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

---

General

- Ask questions in the chat
- Use the BBB status (raise hand or so)

Software Setup

- written guide (at the bottom):
    - http://swc-bb.gitext.gfz-potsdam.de/swc-pages/2020-06-22-virtual/

- Git remote setup

Everyone needs an account at GitHub, so that we can work together in the Git lesson later.
Create a GitHub account here:

- https://github.com/

Shell

set-up:
1. download this file:

- http://swc-bb.gitext.gfz-potsdam.de/swc-lessons/2020-06-22-potsdam-berlin/shell/data/data-shell.zip

2. unzip it to the desktop

3. start a shell:

- on unix / mac
    - Terminal
- on windows
    - Git Bash

For Windows users if they want full functionality they need to activate the Windows Linux Subsystem (v.

1 or v. 2) and then install a Linux Distribution from the Microsoft store. See here:
https://docs.microsoft.com/en-us/windows/wsl/install-win10
This works very well also for other tasks.

Useful info:
1. If you are stuck in a running script: Ctrl+C stops the execution!

2. pwd stands for : "Print working directory".

3. clear: hide current displayed outputs from console. This can be usful to decrease the amount of displayed information in the consol.

4. man: stands for manual (the command is in the git bash under windows not avaible, use `command --help` instead (for example `ls --help`)
5. -a : is used to display all files within a directo-ry(including hidden files)

- usage: ls -a

6. two dots(..): is used to navigate to a parent directory of the current directory you are working.

- usage: cd ..

7. tilde(~) : shorthand for home directory

- usage cd ~ is used to return to a home directory

8. Path to home directory depends on operating system

- Linux/Mac: /home/username
- Windwos (git bash): /c/Users/username

9. in git bash the `ls` command uses as default the argument `-F`, so the output of `ls` and `ls -F` is the same
10. nano: is a text editor that use a command line interface.

- extra: **vim** is another text editor with similar used as nano

11. With the up/down arrow keys you can brows through the history. The history stores the commands you typed in. When typing th command `history` you can show its contents.
12. Copy & Paste depends on the used terminal. Mark the text to copy with the mouse.

- try right cklick and choose "copy" and right click and "paste"
- Ctrl+Shif+C (Copy) and Ctrl+Shoft+V (Paste) on Linux
- also under linux: on some terminals you can paste with the middle mouse button

13. in nano ctrl + X is used to exit the editor
14. With tab it is possible to autocomplete, so you don't have to type the whole file/directory names: For example: `cd /ho<tab>` will autocomplete to `cd /home`

15. touch: is used to create a file /files

- eg:1 touch file1.txt
- eg2: touch file1.txt file2.txt

16 -i : in command is refers to interactive and used to produce confirmation question for the actions that is goiing to be performed

- eg: rm -i file1.txt
    - it will ask if we are sure to delete file1.txt.

17. wildcard * is a placeholder for N number of characters. It can be used to categorize files with similar defined scope to perform a certain operation

- eg: the following command remove all txt files in our current working directory:
    - rm *.txt

18. the pipe bar symbol | can put the output of a program as input for another command, for example:

- wc -l *.pdb | sort -n | head -n 1
    - to get the .pdb filename with the lowest number of lines

Feedback
+ everything was well explained
easy to follow, good shared notes
+ good pace for me, not too fast and not too slow
+ well done, learned a lot!
+ Easy to follow
+ parallel 'Windows' comments in the chat were very helpful
- little fast for me, especially with all the sudden Ctrl+L (add: I'd also recommend to not clear screen so often)
-

Git 1

Useful info:
1. git config is used to configure git with information that can be used for a single repository or globally to be used by all repositories in your system

    - for example, **git config -- global user.name "your name"** will configure author name thta is going ot be used for all comits in all repositories, while if **--global** is removed, the configuration will be applied only to a single repository

2. git.config --list: is used to displaye all information that has been configured for git

3. **git init**: is used to initialized an empity git repository

    - **ls -a** can be used to visualize all files in a newly created git repository

4. **git status:** is used to see the current status of a git repository . If there are files we modified but didnt track the changes, they will be listed.

