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 etherpad.wikimedia.org).

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

---

Useful links

Workshop info page: https://swc-kc-london-01-2020.github.io/SWC-KCL-202001/
This pad: https://pad.carpentries.org/2020-01-09-kcl

The Carpentries: https://carpentries.org/
Data Carpentry: https://datacarpentry.org/
Library Carpentry: https://librarycarpentry.org/
Software Carpentry: https://software-carpentry.org/

Code of Conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

Readings:
- Best Practices for Scientific Computing
https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745

DISCLAIMER: the pad doesn't have a grammar check, so please excuse typos!!

Instructors list

Alessia, Research Fellow in Computational Biology, Trying to learn Modern Greek
Luca, PhD, Materials Science & Engineering, I live close to the Arctic Circle
Stefania, postdoc, Developmental Biology, I love pizza!
Neil, Research Software Analyst,
Walter, Clinical Research Fellow, I think platypus are cool

To keep in touch

Luca, https://www.linkedin.com/in/lucadistasio/, https://www.researchgate.net/profile/Luca_Di_Stasio2
Stefania, stefania.marcotti@kcl.ac.uk
Alessia, https://alesssia.github.io/
Walter, walter.muruet_gutierrez@kcl.ac.uk

Participants

Hi everyone!! In the top right you can find a name list, add your name next to the colour that has been assigned to you!

Cathy Davies (Institute of Psychiatry). Postdoc in Neuroimaging/Psychosis. Need to know how to manage/work with huge data!
Fiona Wardle (Randall Centre, Guy's Campus, genomics research). Co-organiser of this Carpentries workshop.
Natasha Romanova (KDL) - assisted with organising this workshop
Grant Wray (Centre for Developmental Neurobiology, Guy's campus.) IT Manager.
Lara Gardellini (MSc in Digital Humanities at UCL)
Sarah Crofton (Department of English) Teaching fellow in C19th literature
Lorcan Browne (Centre for Developmental Neurobiology, Guy's campus. Post Doctoral researcher).
Hannah Morcos (Post-doc, French) - I am working on a digital edition of an Old French text and would like to learn more about what goes on behind the scenes.
Kalina Tcholakova - MSc student in Organisational Psychiatry and Psychology - scared about programming! but we need to push ourselves out of our comfort zones :)
Connor Fairbairn (Philosophy, PhD Candidate)
Erika Melek Delgado (Department of Spanish, Portuguese and Latin America Studies, Leverhulme Early Career Fellow).
Manfredi de Bernard (Culture, Media and Creative Industries, PhD Student)
Jerome Greenfield (Department of History, early career fellow)
Ben Pelling (Department of History, PhD Candidate)
Sarah Flatters (IoPPN)
Harriet Cook, PhD Student, Spanish, Portuguese and Latin American Studies Department
Henry Ravenhall, PhD student, French
Chen Xiong (Classics Department, Strand Campus, PhD candidate)
Alice Koltchev (MSc student in Neuroscience - Wolfson CARD - hoping to automate analysis of joint angles and locomotion following spinal cord injury)
Suthini , PhD., Digital Humanities
Hannah Connell (CDP History, King's College London and The British Library)
Jiawei Zhao, PhD, CMCI
Sam Thompson, PhD student, German
Liang Ge, PhD, CMCI
Nicol Bergou (IoPPN), Research Assistant in Psychological Medicine
Zain Alhashem, PhD student, Randall Centre

2020-01-09, Thursday

Total participants: 24

Humanities: 16 - 67%
Life Sciences: 8 - 33%

Introduction to working with Data section

Reference webpage(s):
https://librarycarpentry.org/lc-overview/03-jargon-busting/index.html
https://librarycarpentry.org/lc-overview/04-computational-approach/index.html

Data! Terms, concept, ideas?
Do you want to know more?
Short activity: in pairs! Say hi to your neighbour! 3 minutes to think about concepts related to coding/data that you heard about and want to know more. MAKE A LIST!
30 s left! Time to wrap up

- What terms, phrases, or ideas around code, data, or software development have you come across and feel you should know better?

Automated scripts
Database structures (SQL? etc?)
Opensource platforms
Big Data Analysis
Data Transformation
Data preparation
Data cleaning
Collaborative tooling (e.g. GIT)
Data mining
Code compiling
For loops
bioimage analysis
What is code?
Web archiving
Algorithm for Social Network Analysis
statistical analysis?
XSLT

Where to look for definition:
Dictionary of Technical Terms: https://techterms.com/category/technical
Data Thesaurus: https://nnlm.gov/data/thesaurus
Sideaways Dictionary: https://sidewaysdictionary.com/#/

Get in larger groups of 4, write on the PAD two definitions of terms in your (joint) list, one your confident with, another one you think you're struggling with (10 minutes). You can use the tools listed above and your own knowledge!
WRITE BELOW!!
time to wrap up! Write on the pad below!

Terms/concepts we were able to explain
Web Archiving - storage of website 'copies' over time so researchers can see previous iterations.
Data Mining - forming a new dataset using data extracted from a variety of sources (published data sets etc) using 'knowledge discovery techniques' (pattern recognition); it's quite a broad field! Get some data and extract knowledge. Find patterns, groups

Data Transformation - Transforming data from one format to another into a specific structure/style for further use. Absolute nightmare of all data analysts!
Algorithm for Social Network Analysis - Automated instructions on database to understand roles in a SN. Data analysts love automation!

