of value is 3.4? How can you find out?
It's a float!
type(3.4)
print(type(3.4))

# a function can have zero or more arguments
print('before')
print()
print('after')

# can't assign print statement to variable
results = print('example')
print('result of print is', result)

# min max function
print(max(1, 2, 3))

print(min('a', 'A', '0'))

# order characters, 0-9, A-Z, a-z
round(3.14159) # round to the integer

round(3.14159, 1) # round to one decimal place

# access the help documentation for this command
help(round)
```

# method
my_string = 'Hello World!'
print(len(my_string))
print(my_string.swapcase()) # changes the case of every letter
print(my_string.isupper()) # FALSE because not all letters are uppercase
print(my_string.upper().isupper()) # change to uppercase then ask if it's uppercase - returns TRUE

 # 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))
print(max(len(rich), len(poor)))

 # Libraries

a library is a collection of files (called modules) that contain functions for use
need to use import to lead a library
syntax: module_name.thing_name
. means 'part of'
some libraries come with 'base python', essentially pre-installed, but there are a ton of other libraries that you can install yourself if you read about a particular function that you want to use!

import math

print('pi is', math.pi)

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

help(math) # tells you about the functions within the math library that you can use

# import only specific items from a library
# instead of importing the entire library, it only imports the cos and pi functions (saves space/time)
from math import cos, pi
print('cos(pi) is', cos(pi))

 # Reading Data into DataFrames

on the left in Jupyter Notebook, click on your data folder (so you see all the gapminder files on the left)
create a new notebook (click on blue button with + sign, click on Python3 under Notebook)
your untitled notebook pops up, right click and rename (2026-01-14_gapminder.ipynb)

In your new noteboook, add the following information. Best practice is to add your name, email, date,

and a description of the notebook whenever you start a new one

```
# Name
# email
# date YYYY-MM-DD
# Pandas using Gapminder data

# Reading Tabular Data into Dataframes
# tabular data is essentially an excel table (it has rows and columns)
# pandas for tabular data

# import pandas library and give it an alias
# now every time we type pd, it will reference the pandas library
import pandas as pd

data_oceania = pd.read_csv('gapminder_gdp_oceania.csv')

# outputs the data
print(data_oceania)

# pd is the library
# read_csv is the method within that library
# make country the row name with index_col = country
data_oceania_country = pd.read_csv('gapminder_gdp_oceania.csv', index_col = 'country')
print(data_oceania_country)

# gives us information about our data
data_oceania_country.info()

# transpose a dataframe - flip rows and columns
# this doesn't change our data in any way, it just prints out the transposed version
print(data_oceania_country.T)

print(data_oceania_country.describe())

# find data by location
# best practice is to give your dataframes more descriptive names than data (we're just doing this today to make things less complicated)
# iloc stands for integer location
import pandas as pd
data = pd.read_csv('gapminder_gdp_europe.csv', index_col = 'country')
print(data.iloc[0,0])

print(data)

# print just 1952 gdp column for Albania
print(data.loc['Albania', 'gdpPercap_1952'])

# print all columns for Albania
```

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

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

# even without the colon, it knows to print all columns for Albania
print(data.loc['Albania'])

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

# use a subset of data to keep output readable
# \n creates a new line or line break within a string
subset = data.loc['Italy' : 'Poland', 'gdpPercap_1962' : 'gdpPercap_1972']
print('Subset of data:\n', subset)

# What values were greater than 10000?
print('\nWhere are values large?\n', subset > 10000)

# Select values or NaN using a Boolean mask
# NaN - not a number
# show the values that are above 10000 and make the rest null
mask = subset > 10000
print(subset[mask])
```

 # Plotting

```
# matplotlib commonly used in Python
#Jupyter notebooks will render plots inline by default
# to make bulletpoints in a markdown cell, use *

import matplotlib.pyplot as plt

# create a simple plot
time = [0, 1, 2, 3]
position = [0, 100, 200, 300]
plt.plot(time, position)

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

# plot data from a dataFrame
import pandas as pd
data = pd.read_csv('data/gapminder_gdp_oceania.csv', index_col='country')

# Extract year from last 4 characters of each column name
# replace() removes the characters from the string
years = data.columns.str.replace('gdpPercap_', '')
years
```

```
# convert year to integers (so you can do math with them)
data.columns = years.astype(int)
data.columns

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

# dataframe.plot default is row as the x axis
# use the data.T to transform the data and plot multiple countries in a series
data.T.plot()
plt.ylabel('GDP per capita')

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

# plot many sets of data together
# select two countries worth of data
gdp_australia = data.loc['Australia']
gdp_nz = data.loc['New Zealand']

# plot with differently colored marker
plt.plot(years, gdp_australia, 'b-', label = 'Australia')
plt.plot(years, gdp_nz, 'g-', label = 'New Zealand')

# create legend - look at the help to see these different options
plt.legend(loc='upper left')
plt.xlabel('Year')
plt.ylabel('GDP per capita')

# save your plot - in future, give it a more descriptive name than my-figure
plt.gcf().savefig('my_figure.png')

# closing things for the day
# File > Save (there are a lot of save options)
# File > Shutdown
```