5. **git add filename :** is used to track a file
6. **git commit -m "commit message"** : is used to capture the current state of the newly added file. a commit message is required to commit a change into a repository.

    - **-m** refers to a message

7. **git diff**: is used to display newly untracked changes to a file
8. If you didnt add a newly modified file i using git add, but tried to use git commit instead, git won'tt allow the commit because the changes are not staged.

8. **git diff --staged** is used to see changes that are staged
9. **git log**: is used to display commits history

    - git log --oneline : is used to display each commit info in one line

10. **git diff commit_id filename**: is used to see changes between the current vrsion and the version with commit_id

11. **git checkout commit_id filename**: will roll back the file to a version with specified commit_id
12. **git stash**: is used to discard changes but it save them for later
* **git stash list** : used to see the content of stashed changes

- git stash pop: remove the top stashed version from stash list, but also bring it back to be added again.
- instead of git checkout version that might be usful later, it can be stashed and can be recovered after if needed using stash id
    - eg: git stash apply stash@{id}

13.

Feedback
+good examples of why it is useful
+ speed was OK for me
+ good speed and explanations!
+ very clear explanations!
- felt too fast
- a bit too fast
- last part with solving conflicts was a bit too fast for me to follow

Git 2
Useful info:
1. git remote add origin repostiroy_link: used to add the location of the remote repository, so that local changes can be pushed and pexistingchanges that are not in the local branch can also be pulled from the remote
2. git clone link_to_repository: make a local copy of the the remote repository.

Feedback
+ great to see how Git in the local Shell connects to GitHub, always wondered about how that works out
+ Speed was fine
+ very clear explanations!
- adding the commands to the shared notes would be nice
- partner work was a bit confusing because the instruction was at the same time and in the main session. Working with breakout sessions might be easier
- a bit confusing with the 2 sides at once
-to make work easier, it would be useful to upload already preapared files, so everyone is working on the same content. Could avoid some of the problems that occurred
- when we started working with a partner it was not very clear that we both had to add each other, we thought it was just one adding the other one-

Python 1

Please download

- http://swc-bb.gitext.gfz-potsdam.de/swc-lessons/2020-06-22-potsdam-berlin/python/data/python-novice-inflammation-data.zip
- http://swc-bb.gitext.gfz-potsdam.de/swc-lessons/2020-06-22-potsdam-berlin/python/code/python-novice-inflammation-code.zip

Useful info:

1. in python, we can assign value to a variable without pre-defining the type
    - **Syntax: variable_name = value**

2. import library_name: imports the specified library , so that all the functions/modules defined within that library can be used.
3. **numpy**: stands for numerical python (used to work with nuzmerical data in different dimention)
4. imported libraries can be igven an **alias** using as. This can be handyy not to yype the full name or the library. Instead a short name can be given as a nickname.
    - eg: import numpy **as** np

5 . rules for python variable names:

- can contain upper and lowercase char, digits and special characters
- first char cannot be a digit
- it is case sensitive

6. float64 in numpy: is a double precision float

7. **# :** is used for a single line comment
    - there is no standard multiline comment in python, but ''' can be used to write multiline strings which then can be ignored by python . so it can be used as multiline comment.

Feedback
+ liked the exercises
+ first part easy to follow
- the structure of the data example (columns/rows, patients/days) was not clear to me right away / was hard to keep in mind while code was presented
- don't know if there is another way to introduce Python or if this is what most people use Python for but I am not using math in every day work which made it a bit difficult to follow for me
- arrays were a little confusing
- as a beginner I find it easier to follow when no code is omitted
- I found lists easier than arrays so maybe arrays could have been explained at a later point

Python 2

Visualization Using Python:

1. using matplotlib, visualize avg, min and max

```
# Visualizing data using Python
import numpy as np
import matplotlib.pyplot as plt

data = np.loadtxt(fname = 'inflammation-01.csv', delimiter=',')

print("1. Heat-map:")
image = plt.imshow(data)
plt.show()

print("2. Average inflammation per day:")

avg_inflammation = np.mean(data, axis = 0)
avg_plot = plt.plot(avg_inflammation)
plt.show()

print("2. Minimum inflammation per day:")
min_inflammation = np.min(data, axis = 0)
min_plot = plt.plot(min_inflammation)
plt.show()
```

Put your solutions for the tasks here:

```
# Exercise
# 1. Plot std of recorded inflammation data
print("5. Standard deviation of inflammations per day:")
std_inflammation = np.std(data, axis = 0)
std_plot = plt.plot(std_inflammation)
plt.show()

# 2. Plot the difference between min and max inflammation recorded per day
print("6. Difference between Min and Max:")
diff_inflammation = max_inflammation - min_inflammation
diff_plot = plt.plot(diff_inflammation)
plt.show()


# 3. Plot three subplots (min, diff, std) of recorded inflammation together aligned in one column
print("6. Grouping new plots together:")

fig = plt.figure(figsize = (3.0, 10.0))

axes4 = fig.add_subplot(3, 1, 1)
axes5 = fig.add_subplot(3, 1, 2)
axes6 = fig.add_subplot(3, 1, 3)

axes4.set_ylabel("Minimum")
```

```
#axes4.set_ylim(0, 22)
axes4.set_xlabel("Days")
axes4.plot(min_inflammation, drawstyle = "steps-mid")

axes5.set_ylabel("Difference between Min and Max")
#axes5.set_ylim(0, 22)
axes5.set_xlabel("Days")
axes5.plot(diff_inflammation, drawstyle = "steps-mid")

axes6.set_ylabel("Standard deviation")
#axes6.set_ylim(0, 22)
axes6.set_xlabel("Days")
axes6.plot(std_inflammation, drawstyle = "steps-mid")

fig.tight_layout()
plt.show()
```

```
# 1. Plot standard deviation of recorded inflammation data
std_data = np.std(data, axis=0)
std_plot = plt.plot(std_data)
plt.show()

# 2. Plot the difference between min and max value recorded data
diff_data = max_inflammation - min_inflammation
diff_plot = plt.plot(diff_data)
plt.show()

# 3. three subplots aligned in one column (min, diff, std)
fig = plt.figure(figsize=(5.0, 10.0))

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

axes1.set_ylabel('min')
axes1.plot(min_inflammation)

axes2.set_ylabel('diff')
axes2.plot(diff_data)

axes3.set_ylabel('std')
axes3.plot(std_data)

fig.tight_layout()
plt.show()
```

Exercise answer:

Exercise 1

#1. Plot standard deviation of recorded inflammation data:

```
print("standard deviation of recorded inflammation data per day:")

std_inflammation = np.std(data, axis = 0)
std_plot = plt.plot(std_inflammation)
plt.show()
```

#2. Plot the difference between min and max inflammation recorded per day:

```
print("Difference between min and max inflammation recorded per day:")

diff_inflammation = max_inflammation - min_inflammation
std_plot = plt.plot(diff_inflammation)
plt.show()
```

#3. Plot three subplots (min, difference between min and max, and
standard deviation of recorded inflammation per day)
together aligned in one column

```
print("Group plots for min, difference between min and max, and standard deviation:")

fig = plt.figure(figsize = (6.0, 10.0))

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

axes1.set_ylabel('min')
axes1.set_xlabel('days')
axes1.plot(min_inflammation)

axes2.set_ylabel('max-min')
axes2.set_xlabel('days')
axes2.plot(diff_inflammation)

axes3.set_ylabel('std')
axes3.set_xlabel('days')
axes3.plot(std_inflammation)

fig.tight_layout()
plt.show()
```

Questions:

- If I have a .csv with headers (e.g. first column: depth), can I do something similar like data$depth in R?

Nils: There is a python package called pandas, that provides exactly this functionality (then with data['depth']; so using the squared brackets to get the column.)

Michael: If you have very large data in csv, or other text (ASCII) files, I found better to use numpy's loadtxt function because pandas seems to be relatively slow (in my reference case with > 20k entries). Also there is a built-in csv reader module which is a bit more complicated but very quick.