Terms/concepts we are still struggling with
GIT - collaborative tools for code and project management / sharing / development. GIT is scary!! You are not alone :) It's a place where we store our code. In the simplest scenario it's just you, but it can be collaborative. Every change is traced over time, so it'll remember all the previous versions. (more resources to come!)
compiling/ code compiler - (more resources to come!)
big data - some kind of data that (according to the person defining it) are BIG! E.g. weather, genomics. The concept is related to data mining: discover trends! https://sidewaysdictionary.com/#/term/big-data
data mining - see above
Extensible Stylesheet Language - if we speak to a machine we can't talk "HUMAN", so we need to use a language that is shared with the machine. It's a way of formatting information so that the computer can understand. (HTML, XSL, GML). Nesting data in a readable format

Some Resources

- GIT:
    - Software carpentries: https://swcarpentry.github.io/git-novice/
    - Getting started (a beginner guide): https://towardsdatascience.com/getting-started-with-git-and-github-6fcd0f2d4ac6
    - A step by step guide: https://opensource.com/article/18/1/step-step-guide-git
    - Hands-on workshop (you need a few friends for this!): https://github.com/feiphoon/github-workshop
    - The reality: https://xkcd.com/1597/
- Compiling troubleshooting
    - Google it! (Google is always your friend)
    - Look for an INSTALL file, or a README file. It may contains useful information
    - You can always contact the developer(s), often there are also mailing list/Git issues/Google Groups where developers and other users can help you

- Is there something you would like to automate in your work?

One of the key concepts! Manual way vs. automated way (tell a machine to do it for us). Why? Time-saving (repetitive tasks, if the PC does it for me I can go for coffee); less likely to make mistakes. Not everything can be automated :(and not everything is worth it to be made automatic (consider how long it'll take to make it automatic?)

Groups of 4 people: is there anything you'd like to automate in your work? You don't have to know *how* to do it (yet) (5-10 minutes), off you go!!

In our research on spinal cord injury, we perform a series of tests to assess locomotor function and spasticity; watching videos of these tests typically takes weeks and weeks, and automating the analysis of joing angles/general locomotion would be extremely useful and time-saving, and potentially help remove bias.

In archival work, we often have to enter historical/factual/background information manually from spreadsheets into databases. Automating the process would save much time and keep the two files consistent.

Checking word frequency (and usage) in large sets of data; identifying, retrieving and collating data from different platforms.

Comparing many literary texts for common ngrams (but especially to find patterns with variations) for genre analysis or uncovering intertextual influence. (I've been misuing plagiarism software for this...)

Extract text strings (e.g. dates, names) from large collection of text documents.
Topic model (LDA) visualisations.
Opening / Transforming Rdata in Python.

For a set of files containing numeric data (e.g. csv files, excel files), read in the data into a python dataframe, perform a series of operations, and write the new data out into an output file.

Automate the management, preprocessing and analysis of MRI data - so using a list of subjects, and having a script perform a set of operations for each subject (e.g. amending JSON and header files, moving, renaming, opening/running data through specici software within python). To have error messages reported when it goes wrong/quits!

Automate segmentation, deconvolution, spot detection and co-localasation of florescent microscopy images

Understand roles in Social Network Analysis

Extracting data from clinical health records

How automation really works:
https://xk+cd.com/1319/

- Which of these most accurately describes your data analysis/coding experience?
    - No idea! XXXXxxxxxXXx 12 - 50%
    - I'm confident with user interface software XXXXX 5 - 20.8
    - I can read code X 1 - 4.2%
    - I can write some code XxXXxX 6 - 25%
    - I can write simple programs

Open Refine Section

Reference webpage(s):
https://datacarpentry.org/OpenRefine-ecology-lesson/

Lesson outline (as followed in the session on Thursday)

https://librarycarpentry.org/lc-open-refine/01-introduction/index.html

For some sample data to import in to OpenRefine please use this download link:
https://raw.githubusercontent.com/LibraryCarpentry/lc-open-refine/gh-pages/data/doaj-article-sample.csv

DON'T FORGET to use your sticky notes if you get lost at any point!

OPEN REFINE is a tool for cleaning and tidying data (definition of database: https://sidewaysdictionary.com/#/term/database). It can work with different data formats (e.g. excel files, comma-separated values files), everybody familiar with excel sheets? You can explore the nature of the data, you can do data cleaning/ tidying (e.g. adjust typos, date formats, extract information about addresses into discrete values that the machine can understand: remember? Machines don't speak HUMAN), data reconciliation (e.g. find information in the data set referring to the same person)

http://127.0.0.1:3333 to open OPEN REFINE (OR) in the browser

There's a number of ways to bring data into OR, upload from files, from a website, write it in OR directly. Link sample data by using the address https://bit.ly/35xffGr in web address URL field on OR

Give project a name: rename to "MyProject2"
Let's have a look at the user interface!
OR tidied our data into a tabular view
Check "Parse next" to identify the column titles
Check "Character encoding" option (sample data is in UTF-8) - this should sort out weird character strings
Don't check "Parse cell text into numbers, dates, ..." (we are happy to read everything as text for now)

Click on "CREATE PROJECT"
We are presented with a preview of the first rows of data (default view, Excel-like);
Show as *rows* or *records* (associate multiple rows)
This character "|" is called pipe and indicates multiple entries in the same field (unlikely to appear in the database, oppositely to commas, semi-cols, and so on)
[AUTHORS > Edit Cells > Split multi-valued cells] (select the pipe character "|")
Now *rows* and *records* do not show the same thing anymore, we divided multi-valued cells in the AUTHORS field
It's not telling us a lot (yet) but it can be a powerful tool for later operations!
Let's put it back as it was (Join multi-valued cells in the AUTHORS field)

Look at the data again, go to the SUBJECTS column and split the multi-valued cells. What is the separator this time?