Day 2

Sign-in

- Sarah Stevens (she/her/hers), Data Science Hub - instructor
- Annika Pratt (she/her/hers), Plant Pathology
- Sydney McCauslin (Plant Pathology)
- Peter Cruz Parrilla (he/him), Chemistry - Helper
- Sven Sierra, Cereal Crop Genetics
- Likulunga Emmanuel Likulunga (he/him), Botany, visiting scholar: Plant-soil-microbial

interactions
- Jose Sanz, Mechanical Engineering, Energy harvesting from the sea
- Saketh Edpuganti, Data Engineering
- Dante Baker, zoology
- [helper] Aaron Tran, physics
- Aadhi Balaji, Computational Biology
- Kayannah Luther, Neurobiology
- Mari Shishido MS Library Studies
- Trisha Adamus (she/her), Ebling Library - helper
- Hanna Yu, MS Information
- Ibrahim Momohjimoh (He/him), Materials Science and Engineering, TEM DATA analysis.
- Suihan Zhang (he/him), Biochemistry, Protein function and sequence bioinformatics
- Aidan Horn (he/him) undergrad, math/genetics
- Yuetong chang cbml
- Elizabeth Hrycyna (she/her) Entomology master's, sustainable pest management
- Juan Miguel Valderrama (he/him), Nuclear Engineering & Engineering Physics

Questions from yesterday's feedback:
message from Annika: we ran through pipes, loops, and shell scripting really quickly (sorry!) you can always go back to the lesson and work through those examples at a slower pace whenever you want! https://swcarpentry.github.io/shell-novice/04-pipefilter.html

- List of commands we learned: ls, pwd, cd, cat, head, tail, mkdir, rmdir, rm, nano, cp, mv, wc, history, grep, find
- When you open a new terminal, you will always be in your home directory (you can change what your home directory is if you want it to be different than the default) so you won't pick up exactly where you left off, but your commands from previous sessions are saved in "history" so it isn't exactly a clean slate either.
- loops and pipes are different
    - pipe:
        - denoted with a |
        - used for stringing commands together
        - the output of one command becomes the input for the next command
        - sort filename.txt | head | tail
            - will sort the file, then grab the first few lines of the sorted output, then grab the last few lines of the head output
    - loop:
        - for thing in list-of-things; do commands; done
        - used when you have a list of things (ex. 20 files) and you want to run the same command on all of them

Day 2

Notes:

open a terminal and type git --version to check your installation for today!
git --version
For those who created their SSH key without a passphrase (just pressed Enter), and would like to change it later, you can run the following command:

- ssh-keygen -p -P "" -N "YOUR_NEW_PASSPHRASE" -f ~/.ssh/PATH_TO_KEY_FILE
    - The -p switch means you want to change your passphrase. The -P argument takes your old passphrase. The -N argument takes your new argument. This is also a way for you to create a passphrase and be able to see what you type.
    - example: ssh-keygen -p -P "" -N "test" -f ~/.ssh/id_ed25519

version control for JupyterLab
https://marimo.io/features/vs-jupyter-alternative

 # git vs. GitHub

git - the command line software that keeps track of versions
GitHub - online to keep track of folders (repositories) and share with collaborators
GitHub Desktop, VSCode, RStudio - GUIs that use git in a user-friendly way
programming languages let you do more stuff than the graphical user interface (GUI)

 # What is version control?

keeps track of changes made to files
can track things that multiple users do (collaborate on code)
sync files at multiple locations (your laptop and lab computer, for example)
publish your code - many journals are requiring this now
international standard date: yyyy-mm-dd
using version control keeps things more organized and makes it easier to go back to old versions/see the difference between new and older versions
you can contribute to packages on GitHub

