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

---

Please scroll to the end for fresh notes.

Links & Resources

Welcome to the 2023-01-25 Carpentries Workshop at MIT. We will use this Etherpad for discussion, shared note-taking, quick practice questions, and anything else that comes up.

General Setup & Tasks

- Workshop website: https://carpentries-mit.github.io/2023-01-25-mit-online/
- Pre-Workshop Survey: (Please take the time and fill this out if you haven't already done so): https://libcal.mit.edu/calendar/events/carpentries2023-01-25.
- Welcome/Setup slides: http://bit.ly/SCW_MIT_setup
- Please fill in your name in the upper right corner of the Etherpad (next to the color square) and then **sign in under the Attendee section and tell us a bit about yourself.**
- **No question is stupid**. Use the Chat window to the right to ask questions anytime.
- Zoom Connection details:
    - Join through the previously-installed Zoom client application on your computer instead of using the link to 'join from your browser’
    - Find the Zoom connection details in your registration confirmation email
    - type your chosen name, pronouns, and/or pronunciation guide into the participant window
    - keep your video on if bandwidth & situation allows
    - wear headphones (if you have them) to hear better
    - please mute yourself to start

Lesson materials

- SetUp Slides: http://bit.ly/SCW_MIT_setup
- Unix Shell Lesson: https://swcarpentry.github.io/shell-novice/

- Unix Shell Slides: http://bit.ly/SCW_MIT_unix
- Unix Shell Data: http://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip
- Python Lesson: http://swcarpentry.github.io/python-novice-gapminder/
- Python Slides: Slide 34 - 38 of http://bit.ly/SCW_MIT_setup
- Python data download http://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip
- Git/GitHub Lesson: https://carpentries-mit.github.io/githublesson/notebooks.html
- Git/GitHub slides: slide 41 - 42 in http://bit.ly/SCW_MIT_setup
- Open Science Slides: https://docs.google.com/presentation/d/1hY-T2fq2v8IxJZMIBCV9K37S4HcuwdUgcrOjDp8R-q0/edit?usp=sharing

Feedback links

- **Daily feedback**. At the end of each day, use this Jamboard to provide feedback on the day's session: https://bit.ly/dayfeedback20230125
- **Post-workshop Survey.** Please take the time to fill out the post-workshop survey at the end of all classes (after Jan 27): https://carpentries.typeform.com/to/UgVdRQ?slug=2023-01-25-mit-online

Attendees

Add your Name, Department/Affiliation, email, twitter, github username or anything you'd like to share about yourself with the group.

- Ye Li, MIT Libraries, yel@mit.edu, @yelibrarian, GitHub: YeLibrarian
- Ece Turnator, MIT Libraries, turnator@mit.edu, Github: Eturnator
- Mark Szarko, MIT Libraries, szarko@mit.eu, Github: mszarko
- Daniel Sheehan, MIT Libraries, dsheehan@mit.edu
- Christine Malinowski, MIT Libraries, cmalin@mit.edu, Github: cmalinowski3
- Harry Halpin, Connection Science, halpin@mit.edu Github: hhalpin
- André Piza, Research Project Manager, Alan Turing Institute (London, UK), @andrepiza, Github: andrepiza
- Sudhansu Mishra, IC Design engineer based in Edinburgh UK, Githubhttps://github.com/Sud-ana
- Meral Hacikamiloglu, Research Assistant at McGovern, meral@mit.edu, meralha
- Margaret Schroeder, Brain and Cognitive Sciences PhD stduent, mschro@mit.edu, Github: schroeme
- Léllé Demertzi, Project Administrator - Living with Machines, The Alan Turing Institute, Github: LelleDemertzi
- Bennett Capozzi, Urban Studies and Planning Masters Student, MIT, Github: bennettcapozzi
- Torsten John, Biological Engineering, MIT, tjohn@mit.edu, @torsten_john, GitHub: chemist09
- Shrooq Alsenan, Postdoctoral Fellow in Jameel Clinic, alsenans@mit.edu
- Yihao Zhang, MIT MECHE, yihaozh@mit.edu, github: bluekaze
- Nicholas King, Chemical Engineering PhD student, gxking@mit.edu, GitHub: nickking9450
- Cara-Beth Lillback, lillback@alumni.upenn.edu, GitHub: carabel https://www.linkedin.com/in/cara-beth/
- Elyse Oliver, DUSP Master in City Planning (MCP) student, eoliver@mit.edu, GitHub: eolives
- Kate Koch, Biology graduate student, MIT, GitHub: kkoch1129