The next thing is exploring data by using facets. Facets are powerful search tools.
Scroll over to the PUBLISHER column. A facet is a way by grouping data by selected data types and values.

Create a text facet on the LICENSE column. What is the most commonly used license? (Remove it to move on)

You can use the UNDO/REDO menu to go back to a previous stage in the analysis. You can "Extract

Operation History" to obtain a log of the operations you performed. This could help if we perform the same operations over and over again, to speed up this phase. If you click on the very first stage every following operation will be lost (you are changing the very nature of the database); any other stage allows us a back/forth approach and is reversible.

Let's look at filters. Let's look at the SUBJECT column, select the text filter to look for a specific string (e.g. "fish")

You can edit the data using the Facet function. Create a new text facet in the LANGUAGE column (check the available unique values). We want to format the field "English", click on edit and replace with "EN" so that we have consistency in the language encoding. We can include multiple facets at once (use the include option). There are other ways of doing this! (Remove the facets to move on)

Split the AUTHORS column into single values cells (remember little old pipe? ||||). We can CLUSTER similar values together. Click on [AUTHORS menu > Edit cells > Cluster & Edit]. Drop down boxes define the algorithms to decide if things are candidates to be the same thing (e.g. is A. Khan Vakeel the same of Vakeel A. Khan?). Scroll down the list and select which instances are the same (e.g. the same author): do you agree with OR suggestion? Hit "Merge selected and re-cluster" to confirm. Test changing the "keying function" in the drop down menu: it changes the algorithm that looks for matches (rejoin multi-value cells to move on)

Some resources on different available algorithms in OR:
https://github.com/OpenRefine/OpenRefine/wiki/Clustering-In-Depth

How to change the order of column display? Check the drop down menu in each FIELD to move right, left, at the beginning, at the end. Or use the ALL drop down menu to rearrange or remove columns. Rearranging columns will not affect the order of the columns in the output (when exported; the rearrangement is local to OR), but if you remove a column it's GONE FOREVER!! Use redo/undo menu to retrieve.

Transformation time!! We've done some data cleaning up to now, it's time to define expressions to transform data. [Field menu > Edit cells > Transform]: we can define an expression to apply a transformation (e.g. value.touppercase() to transform field into upper case). We can stack expressions with this syntax e.g. value.toUppercase().toLowercase().toTitlecase() (title case will capitalise the first letters of each word). The pop up window will show us a preview of the results after the transformation. This language is called "general refine expression language (GREL)". [Field menu > Edit cells > Common transforms] gives us a list of commonly used transformations.

Data types: *string* (collection of alphanumeric characters stored as text), *number*, *date* (can be sorted chronologically), *boolean* (see below), *array* (list of values)
Boolean is a True/False variable! Can also be 0/1 (in machine language). E.g. is the word "fish" appearing in a field? If yes assign True (or 1), otherwise False (or 0)
How to change the data type? [DATE > Edit cells > Transform value.toDate("dd/MM/yyyy")] where dd stands for day (2 digits), MM for month (2 digits capital letters), yyyy for year (4 digits). To do the opposite we would use toString()
We can add columns based on other columns content. [FIELD menu > Edit Cells >Add column based on column X] where X is another column of the database (e.g. if we don't want to transform the values in a column locally, but add a column with the new transformed values).
Arrays are lists of values (e.g. animals = ["cat", "dog", "mouse", "llama"]; this is an array! Note that most

programming languages start counting from 0, so animals[1] = "dog", not "cat" as we might expect; animals[0] = "cat")

Function that returns a Boolean: look for commas in a field [FIELD menu > Facet > Custom text facet] value.contains(",") and apply facet. This returns a boolean type (true if the comma is there, false if it isn't)
Function that uses an array: To get the author name in the natural order you can reverse the array and join it back together with a space to create the string you need. We first divide name and surname into a list of values with the function match() ; e.g. [Neil, Jakeman] will become array=["Neil", "Jakeman"]. [Edit cells > Transform] value.match(/(.*),(.*)/).reverse().join(" "). Between the "/"s is the syntax we are looking for; the "."s indicates the daisy-chain of operations we are performing. Reverse change the order of the array values and join put them back together in a string (not an array anymore!).

Reconciliation of data allows to find matches between possibly similar entries. Reconciliation services allow you to lookup terms from your data in OpenRefine against external services, and use values from the external services in your data. [FIELD menu > Reconcile > Start Reconciling] using the VIAF Reconciliation Service. Single tick: correct identification; Double tick: accept reconciliation in the whole database.

Questions!
How would one merge in 'Cluster & Edit' if there is a set of three but only two need to be merged? You might want to check the Reconciliation of data service using external references as a different way to obtain the same result as we were trying to achieve with the Cluster & Edit. It should be more powerful in recognising correct matches.

Automating Tasks with the Unix Shell section

<---------->
Intro Material

Reference webpage(s):
http://swcarpentry.github.io/shell-novice/

How to install Windows Subsystem for Linux (WSL):
https://www.windowscentral.com/install-windows-subsystem-linux-windows-10

How to improve your coding skills:
https://xkcd.com/323/

On the choice of text file editor:
https://xkcd.com/378/

LINKS TO FILES:
https://swcarpentry.github.io/shell-novice/data/data-shell.zip
https://raw.githubusercontent.com/SWC-KC-London-01-2020/SWC-KCL-202001/gh-pages/workshop-material/loremipsum.txt
https://raw.githubusercontent.com/SWC-KC-London-01-2020/SWC-KCL-202001/gh-pages/workshop-material/cicero.txt
https://raw.githubusercontent.com/SWC-KC-London-01-2020/SWC-KCL-202001/gh-pages/workshop-

material/translation.txt

What's the *lorem ipsum*?
https://www.lipsum.com/

<---------------->
Survey: OS

Windows: +++++++++ +(10)
Mac: +++++++++++++++ (15)
Linux:

if Windows, do you have Windows 10?++++++++++ (10?)