 # Setup

open your terminal
this is stuff you only have to do once in git
if you've used git before, it's okay to do this again
git config --global user.name "Annika Pratt"

we're giving this info so git knows who's making changes to files
git config --global user.email "netid@wisc.edu"
git config --global color.ui "auto"

if you see warnings about line endings later, let Sarah know!
git config --global core.editor "nano -w"
git config --list
git config --global init.defaultBranch main

side note: your prompt may be a $ or %, either is fine for today!
$ is bash, % is z shell

 # ssh key setup

```
ls -al ~/.ssh
```

```
ssh-keygen -t ed25519 -C "email"
```

you should add a passphrase, but you don't have to
if you make a passphrase, you have to type this every time you interact with git
if you are making a new key with a different name, type the whole thing that's in parentheses, but add to make a new name: Ex. /Users/username/.ssh/id_ed25519_swc
you should see some random art created

you can directly download software from gitHub, but downloading only gives you the most recent version of the software, not history

print the key
use tab complete to confirm that the file exists

```
cat ~/.ssh/id_ed25519.pub
```

copy everything from ssh-ed25519 through your email
might need to right click to copy instead of using ^C

Go to GitHub
click on profile picture > settings > SSH and GPG keys > add new SSH Key
give a title that differentiates this computer from others you might use "personal laptop"
paste the key that you copied into the "key" box
click green "add ssh key" button

```
ssh -T git@github.com
```

you'll get a question about if you really want to connect - type "yes" and hit "enter"

 # Creating a Repository

know which directory you are making this in (home is recommended)
make new folder called planets

```
mkdir planets
```

move into the planets folder

```
cd planets
```

initialize git repository
if you type ls -a before running this, there should not be a .git file
if you type ls -a after running this, there should be a .git file
after running this command, git will try and track all files in this folder

```
git init
```

what is the status of the files in our git repository
a commit is a snapshot in history (what is the status of our files at one point in time?)

```
git status
```

 # Tracking Changes

```
# use nano to create a new file
nano mars.txt

# in the nano file
Cold and dry, but everything is my favorite color

# exit nano by typing cntrl X
# press Y to save changes
# press enter or return to save the file

git status

# tell git to start tracking our mars.txt file
git add mars.txt

# now there are changes to commit because we added mars.txt
git status

# give a message for every commit; the more informative the better
git commit -m "started notes on Mars as a base"

# should tell us that everything is clean (there is nothing to add or commit)
git status

# look at history
git log

nano mars.txt

# type this in nano
The two moons may be a problem for Wolfman

git status

# shows the difference between versions of files
git diff

# if you don't add before committing, you will see the git status output (git trying to tell you there's nothing to commit)
git add mars.txt
git commit -m "Add concerns about effects of Mars' moons on Wolfman"

# adding multiple files
# you can use "git add [filename] [filename] [filename]", or "git add -u" to get all untracked changes
# be careful using the -u flag, you might end up adding things that you didn't mean to track
```

see shorter version of log
git log -oneline

add to mars.txt in nano
But the Mummy will appreciate the lack of humidity

git status

git add mars.txt

see difference between version that's been added and version that has been committed
git diff --staged

if you don't use the -m flag to add a message, a text editor will pop up and ask you to write a message
git commit -m "Discuss concerns about Mars' climate for Mummy"

if you don't set your default text editor to nano...
hit "escape", then type ":wq" to get out of the text editor (vim)

 # Challenges

Which of the following commit messages would be most appropriate for the last commit made to mars.txt?

1. “Changes”
2. “Added line ‘But the Mummy will appreciate the lack of humidity’ to mars.txt”
3. “Discuss effects of Mars’ climate on the Mummy”

Which command(s) below would save the changes of myfile.txt to my local Git repository?