- Cheng-Hsin Chan, MIT Architecture SMArchS Urbanism, chanch@mit.edu, GitHub: Chengmit
- Wiebke Hengst, Stuttgart University Library, OER Cordinator / Student: M.A. E-Education FernUniversität in Hagen. wiebke.hengst@posteo.de, Github: WiebkeHeHub
- Vineel Teja Kommu, Masters student at University of Potsdam, Germany, kommu@uni-potsdam.de
- Cian Schmitt-Ulms, Brain and Cognitive Sciences PhD Student, cian@mit.edu, Github: CianSchmittUlms
- Ruijen Yang, Harvard Graduate School of Design MArch2, ruijenya@mit.edu, Github:ruijenyang
- Juana De La O, MIT Biology PhD, jdelao@mit.edu, Github: Juana-DeLaO
- Gabe Winter, Population Ecology, Friedrich-Schiller University Jena, Germany, gabe.winter@uni-jena.de, tt @GbWinter, github: gabewinter,
- Pooja Ayachit, Graduate Student at the Department of Epidemiology, University of Washington, payach12@uw.edu, Github: payach12
- Amanda Guan, Accountant, ag.amandaguan@gmail.com, Github:amandaguan-ag
- Ana Mendes, Visual Artist, Goldsmiths College, ana@anamendes.com, GitHub: anamendesana
- Lydia Harrington, Post-Doctoral Fellow at MIT's Aga Khan Program for Islamic Architecture, lharrin@mit.edu
- Sonia Liou, IR MIT, sliou@mit.edu, Github: sliou

Share your GitHub username here if you haven't seen an invitation to join GitHub repo from Ye.

@vineelkommu
@cecqw -Cigdem's

Notes:

Day 1:

- Unix Shell Slides: http://bit.ly/SCW_MIT_unix
- Unix Shell Lesson: https://swcarpentry.github.io/shell-novice/
- Difference between Programming Lang and Scripting Languages: https://www.geeksforgeeks.org/whats-the-difference-between-scripting-and-programming-languages/ (programing languages are closer to human languages, scripting langs farther but can be more powerful).

- Unix Shell Data: http://swcarpentry.github.io/shell-novice/data/shell-lesson-data.zip

- Summary and Glossary for this lesson https://swcarpentry.github.io/shell-novice/reference

Bash Options: Info on all the many bash prompt options: https://phoenixnap.com/kb/change-bash-

prompt-linux

Command to change how the pointer look like on Terminal
export PS1="\w \u$ "

pwd --> print present working directory (shows you where you are in the computer)
cd --> change directory

ls --> list the folders and files in pwd
Autocomplete with tab command - does this only
ls -F --> differentiate folder with a slash
ls -R --> list all folders in all subdirectories

mkdir --> make a directory/folder
mkdir -p --> make nested directories (subdirectories)

Use up arrows to retrieve previous commands
.. --> go back up one folder level --> Allows you to see files that are stored in folders that you are not currently in, which helpful if you want to refer to other data files without moving around

mkdir --> make a new directory, specify name as argument
mkdir -p dir1/dir2/dir3 --> make nested directories
ls -R dir1 --> list all directories within top directory
Guide for naming conventions: https://libraries.mit.edu/data-management/store/organize/

clear --> clears terminal

how to create a text file?
1. use a text editor (like nano)
- e.g. nano draft.txt
2. touch
- e.g. touch myfile.txt
- advantage over text editor? can incorporate that into a longer script

how to move files?
- mv [source] [destination] --> moves a file (can also change the name)
- if destination is "." will move file into current directory
- cp [source] [destination] --> makes a copy of the file
- cp -r --> recursive copy, will copy whole directory

Exercise:
EXERCISE: 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 .