<---------------->
Navigating files and folders

We've been using a user-interface software this morning (OR), now time to move on to some coding! User-interface software use code in the background, it's just that we don't see it

Windows users: Search>Command Prompt this is not the bash shell!

Shell is key sensitive; it's built in a modular way, with functions that can be customised using optional features. You can piece together different functions (simple blocks) to build complex pipelines. It works on the "no news good news", so if no output the operation requested was performed correctly.

You can use arrows (up and down) to navigate the history of commands we wrote and modify them if needed. Use tab for autocomplete (twice in git for Windows to obtain suggestions).

Open the terminal: it's a black screen with a first line telling you: user@computer system ~, with ~ indicating the home folder for the shell; the $ sign indicates where you can write things. You can add things to the "title" area to display more information (e.g. date, time, ...), but we'll leave it as it is. Shell is really flexible!

How to get help? Let's search for help regarding the function "cd" with the command [*cd --help*]; this works with every function/command (e.g. ls). Another option (which might not be installed, so no worries if it doesn't work for you, especially for Windows users) is to use [*man ls*] with 'man' standing for manual (here we are looking for the manual of the command ls). Sometimes the shell prompts us in a different environment (like now in the manual): we can find instructions on how to navigate at the end line of the prompt. The help will tell us what is the function name , what are the options we can set ([OPTIONS]) and what is the function argument (input value).

Navigate and managing files and folders

- [*pwd*] display current folder;
- [*ls*] list folder content (stands for list)
- [*ls -a*] list folder content with an option (indicated with a '-' sign) to not ignore entries starting with . (use -a for all); folders starting with a dot in the shell are hidden folders.
- [ls -l] list folder content with option -l for long (all the info regarding that file)
- [*cd ..*] move to parent folder (cd stands for change directory); parent directory means "going up

one level" in your directory tree
- [*cd ~*] move to home folder
- [*cd folder_name*] move to called folder; e.g. if our home folder is called 'lucad': [*cd lucad*] will do the job Only if *folder_name* is in the pwd. To go home from anywhere it's cd ~. . Good point! You can use ls to find what folders you have
- [*mkdir folder_name*] create a folder named (foldername); e.g. if we want to create a folder called 'WD' we can call [*mkdir WD*]. If we want to create a folder in a different folder than the one we are in, we can use [*mkdir ../folder_name*], the '..' specify we want to move into the parent folder and create the folder there. This uses a relative path (relative to the current working directory), saying: go up to parent (../) than create the folder called folder_name; we can use wild cards here (see below).
- [*touch file_name*] to create an empty file named file_name; handy to initialise an empty file to save outputs afterwards; e.g. [*touch example.txt*]. A file can also be created without an extension (e.g. [.txt], extension is whatever comes after the dot in the file name): the machine doesn't care in a MAC or LINUX environment, as it'll be automatically assigned; WINDOWS wants an extension. The main families of file types are text files (text) or byte files (numbers). Best practice is to always assign extensions regardless of OS. Extension [.sh] creates an executable (readable) for the shell.
- [*nano file_name*] allows us to modify a file. It opens a text editor in a prompt; instructions for navigations are in the lower part of the prompt. To save we can use '^O' (write out) + Enter; to exit '^X'. '^' stands for ctrl. On the Mac, it's ctrl-O, not command-O which will ask the terminal application to open a file. We can also use [*nano file_name*] to create the file directly.
- [*rm file_name*] or [*rm -r folder_name*] to remove files or folders; we need the -r (recursive) option for folders containing files (we need to apply the operation on all files as well as on the folder itself). We can also remove multiple files by providing a list of files as in [*rm file_name file_name1 file_name2*]
- [*cp file_name file_name_backup*] to copy a file called file_name into another called file_name_backup or [*cp file_name folder_name*] to copy a file called file_name in another directory called folder_name (same file name will be assigned)
- [*mv file_name folder_name*] to move a file called file_name to a folder called folder_name. We can also specify a relative path if the folder is not in the current directory (not sure what your current directory is? You can use [*pwd*]). If we call [*mv file_name file_name_new*] it works as a rename: a file named file_name_new is created, the content of file_name is copied into file_name_new and finally file_name is deleted (we rename file_name as file_name_new); file_name_new will be overwritten if it already exists.
- [*cat file_name*] to print to screen the content of the file called file_name
- [*sort file_name*] to sort the content of a file (file_name) and print to screen, by default it will sort the first column alphabethically; if we want to save to a file called file_name_new we need to call [*sort file_name > file_name_new*]. We can sort for a specific column by using the option -k (e.g. [*sort -k 5 file_name*] sorts alphabetically the 5th column, start counting from 0); we can use the option -n to sort numerically and not alphabetically; e.g. [sort -n -k 5 file_name] will sort the 5th column numerically.
- [*head file_name*] to print to terminal the beginning of the file called file_name. We can print the first 3 lines we can specify it by calling [*head -3 file_name*]
- [*tail file_name*] to print to terminal the ending of the file called file_name. We can print the first 4 lines we can specify it by calling [*tail -4 file_name*]
- [*wc file_name*] to obtain the length (default is clmw; we can use options to obtain only number of lines -l, number of words -w, characters -m, bytes -c) of the file called file_name

We can add commands to each other! Some sort of daisy chain of commands on the same line e.g. [*mkdir folder_name && cd folder_name*] will do both creating the folder and moving into it in just one command. We will get error if one of the two (or both) command is not executed.