1. $ git commit -m "my recent changes"
2. $ git init myfile.txt $ git commit -m "my recent changes"
3. $ git add myfile.txt $ git commit -m "my recent changes"
4. $ git commit -m myfile.txt "my recent changes"

a commit is one point in history, you only have one message
make sure to commit whenever you are done with one set of changes
it is really nice if you have multiple files that are made/changed for the same project and you can write one commit message for all of those changes
try not to have one commit for multiple unrelated files
get in the habit of committing often, some programmers get into a bad habit of not committing enough and running into a bunch of issues

 # Exploring History

add to mars.txt
An ill-considered change

```
git status
git diff

# make sure you write very clear message if you're committing broken code

# we want to go back to an old version
# HEAD refers to the most recent commit
git diff HEAD mars.txt

# one version before the most recent
git diff HEAD~1 mars.txt

# two versions before the most recent
git diff HEAD~2 mars.txt

git log --oneline

# ac1a3bb is the unique idetifier for the commit that you want to compare with
# you can get the unique identifier by looking at git log
git diff ac1a3bb mars.txt

# restore to the committed version (an ill-considered change is erased)
git restore mars.txt

# going back to an older version
git resotre -s <unique-identifier-of-commit> mars.txt
vn
git restore mars.txt
```

 # Challenge

Jennifer has made changes to the Python script that she has been working on for weeks, and the modifications she made this morning “broke” the script and it no longer runs. She has spent ~ 1hr trying to fix it, with no luck…
Luckily, she has been keeping track of her project’s versions using Git! Which commands below will let her recover the last committed version of her Python script called data_cruncher.py?

1. `$ git restore`
2. `$ git restore data_cruncher.py`
3. `$ git restore -s HEAD~1 data_cruncher.py`
4. `$ git restore -s <unique ID of last commit> data_cruncher.py`
5. Both 2 and 4

 # Ignoring Things

we are skipping this lesson today, but you can watch this video or follow along in the carpentries lesson linked below
https://carpentries-incubator.github.io/git-novice-branch-pr/instructor/06-ignore.html

Youtube video: https://youtu.be/UR6XcANjbac?si=Yi_zCdRvDLY3IPH6
Full playlist: https://www.youtube.com/playlist?list=PL90vX4e9ORqeE1wyMbST9pJ1-Mj45f2JY

 # Branches

split our history, collaborate with others
example use: your collaborators are using your software and you want to make sure that code stays stable on the main branch. You want to test new code, so you create a new branch to test out the new software and make sure it works before opening it up to your collaborators
you can name these branches whatever you want
a common branch name is "dev" where programmers are developing new stuff

see a list of your branches
```
git branch
```

make a new branch
```
git branch pydev
```

tells us which branch we're on
```
git status
```

switch to a different branch
```
git switch pydev
```

also tells us which branch we are in
```
git log --oneline
```

ALWAYS check which branch you are on before committing
use git branch, git status, or git log to double check

make a blank file
```
touch analysis.py
```

add and commit new file to pydev branch
```
git add analysis.py
git commit -m "wrote and tested python analysis script"
```

switch to the main branch
```
git switch main
```

create a branch and switch to it
```
git switch -c bashdev
```

```
git add analysis .sh
git commit -m "Wrote and tested bash analysis script"
```

a nicer way to see the log
```
git log --oneline --all --graph --decorate
```

```
git switch main

# merging branches
git merge pydev

# delete pydev branch
git branch -d pydev
```

 # Conflicts

```
# whoever gets the conflict has to slove it
# 2 branches have different versions of the same file

# create new branch
git branch marsTemp

nano mars.txt

# in nano
I'll be able to get 40 extra minutes of beauty rest

# make sure I am committing this to the main branch
git status

# if you are only committing one file that you've committed before, you can add and commit at the same time
git commit -m "Added line about daylight on mars" mars.txt

git swtich marsTemp

nano mars.txt

# in nano
Yeti will appreciate the cold

git add mars.txt
git commit -m "Added  line about the temp on Mars"

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

git switch main
git branch

# this will produce a conflict
git merge marsTemp

# take a look at the conflict, fix it the way you want, save, add, and commit
nano mars.txt
```

```
git add mars.txt
git commit -m "merged changes from marsTemp"

git log --oneline
```

 # GitHub

go to github.com
create a new repository
give it a name (ex. planets)
nice if name of repository on GitHub matches what's on your computer
give it a description
choose whether it's public or private
no template, readme, gitignore, or license - only use these options if you are creating a repository from scratch on GitHub (we already have a repository on our local computer)
make sure ssh is clicked on the Quick setup
copy the address next to ssh

- # git@github.com:username/planets

go back to your terminal
remote is often called origin
this sets up the connection between your computer and github

```
git remote add origin <paste address from github>
git remote -v
```

now we have to sync between github and our computer
pushing the main branch to github from our local computer

```
git push origin main
```

there's a lot of stuff you can do on GitHub to edit files, etc.
Sarah added to the `analysis.py` script the line `import pandas as pd`

pull branch from github to your local computer

```
git pull origin main
```

Day 1

Sign-in (Name, Dept, Describe your research/work in a couple of sentences)