how to remove a file?
rm --> deletes files
*delete is forever (no trash or recycle)
rm -r --> deletes directory
-r stands for recursively
rm -i --> asks for a confirmation before deleting

wc --> word count
1. number of lines
2. number of words
3. number of characters
-l --> number of lines
-w --> number of words
-m --> # characters

cat --> concatenate (prints on screen)
less --> for displaying large files

to get out of a loop: ctrl+c or q

sort --> sort numerically or alphanumericallysort
sort -n --> sorts numerically

head --> from beginning
tail --> sorts from end

Post related to navigate to a different drive or networked drive in Unix Shell:
https://askubuntu.com/questions/1386402/how-to-change-directories-to-a-different-hard-drive-using-ubuntu-on-wsl
https://superuser.com/questions/1128634/how-to-access-mounted-network-drive-on-windows-linux-subsystem

For Loops:

Syntax is like so:

```
for thing in list_of_things
do
   operation_using $thing
done
```

< thing > above is a variable name, it doesn't matter what you use to Shell, but for readability purposes give your variables meaningful names.

In order to use the autocomplete option type a few characters of the directory name or filenname and hit < TAB >

Q: is there a reason for no prompt after done in the loop and no return results ?
A; it might be stucking in the loop. Try press q to see if it helps.

In the shell-lesson-data/exercise-data/proteins directory, what is the effect of this loop?

```
for alkanes in *.pdb
do
    echo $alkanes
    cat $alkanes > alkanes.pdb
done
```

Use up and down arrows to navigate between commands

> write to file
>> append to file (if exists)

*don't use spaces in filenames or folder names

We can do loops inside of loops.

to make scripts from what you've done
history | tail -n 10 --> returns last 10 lines

history command gives you list of commands used.

shell scripts end in .sh

And to RUN those shell scripts use the bash command.

bash DirectoryLocation/file.sh

grep --> globally search for regularbas expression and print matching lines

Absolute path is from the Root directory . Explained here: https://swcarpentry.github.io/shell-novice/02-filedir/index.html

Day 2

- Python Slides: Slide 34 - 38 of http://bit.ly/SCW_MIT_setup
- Python Lesson: http://swcarpentry.github.io/python-novice-gapminder/
- Python data download http://swcarpentry.github.io/python-novice-gapminder/files/python-novice-

gapminder-data.zip

- Start Anaconda Navigator
    - Mac: Launchpad > Anaconda-Navigator
    - Windows: Start > Anaconda Navigator
    - Linux: open terminal and type anaconda-navigator
- Start Jupyter Lab on the Anaconda Navigator window in your browser

Shift + Enter or Shift + Return --> run the code in the cell

Cell type: Use dropdown menu from the top or use the shortcut below to change the cell type
Code : Esc, then press y
Markdown: Esc, then press m

- A way to write, read, and edit text on the web. Similar syntax to html but more limited than html.

Shortcut

navigating jupyter notebook cells:
esc: escape current cell
a: add a cell above
b: add a cell below
x or dd: deletes highlighted cell
z: undo last operation
esc+m: switch to markdown
esc+y: switch to code mode

Markdown cheatsheets (add your own!):
https://www.markdownguide.org/cheat-sheet/
https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Make different levels of headings
#Heading 1
##Heading 2
###Heading 3

While on markdown mode
1. -> numbered list
* -> bullet list

Two ways to create links

[Create links](https://carpentries.org) with '[...](...)'.

Or use [named links][carpentries_website]

[carpentries_website]: https://carpentries.org

https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/Giraffe_Mikumi_National_Park.jpg/800px-Giraffe_Mikumi_National_Park.jpg
![AmimalPic](https://upload.wikimedia.org/wikipedia/commons/thumb/9/9e/Giraffe_Mikumi_National_Park.jpg/800px-Giraffe_Mikumi_National_Park.jpg)

link to email:
yel@mit.edu

- A variable name must start with a letter or the underscore character
- No SPACES in variable names
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Strings are enclosed in quotes. Can be single or double quotes, but be consistent with what you use.

In Python, Indices are numbered from 0.

for example,
element[0] will return the first char of the element string variable
element[-1] will return the last char

To return a slice:
element[0:3] or variableName[start:end]

EXAMPLE: Fill the table showing the values of the variables in this program after each statement is executed.
element

# Command	# Value of x	# Value of y	# Value of swap #
x = 1.0	1.0	#	# #
y = 3.0	#	#	# #
swap = x	#	#	# #
x = y	#	#	# #
y = swap	#	#	# #

Solution:

# Command	# Value of x	# Value of y	# Value of swap #
x = 1.0	# 1.0	# not defined	# not defined #
y = 3.0	# 1.0	# 3.0	# not defined #
swap = x	# 1.0	# 3.0	# 1.0 #
x = y	# 3.0	# 3.0	# 1.0 #
y = swap	# 3.0	# 1.0	# 1.0 #

EXAMPLE: Given the following string:

!! Do any 3 of the below:

species_name = "Acacia buxifolia"
What would these expressions return?

species_name[2:8]
species_name[11:] (without a value after the colon)
species_name[:4] (without a value before the colon)
species_name[:] (just a colon)
species_name[11:-3]
species_name[-5:-3]
What happens when you choose a stop value which is out of range? (i.e., try species_name[0:20] or species_name[:103])

Additional resources on indexing & slicing:
https://www.geeksforgeeks.org/how-to-index-and-slice-strings-in-python/
https://www.tutorialspoint.com/what-is-a-negative-indexing-in-python

Data types in python:

- Integer: int
- Float: float
- String: str

Text type: str
Numeric Types: int, float, complex
Sequence Types: list, tuple, range
Read more:
https://www.w3schools.com/python/python_datatypes.asp
https://docs.python.org/3/library/stdtypes.html

These are built-in data types.

Data Types can be defined by users too, often called class.
One of the features make python flexible and powerful.

Operations:
+ can be used to join strings too
/ can devided float and integer
// division result is an integer

What type of value (integer, floating point number, or character string) would you use to represent each of the following? Try to come up with more than one good answer for each problem. For example, in # 1, when would counting days with a floating point variable make more sense than using an integer?

1.Number of days since the start of the year. --> integer (int) or float
2. Time elapsed from the start of the year until now in days.
3. Serial number of a piece of lab equipment.
4. A lab specimen's age --> depends could be integer or float
5. Current population of a city.

6. Average population of a city over time.

Arithmetic with Different Types
Which of the following will return the floating point number 2.0? Note: there may be more than one right answer.

first = 1.0
second = "1"
third = "1.1"

1. first + float(second) --> this one
2. float(second) + float(third)
3. first + int(third)
4. first + int(float(third)) --> this one
5. int(first) + int(float(third))
6. 2.0 * second

Getting help:
- help(functionName)
- functionName?
- functionName + Shift + tab

.isupper is a method for strings not a function. So, if you type element.isupper() and do shift + tab, it will work to pull up the help info.

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)))
```

whos: lists the variables you currently have defined

Libraries:

We can expand capabilities (functions) available to us by importing libraries

Syntax:

import <LibraryName>

import math

We can also import part of a library like so:
from math import cos, pi

no longer need to put math.pi is how you later access it

pip list: shows all the packages you currently have installed (includes default ones included in Anaconda install)

other library (package) resources
- Standard Library https://docs.python.org/3/library/
- PyPI https://pypi.org/
- Science, math, engineering communities https://www.scipy.org/
- Powerful package for visualization https://matplotlib.org/

Practice:
from math import log
log(0)dat

data.info(): provides info on the loaded dataset 'data'

To review:
- class: how you define a data type
More info: https://www.w3schools.com/python/python_classes.asp
- type() will let you know what data type the object is (e.g., dataframe)
- function: takes a variable explicitly
type(data), the type function takes the data variable
- method: don't pass variable explicitly. instead the method applies to object implicitly
data.info()

difference between method and function
https://www.geeksforgeeks.org/difference-method-function-python/
https://stackoverflow.com/questions/155609/whats-the-difference-between-a-method-and-a-function

Function -
- can be called by name.
- Can be EXPLICITLY passed data to operate on and
- can optionally return data

Method -
- Called by a name that is associated with an object
- method is implicitly passed the object on which it was called
- a method is able to operate on data that is contained within the class. (an object is an instance of a class)

print(data.T): transposes data
data.describe(): provides basic stats on data

print(data.iloc[0,0]) will print out the 1st row, 1st col cell of the data dataframe
print(data.iloc[row index,col index])
print(data.loc[:, "gdpPercap_1952"]) will print out all the rows with the gdpPercap_1952 column

Subsetting:

example: print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972']): rows Italy thru Poland and cols gdpPercap_1962 to gdpPercap_1972

to get max of each col in subset, add .max()
print(data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972'].max())

# which values were greater than 10,000?
print(subset > 10000)
will provide a logic value for each cell to state whether the statement is True or False

dir(data) --> to retrieve methods that can be applied to the object in its class.

PLOTTING :

import matplotlib.pyplot as plt

time=[1,2,4,6] # x -axis
position=[0,100,150,200,300] # y-axis

Create a plot:
plt.plot(x-axis,y-axis)
# add axis labels:
plt.xlabel('x-axis label')
plt.ylabel('y-axis label')
Some environments require the following to see plot (or if you want to render plot a little more cleanly in Jupyter Notebook):
plt.show()

Magic line to see plot without the stuff in [] : %matplotlib inline
 plt.show() can also work
 jupyter lab is not showing the problem anymore but if you encounter either, try the magic line of the plt.show()

strip

More plot styles
https://python-graph-gallery.com/matplotlib/

To save the last created figure:
plt.savefig('nameof_figure')

If the current plot isn't clear, use:
plt.get_figure()
or
plt.gcf().savefig()

make your plot accesible
1. large enough font size

2. graph elements easy to see
3. Using color to differenciate but pay attention to colorblind pallete
https://colororacle.org/ https://www.color-blindness.com/coblis-color-blindness-simulator/ to simulate how it looks like for those with colorblindness

Many styles of plot are available: see the Python Graph Gallery for more options. https://python-graph-gallery.com/matplotlib/

Other packages for data visualization https://www.geeksforgeeks.org/top-8-python-libraries-for-data-visualization/

list_name.append to add an item to the end of a list
list_name.extend allows you to combine two lists
del to remove items from a list entirely
del primes[-1] removes last item in list

Identifying Item Errors:

Read the code below and try to identify what the errors are without running it.
Run the code, and read the error message. What type of error is it?
Fix the error.

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']
print('My favorite season is ', seasons[4])
```