Scripting in shell: we start by creating an executable file (extension .sh) with [*nano script_name.sh*] and we write the functions we want the shell to execute in the file itself (as if it was a text file). For example, we write [*touch example.txt*] as the body of the executable to create an empty file called example.txt. To check the content of the filewe can use nano (opens in a prompt window), or we can use the function [*cat file_name*] to print to shell (locally to the terminal) the file content. This is good practice before executing, to check what we are about to run. To execute the file we call it in the shell by writing [*bash script_name.sh*]. This will create the file example.txt. To execute we can also call [*./script_name.sh*] (in Windows Git shell); for other OS, use [*chmod +x script_name.sh*] to change the privileges of a file (readable r, writable w, executable by everybody X, executable x, ...; BEWARE to unknown sources), in this case to make it executable, then execute with [*./script_name.sh*].

Downloading material into files: two functions are available, [*wget*] and [*curl*] (wget might not be available depending on distribution; curl should be available to all). By default, [*wget*] saves to file locally a file with the same name as the one linked via the web address, [*curl*] instead only print to screen (to terminal). [*curl web_link*] reads the content of the file that we are linking via the web_link address (we are using the links pasted above at the beginning of this section under the heading *Intro Material*). We can use [*curl web_link > file_name*] to save the output of the web_link to a file called file_name.

- The symbol '>' allows us to create pipelines by redirecting the output of a function call into another function. By default '>' will overwrite the content of the file if we call the same [*curl web_link_new > file_name*] again but with a different web_link file.
- If we don't want to overwrite we have to use '>>' instead, resulting in [*curl web_link >> file_name*]; this will append the new content to the old one. It will create a new file if the file doesn't exist.
- [*curl web_link --output file_name*] is helpful when reading a binary (numerical) file and saving it into a file called file_name
- [*unzip file_name.zip*] (Windows) or [*open file_name.zip*] (Mac, it'll work with any kind of files, looking for the best program to open that file format) to open a zipped file and create a folder containing the extracted files.

Wild cards: we can use '?' or '*' as wild cards in file names, meaning that they stand for any character. For example, we want to remove files called example1.txt, example2.txt, example3.txt, we can call [*rm example?.txt]* where '?' can be 1, 2, or 3 (or any other single character). '*' works with any number of characters, e.g. [*rm *.txt*] will delete all text files in the current folder; [*rm -r *.txt*] will delete all text files in the current folder and any daugther folders (remember -r for recursive).

Exercise - Let's now work with the data folder we downloaded and unzipped and open the [molecules] folder, it contains a number of text files [data-shell/molecules]. Copy all of the files in [molecules] to [molecules-backup] by staying in [molecules-backup] as your working directory. Paste your answer below! Don't be shy :)

cp ~/WD/data-shell/molecules/*.pdb ~/WD/data-shell/molecules-backup
cp -r ../molecules/*.pdb ../molecules-backup
cp ../molecules/*.pdb . ('.' indicates the current folder, where you are now)
cp ../molecules/*.pdb ./ (same as above)

mkdir ~/WD/data-shell/molecules-backup && cd ~/WD/data-shell/molecules-backup && cp ..//molecules/* ./ && ls -al ~/WD/data-shell/molecules-backup

More effective: can be less code, less time, or less memory (less RAM)

Piping: Feed the output of a function/command as input to another function. It uses the symbol '|', called indeed pipe (remember OR earlier?) e.g. [*head -7 methane.pdb | tail -5 > methane-filtered.pdb*] Apply head function to methane (take first 7 lines), its output is given to function tail (take last 5 lines), save result in file called methane-filtered.pdb. Another example [*head -7 methane.pdb | tail -5 | sort -n -k 5 > methane-filtered.pdb]* will do the same operations as before but adding also a numerical sorting of column 5.

Exercise - staying in folder [molecules], create a subfolder [script] and create a script in the folder [WD/script] called [filter.sh]. If possible use only one line of code! Paste your code below! As usual, don't be shy, there is more than one way to do this!

mkdir ~/WD/script && touch ~/WD/script/Filter.sh
mkdir ../../script | touch ../../script/filter.sh
mkdir ~/WD/script && touch ~/WD/script/filter.sh && chmod u+x ~/WD/script/filter.sh
?? mkdir ../../script | nano ../../script/filter.sh ?

[*nano ../../script/filter.sh*] <-- this is an efficient solution (nano is already what we need to create a text file)

For loops: we can use for loops to perform the same operation on many files with different names, where the wild cards are not enough. Let's see an example below
[*for file in *.pdb*
do

- *head -7 $file | tail -5 | sort -n -k 5 > filtered-$file*

done]
for, *do*, *done* are keywords for calling the loop; file is a variable that is created within the loop and temporary store a value for each run of the loop (it is destroyed before the next run), it can have any name and in this case it takes the file names of the molecules (e.g. methane.pdb). Between do and done is the body of the loop, here we specify what we want to execute everytime we run the loop. The '$' is used when we call the variable (file in this case) in the loop body. This for loop will open every .pdb file in the current folder, keep only the central 5 lines (head and tail functions), sort it numerically for the 5th column and save each file to another file called filtered-*file_name.* This works on any number of files, as long as they have the format (pattern) specified by the wild card (in this case *.pdb).
If we wrote this for loop into the [filter.sh] executable we created in the previous exercise, we can perform the described operation by calling [*bash ../../script/filter.sh*]

Building Programs with Python section

Reference webpage(s):
http://swcarpentry.github.io/python-novice-inflammation/

LINKS TO FILES

https://swcarpentry.github.io/python-novice-inflammation/data/python-novice-inflammation-data.zip
https://swcarpentry.github.io/python-novice-inflammation/code/python-novice-inflammation-code.zip

Python is a software (just like nano yesterday) that allows us to perform calculations, write functions, construct pipelines.