- Annika Pratt (she/her/hers), Plant Pathology, I use computational tools to explore the roles of transposable elements in plant fungal infections
- Ryan Bemowski (he/him), Data Science Hub
- Trisha Adamus (she/her), Ebling Library - helper
- Ibrahim Momohjimoh (He/him), Materials Science and Engineering, Diffraction data from TEM imaging
- Sydney McCauslin (She/Her), Plant Pathology, Fungal/Bacterial interaction for natural product

discovery
- Saketh Edpuganti, Data Engineering, professional masters program, developing workflows for automating data management
- [helper] Aaron Tran, Dept. Physics, plasma physics simulations on high-performance parallel computers
- Peter Cruz Parrilla (he/him), Chemistry, helper, bioinformatics and quantitative proteomics
- Aadhitya Balaji, Computational Biology and Machine Learning, Developing ML models to solve biological problems
- Kayannah Luther, (she/her, undergrad), Neurobiology, Research: Speech Processing in Stuttering Adults
- Yuetong Chang Computational Biology and Machine Learning, topological ML
- Sven Sierra, Plant Breeding and Plant Genetics, Cereal crop genetics and bioinformatics
- Dante Baker, zoology, just prepping for independent research this summer in genetics.
- Juan Miguel Valderrama (he/him), Nuclear Engineering & Engineering Physics, researching fusion energy technologies
- Mari Shishido, MS Library Information Studies
- Rosie Bae, DAPIR, Institutional Policy Analyst
- Erwin Lares - he/him - Research Cyberinfrastructure
- Hanna Yu, MS Information
- Chloe Oh - Psychology (Data Science in Human Behavior)
- Josh Wright - Counseling Psychology and Center for Healthy Minds, researching psychology of meditation & mindfulness
- Elizabeth Hrycyna she/her - Entomology, sustainable pest management
- Sely Pena-Rivera - she/they, Bacteriology, Observer
- Aidan Horn - Math/Genetics, Undergrad
- Suihan Zhang (he/him) - Biochemistry, protein function and sequence bioinformatics
- Jose Sanz, Mechanical Engineering, Energy harvesting from the sea
- Likulunga Emmanuel Likulunga (he/him), Botany, visiting scholar from Zambia working on: plant-soil-microbial interactions

Notes:

- “don’t invoke the name of a folder you’re already in”
- press the 'tab' key to autofill
- go slow and be mindful when deleting folders recursively (rm -r)
- find and ls -R show the same stuff, just in a different way (personal preference)

Challenge!

Exploring More ls Options

You can also use two options at the same time. What does the command ls do when used with the -l option? What about if you use both the -l and the -h option?
Some of its output is about properties that we do not cover in this lesson (such as file permissions and ownership), but the rest should be useful nevertheless.

paste your answer in chat - what does ls -l and ls -l -h do?

challenge!

Absolute vs Relative Paths

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

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

Creating Files a Different Way

We have seen how to create text files using the nano editor. Now, try the following command:

BASH

$ touch my_file.txt

1. What did the touch command do? When you look at your current directory using the GUI file explorer, does the file show up?
2. Use ls -l to inspect the files. How large is my_file.txt?
3. When might you want to create a file this way?

Moving Files to a new folder

After running the following commands, Jamie realizes that she put the files sucrose.dat and maltose.dat into the wrong folder. The files should have been placed in the raw folder.

BASH

$** ls -F **analyzed/** raw/**$ ls -F analyzed**fructose.dat** glucose.dat maltose.dat sucrose.dat**$** cd analyzed

Fill in the blanks to move these files to the raw/ folder (i.e. the one she forgot to put them in)

BASH

```
$ mv sucrose.dat maltose.dat ___/___
```

List filenames matching a pattern

When run in the alkanes 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.

What Does >> Mean?

We have seen the use of >, but there is a similar operator >> which works slightly differently. We'll learn about the differences between these two operators by printing some strings. We can use the echo command to print strings e.g.

BASH

```
$ echo The echo command prints text
```

OUTPUT

```
The echo command prints text
```

Now test the commands below to reveal the difference between the two operators:

BASH

```
$ echo hello > testfile01.txt
```

and:

BASH

```
$ echo hello >> testfile02.txt
```

Hint: Try executing each command twice in a row and then examining the output files.