Feedback
+ very usefull for everyday life, easy to follow
- with subplots it would have helped me to see the desired result before coding or even better while coding

Python 3

Useful info:

1. list vs string:

- elements in sting cannot be modified by position(immutable), while in list, we can modify an element value(mutable)

2. List vs array:

- element sin array should have the same data type, while not in list
- both are mutable

3. we can nest lists and they can be accessed using their position:

- eg: list1 = [['a', 'b'], ['c', 'd']]

4. **append**: is used to append additional element to existing list

- eg: list1.append('f')

5.

Excersis:

1. Use a for-loop to convert the string “hello” into a list of letters: ["h", "e", "l", "l", "o"]

Hint: You can create an empty list like this:
my_list = []

2.

```
string_for_slicing = "Observation date: 02-Feb-2013"
list_for_slicing = [
                ["fluorine", "F"],
                ["chlorine", "Cl"],
```

```
        ["bromine", "Br"],
        ["iodine", "I"],
        ["astatine", "At"]
    ]
```

Feedback
+ I liked the part where it was shown that some code has the same result (:, :35,: len(beatles))
+ speed was good
- would be could the have the code for excercises beforehand (agree)
- would have needed more time for exercises
- this was too fast for me

Python 4

Excersis:

```
#Exercise 1
#1. check if the maximum recorded value for patients on day 10
#is larger than the one on day 12

max_inflammation_10 = np.max(data, axis=0)[10] # [0] on day 10
max_inflammation_12 = np.max(data, axis=0)[12] #[20] on day 12

if max_inflammation_10 > max_inflammation_12:
    print('Max_Inflammation on day 10 is greater than on day 12')
else:
   print('Max_Inflammation on day 12 is greater than on day 10')

#Exercise 2
#check if the maximum recorded value for patient 1 larger than the
#maximum value recorded for patient 2 throughout the 40 days

max_inflammation_p1 = np.max(data, axis=1)[0]
max_inflammation_p2 = np.max(data, axis=1)[1]

print(max_inflammation_p1)
print(max_inflammation_p2)

if max_inflammation_p1 > max_inflammation_p2:
    print('Max_Inflammation of patient 1 throughout 40 days is greater than patient 2')
elif max_inflammation_p1 == max_inflammation_p2:
    print('Max_Inflammation of both patient is equal')
else:
   print('Max_Inflammation of patient 2 throughout 40 days is greater than patient 1')

###

max_inflammation_0=np.max(data0, axis=0)[0]
```

```
max_inflammation_20=np.max(data0, axis=0)[20]
max_inflammation_9=np.max(data0, axis=0)[9]
max_inflammation_11=np.max(data0, axis=0)[11]
max_inflammation_pat0=np.max(data0, axis=1)[0]
max_inflammation_pat1=np.max(data0, axis=1)[1]

if max_inflammation_0==0 and max_inflammation_20==20:
    print('Suspicious maximum value present.')
elif np.sum(np.min(data0, axis=0))==0:
    print('Sum of minimum is zero.')
else:
    print('Everything is OK!')

if max_inflammation_10 > max_inflammation_12:
    print('larger value')
else:
    print('equal or smaller')

if max_inflammation_pat0 > max_inflammation_pat1:
    print('larger value')
else:
    print('equal or smaller')

max_inflammation_10 = np.max(data, axis=0)[10]
max_inflammation_12 = np.max(data, axis=0)[12]

if max_inflammation_10 > max_inflammation_12:
    print('yes')
else:
    print('no')

max_inflammation_0 = np.max(data, axis=1)[0]
max_inflammation_1 = np.max(data, axis=1)[1]

if max_inflammation_0 > max_inflammation_1:
    print('yes')
else:
    print('no')

def diff_numbers(number1, number2):
    diff = number1 - number2
    if diff % 2 == 0:
        print('False')
    else:
        print('True')
```

Feedback

+easy to follow, good exercises

+ good pace, easy to follow

+ explanation is clear

-