Python! How to access? Option 1: run Python from the shell. From the terminal [*python --version*] should return the available version on your machine or use the Anaconda Prompt (Windows users, it depends on your installation, if Python wasn't installed at the system level); to start Python we enter [*python*], we are still in the Shell environment but now Python is running (quick test: 3+3 should return 6!). Option 2: run Python via Jupyter Notebooks. Double click on Anaconda Navigator (in the Application menu of your machine) to open Jupyter Notebooks. Anaconda offers a Python distribution (for free); Jupyter notebook offers a user interface with an interactive shell (but it works on Python exactly like in Option 1) and it allows to add formatted text.
Normally we start with Jupyter to prototype code and to run all the preliminary tests and analyses. We can move to the shell version when the code is written and we need to control a batch analysis.

Start a new notebook in Python 3 now (top right after launching the Notebook)!
Write some calculations (e.g. 3+3) in a shell and press Shift+Enter to obtain the result (execute the content of the cell)

Errors are useful! Python is telling us what we did wrong, don't be scared by the red writing. Python tells us in which line the error is and which kind of error was encountered. How to add comments for future users (or future self)? We can change the cell format to markdown (drop down menu top centre) and this allows to write simply formatted text (https://www.markdownguide.org/cheat-sheet). If we are in a Code cell we can add comments with '#' (whatever is after the # is not going to be executed). We can insert cells above and below with the [Insert] menu.

Variables: variables are fundamental in Python, let's create one called weight_kg and assign value 60 [*weight_kg = 60*]. If we call [*weight_kg*] in a new cell we'll get the value 60 returned. It works with text as well: [*some_text = 'some example text'*], note the apeces (can be 'single' or "double", it works the same). Let's perform some calculations with variables [*force = weight_kg * 9.8*]; this returns nothing, but if we call [*force*] we'll obtain the result [*588.0*]. Naming of variables: can't have spaces (use underscores), can't start with a number

- [*print()*] to print results: [*print('The force is ' + str(force))*] should return [*The force is 588.0*]; we can also use [*print('The force is ', force)*] to let the function print work out the conversion of the data type (note the comma, instead of '+', see below in Strings)

Data types: strings (use apeces around the text), integers (numbers with no values after comma) and floats (numbers with decimals). We need to perform operations on variables of the same type, that's why the following functions can be handy.

- [*str(variable)*] to transform a variable given as input into a string: [*str(force)*] transform force from a number to a string (this allows us to perform string operations on it!).
- [*int(variable)*] to transform a variable given as input into integer: [*int('350')*] will return [*350*]
- [*float(variable)*] to transform a variable given as input into float: [*float('350')*] will return [*350.0*]

Strings: to concatenate text we can use '+', e.g. [*more_text = some_text + 'not sure what to write'*] will

save in more_text 'some example text not sure what to write' (where 'some example text' was the content of the variable some_text]. Strings are not mutable, we can't modify a single element of the string. How to access a specific element of the string given its index? See below in **lists**.

Data structures: they represent collections of variables

- **lists**: [*odds = [1,3,5,7]*] note the squared brakets, we can call the variable with [*odds*] or [*print(odds)*]; it works with text as well [*cities = 'London', 'Rome', 'Berlin', 'Paris', 'Bristol'*], it doesn't matter the number of characters of each string or if there are spaces in the strings. We can also have mixed data type in a list (strings and numbers); you can also have nested lists (so one element of a list can be a list itself)
    - [*list[element_index]*] How can we access elements in the list? Remember OR yesterday. We use squared brakets to indicate the position we want to access: [*odds[0]*] will return 1 (the 0th value of the list, remember Python counts from 0, not for 1), [*odds[1]*] return 3 (the 1st value of the list), [*odds[-1]*] return the last value of the array (7 in this case; it also works with -n, we are counting backwards from the end). The same syntax works to access single characters in a string! Lists are mutable (opposite to strings), we can modify single elements of a list. [*odds[1] = 2*] will not return an error (odds = [1,2,5,7] now).
    - To access a nested list we can use a double call to the squared brakets [*list[index1] [index2]*], provided list[index1] is a list itself, so the element index1'th of the list is a list.
    - [*len(list)*] return the length of the list; it works with strings as well.
    - [*list.append(new_value)*] let us append a new_value to the original list (add at the end) e.g. [*odds.append(9)*] makes odds=[1,2,5,7,9]
    - [*list.insert(index, new_value)*] let us insert a new_value before the index specified e.g. [*odds.insert(-1,'new value')*] makes odds=[1,2,5,7,'new value',9]

For loops: new string variable [city = 'Stockholm'], how can we print each character one at a time? The function we want to use is [*print(city[0])*], [*print(city[1])*], and so on. For loops makes our life easier, let's define it using the following syntax (note the keyword 'for', the colon (':') and the indent before the function body (equivalent to 4x spacebar)):
[*for variable_name in list:*
    (do something - loop body)]
In the city example, this will be written as:
[*for char in city:*
    print(char)]
or
[*for c in cities:*
    print(c)]
This is to demonstrate that the for loop works both on strings or on lists (because remember! strings are just a special kind of non mutable lists)

- how can we add values to a list? (note that we can write comments with # followed by text)
- [new_list = [] # here we are initialising an empty list
- for i in range(0,10):
- new_list.append(i) # here we append the current value i to new_list]
- The variable new_list will now be new_list=[0,1,2,3,4,5,6,7,8,9]
- [*range(start, stop, step)*] https://www.pythoncentral.io/pythons-range-function-explained/ allows to run through a list of numbers from the 'start' number, to the 'stop' (not included) number, using

intervals of 'step'. The options start and step are optional. We use range mainly inside for loops when we want to iterate over numerical indices (it's a special data type, called indeed range, we can't use it as a 'normal' numerical variable).