Example:

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

```
values = []
values.append(1)
values.append(3)
values.append(5)
print('first time:', values)
values = values[1:2] or values[1:]
print('second time:', values)
```

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

for loops syntax example:

```
for number in [2, 3, 5]:
    print(number)
```

Indents matter here!

Example:

Reversing a String

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

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

Example:

Create an acronym: Starting from the list ["red", "green", "blue"], create the acronym "rgb" using a for loop.

Hint: You may need to use a string method to properly format the acronym.

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)
```

pressure = 25

If, elif, and else - When nothing else is true, do this

```
masses = [3.54, 6.45, 7.89]
for m in masses:
   if m > 5.0:
      print(m, 'is greater than 5')
   elif m > 3.5:
      print(m, 'is greater than 3.5')
   else:
      print('none of the above is true for', m)
```

Output:
3.54 is greater than 3.5
6.45 is greater than 5
7.89 is greater than 5

Identifying Syntax Errors

· Read the code below and try to identify what the errors are without running it.
· Run the code and read the error message. Is it a SyntaxError or an IndentationError?
· Fix the error.
· 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.")
```

Errors: need (argument1, argument2, etc. as needed) after another function and also add :
Need to indent as well

What does the following program print?

```
def report(pressure):
   print('pressure is', pressure)
   print('calling', reporefdefkot, 22.5)
```

Global variables are accessbile outside of functions. Local variables are accessible only within a function. It's important to keep this in mind when defining functions.

READING ERROR MESSAGES:

Read the traceback below, and identify the following:

· How many levels does the traceback have? 3 levels of code
· What is the file name where the error occurred? errors_02.py
· What is the function name where the error occurred? print_friday_message()
· On which line number in this function did the error occur? Errors best read from end of error message to top this line is listed last (but if you read from end it tells you right away the line where the error occured: line 11 in this case
· What is the type of error? key error, meaning that wne the print_frinday_message() function is called the function was not given a message, like the sunday message is. It should have had "friday" : "end of the week yay!"

  - "friday" : "end of the week yay!" --> missing value example. "friday" is key and "message" would be the value the function would return but since the value is not given, we are getting an error.