- It's useful to use [*print*] inside a for loop to evaluate how variables change inside the loop. This is a way to perform 'debugging' (finding mistakes, or bugs, in the code, see definition at https://www.techopedia.com/definition/16373/debugging); we can also use [*print*] to print to screen error messages if an input is not in the format expected, or if it's higher than the expected values, and so on.
- [*enumerate(list)*] https://book.pythontips.com/en/latest/enumerate.html this function allows to obtain both the index and the value of the current element in a list (or string) inside a for loop. Similarly to range, we use this function inside a for loop and not to define numerical variables (another special data type here!).

- Let's see an example:
- [*for index, value in enumerate(string):*
    - *print('Character ' + str(value) + 'is placed at postion ' + str(index))*]
- if string = 'pizza', this command will print:
- Character p is placed at position 0
- Character i is placed at position 1
- Character z is placed at position 2
- Character z is placed at position 3
- Character a is placed at position 4

If loops: what if we want to execute part of the code only if a condition is met? We can use an if loop. [*If*] can have additional calls to [*elif]* (it means else *if*) or [*else*] if we want to add additional conditions (*elif*) or we want to catch all the instances in which the condition wasn't previously met (*else*).
We can use the following syntax (note the keywords 'if', 'elif' and 'else', the colon (':') and the indent before the function body (equivalent to 4x spacebar)):
[*if condition1 is met:*
    (do something - loop body)
elif condition2 is met:

- *(do something else)*

else:

- *(do something else)*]

Let's see an example below:
e.g. [*num = 58*

- *if num < 50:*
    - *print('The number is lower than 50')*
- *elif num == 0:*
    - *print('The number is equal zero')*
- *else:*
    - *print('The number is higher than 50')*]

Try changing the value of num in order to enter different blocks of the code and observe how the print change!

- We can call Boolean operators '*and'* or '*or'* to specify more complex conditions that have to check multiple expressions: e.g. [*if num > 50 and num < 100*] will check that [num] is both higher than 50 and lower than 100 (so num = 58 would enter into this if block, num = 135 would not).

Useful operators:

- '+=' can be used as [a += 3] is equivalent to write [a = a + 3]
- '-=' can be used as [a -= 3] is equivalent to write [a = a - 3]
- '*=' can be used as [a *= 3] is equivalent to write [a = a * 3]
- '/=' can be used as [a /= 3] is equivalent to write [a = a / 3]

Exercise 1 - Revert the word "Great Britain" using a *for loop* in Python to obtain "niatirB tearG"
[*oldstring = 'Great Britain'*
newstring = [] # empty string (initialise)
for char in oldstring:

- *newstring = char + newstring* # put the new char before the string

print(newstring)]

[*oldstring = 'Great Britain'*
newstring = [] # empty string (initialise)
for i in range(1, len(oldstring)+1):

- *newstring += oldstring[-i]*

print(newstring)]

[*oldstring = 'Great Britain'*
newstring = [] # empty string (initialise)
for i in range(len(oldstring)-1,-1,-1):

- *newstring += oldstring[i]*

print(newstring)]

Exercise 2 - Paste your code to the pad once you're done!
You have 2 strings:
1. Great Britain
2. Grrat Britain
Write a for loop to print each character from both strings in sequence

Output should be:
GG
rr
er
aa
tt

BB
...

```
country1="Great Britain"
country2="Grrat Britain"
for index in range(0, min(len(country1),len(country2))):
    print(country1[index], country2[index])
```

OR

```
country1="Great Britain"
country2="Grrat Britain"
for index, value in enumerate(country1):
    print(value, country2[index])
```

Exercise 3 - Paste your code to the pad once you're done!
You have 2 strings:
1. Great Britain
2. Grrat Britain
Compare the 2 strings character by character using a for loop and check if they are equal. Print a message stating if they are equal or not.

```
uk='Great Britain'
uk2='Grrat Britain'
for index, value in enumerate(uk):
    print(value, uk2[index])
    if value==uk2[index]:
        print ('Spelled correctly')
    else:
        print ('Misspelled')
```

```
country1 = "Great Britain"
country2 = "Grrat Britain"
error = "No error"
for index in range(0, max(len(country1),len(country2))):
    try:
        a = country1[index]
    except (IndexError):
        a = "_"
        error = "Different length"
    try:
        b = country2[index]
    except (IndexError):
        b = "_"
        error = "Different length"
    print(a,b)
    if a != b:
        error = "Not the same"
print(error)
```

```
for index in range(0, min(len(country1),len(country2))):
    if country1[index] == ' ' and country2[index] == ' ':
        print(country1[index], country2[index], 'BLANK')
    elif country1[index] == country2[index]:
        print(country1[index], country2[index], 'MATCH')
    else:
```

```
    print(country1[index], country2[index], 'NO MATCH')
```

[uk == uk2] will compare the two strings and return True/False

---

Plotting in Python

Reference webpage(s):
http://swcarpentry.github.io/python-novice-gapminder/ Sections 6-9

For some sample data to import please use this download link:
https://swcarpentry.github.io/python-novice-gapminder/files/python-novice-gapminder-data.zip

Follow the link above and download the zip file containing the gapminder dataset.