· What is the error message? KeyError: 'Friday'

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-2-e4c4cbafeeb5> in <module>()
     1 import errors_02
----> 2 errors_02.print_friday_message()

/Users/ghopper/thesis/code/errors_02.py in print_friday_message()
    13
    14 def print_friday_message():
---> 15     print_message("Friday")
```

```
/Users/ghopper/thesis/code/errors_02.py in print_message(day)
      9         "sunday": "Aw, the weekend is almost over."
     10     }
---> 11     print(messages[day])
     12
     13
```

Coding style best practices:

- document your code and ensure that assumptions, internal algorithms, expected inputs, expected outputs, etc., are clear
- use clear, semantically meaningful variable names
- use four white-spaces, *not* tabs, to indent lines (tabs can cause problems across different text editors, operating systems, and version control systems)

Python standard (style guide): https://www.python.org/dev/peps/pep-0008
Google Style Guide for Python https://google.github.io/styleguide/pyguide.html

Docstrings can be multiple lines and as long as they are enclosed within 3 quotation marks the lines will appear when help function is called on the function you defined. So they are helpful documentation in providing guidance for what the function does and what types of variables, and how many variables it needs.

More resources for learning Python
Coursera course https://www.coursera.org/specializations/data-science-python
Microsoft Python learning : https://youtube.com/playlist?list=PLlrxD0HtieHhS8VzuMCfQD4uJ9yne1mE6
Google Machine Learning Crash Course https://developers.google.com/machine-learning/crash-course
Data Quest https://www.dataquest.io/
Havard Data Science courses https://online-learning.harvard.edu/subject/data-science

Links for Instrumentation Python programming
https://magna-power.com/learn/kb/instrumentation-programming-with-python
https://pyvisa.readthedocs.io/en/latest/ (Python package)
https://web.math.princeton.edu/~cwrowley/python-control/intro.html
https://www.oreilly.com/library/view/real-world-instrumentation/9780596809591/ (book)
https://www.youtube.com/playlist?list=PLMFn2UaPBVfWVjCbQh_DsJknlwpVkiIhh (YouTube course)

Links for Python Notebooks used for today in this GoogleDrive Folder
https://drive.google.com/drive/folders/1oe4ZSMIlAT_FEW8UdPRYaLx3cSMibZNa?usp=sharing

DAY 3 GITHUB/GIT LESSON AND OPEN SCIENCE

Git/GitHub Lesson: https://carpentries-mit.github.io/githublesson/notebooks.html
Git/GitHub slides: slide 41 - 42 in http://bit.ly/SCW_MIT_setup
Link to the Open Science Slide Deck: https://docs.google.com/presentation/d/1hY-T2fq2v8IxJZMIBCV9K37S4HcuwdUgcrOjDp8R-q0/edit?usp=sharing

git config --global user.name
git config --global user.email
git config --global core.editor
git config --help

To see your current configurations:
git config --list

how is a git directory different from a regualr folder?
has hidden files that allow for version control with git
cmd+shift+. to display on Mac

Adding an existing repository to GitHub desktop: https://docs.github.com/en/desktop/contributing-and-collaborating-using-github-desktop/adding-and-cloning-repositories/adding-a-repository-from-your-local-computer-to-github-desktop

Tip: To toggle display hidden files,

- On Mac, Command + Shift + Dot
- On Windows 10 / 8, from the File Explorer window (shortcut: windows key + E) choose –> View –> Show/hide –> check Hidden Items.

Creating a new .txt file (nano, notepad, or other text editor) in our repo folder appears as 'change' in the GitDesktop staging area
In order to commit change to main, it's good to write a description to document the change (for future me or other collaborators)

When making multiple changes, using the checks I can decide which changes to commit
*You can undo and ammend commits
*Probably better to do many small commits (?)

When making subfolders within the repo folder, GitDesktop does not recognise the change to be commited until there are files within it.

what are .DS_store files?
- system files for Mac
- not useful to keep track of from project perspective
- gitignore file has a list of exensions that it will ignore. can add _DS_store file to this
- once added to gitignore, git will not track changes

what kinds of files work with github? git is better with tracking simple text files
- python code (similar to .txt file)
- most programming languages have files in a git-friendly format
- jupyter notebook --> converted to json file, although this can be complicated for tracking commit

collaborating
- be careful with who you invite to collaborate, this is powerful
- collaborators have rights to resolve conflits or merge changes (but the owner can change this)
- for strangers, use forks

Examples of different projects on GitHub,
https://github.com/dhmit/lang_learn
https://openclimatedata.net/
https://docs.open-reaction-database.org/en/latest/ (https://doi.org/10.5281/zenodo.6028405)
https://github.com/connorcoley/rexgen_direct
https://ualibweb.github.io/UALIB_ScholarlyAPI_Cookbook/content/about/introduction.html