To check the current folder you can use the function *pwd* in Python. Move the unzip folder to the current folder provided by pwd in the jupyter notebook. You should be able to access the gapminder folder from the jupyter home page.

We use the function *import* to load libraries into our notebook.

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

What happens if:
```
print('cos(pi) is ', math.cos(pi))
```
we get an error! --> Python doesn't know where to look for pi

Get help:
```
help(math)
```

We can assign a new name to the library

```
import math as m
print('cos(pi) is ', m.cos(m.pi))
```

However, be careful with the use of aliases, because they reduce the readability of code

We can import one or a few items from a library, not all of it
```
from math import cos,pi
```

This time we are importing only cos and pi from the library math. We don't need this time to prepend *math.*

```
from math import cos,pi
print('cos(pi) is ', cos(pi))
```

Exercise

Fill the blanks

```
import math as m
angle = _____.degrees(____.pi/2)
print(____)
```

The output should be 90

Solution

```
import math as m
angle = m.degrees(m.pi/2)
print(angle)
```

```
import pandas as pd

data = pd.read_csv('python-novice-gapminder-data/data/gapminder_gdp_oceania.csv')
```

You can use the autocomplete function with *tab* to help you navigating file names available in the current folder

By writing:

```
data
```

we get the content of the file csv formatted as a table

```
data = pd.read_csv('python-novice-gapminder-data/data/gapminder_gdp_oceania.csv',
index_col='country')
```

index_col defines which column you want to use as index for your table, in this case 'country'

By importing the file in the Python workspace using Pandas we created a Pandas dataframe, which you can think of as a spreadsheet on "steroids"

```
data.info()
```

gives us information about the dataset

By using

```
data.T
```

we can transpose the dataset, i.e. columns become rows and rows become columns. data.T is NOT changing the content of the variable data, which contains the dataset. To save it we need to store it in a new variable

```
data.describe()
```

provides basic statistics of the datasets.

Exercise

Retrieve the dataset for the Americas and store it in the variable *americas* and then compute basic statistics of this subset.

Solution

```
americas = pd.read_csv('python-novice-gapminder-data/data/gapminder_gdp_americas.csv',
index_col='country')
americas.describe()

americas.head()
```

prints the first 5 rows of the dataset, but we can specify the number of rows

```
americas.head(3)

americas.tail()
```

prints the last 5 rows.

```
americas.sample()
```

provides 1 random row from the dataset, but we can specify the number of rows

```
americas.sample(6)

americas.to_csv('processed.csv')
```

saves the dataset americas to the file processed.csv in the current folder, as we didn't specify any absolute or relative path.

```
import pandas as pd
data = pd.read_csv('python-novice-gapminder-data/data/gapminder_gdp_europe.csv',
index_col='country')
data.head()

data.iloc[0,0]
```

provides element in row 0, column 0 of the dataset

```
data.loc['Albania','gdpPercap_1952']
```

provides element in row labelled *Albania*, column labelled *gdpPercap_1952*

```
data.loc['Albania','gdpPercap_1982':]
```

provides elements in row labelled *Albania*, in columns labelled *gdpPercap_1982* onwards

```
data.loc['Albania',:'gdpPercap_1982']
```

provides elements in row labelled *Albania*, in all columns until the one labelled *gdpPercap_1982*

```
subset = data.loc['Italy':'Poland', 'gdpPercap_1962':'gdpPercap_1972']
```

the column operator *:* allows us to *slice* the dataset

```
subset.min()

subset > 10000
```

```
mask = subset > 10000
subset[mask]
```

shows only the elements of subset for which mask evaluates to TRUE

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

Exercise
- Show the gdp of all countries in Europe in 1982
- Show the gdp of Denmark for all years
- Show the gdp of all the countries in Europe for all the years after 1982

Solution

```
data.loc[:,'gdpPercap_1982']
data.loc['Denmark',:]
data.loc[:,'gdpPercap_1982':]
```

%matplotlib inline ----> this line is specific to the Jupyter notebook! it is not a general Python command (it allows the plot in the notebook cells)

```
import matplotlib.pyplot as plt

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

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

import pandas as pd
data = pd.read_csv('python-novice-gapminder-data/data/gapminder_gdp_oceania.csv',
index_col='country')
data.loc['Australia',:].plot()
```

here we used the plotting capabilities of the pandas library

```
australia = data.loc['Australia',:]
years = data.columns.str.strip('gdpPercap_').astype(int)
# this command take the columns of the database (data.columns), transform them into strings (str)
# and removes from all of them the first part of the name (str('gdpPercap_')),
# so that we are only left with the years (which we transform into integers (astype(int))
plt.plot(years,australia, 'g--')
plt.xlabel('Years')
plt.ylabel('GDP per capita')
```

here we used the plotting capabilities of the matplotlib library

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

australia = data.loc['Australia',:]
```

```
newzealand = data.loc['New Zealand',:]
years = data.columns.str.strip('gdpPercap_').astype(int)
plt.plot(years,australia, 'g--',label='Australia')
plt.plot(years,newzealand, 'b*',label='New Zealand')
plt.xlabel('Years')
plt.ylabel('GDP per capita')
plt.legend()
plt.title('Plot')
fig = plt.gcf()  ---> gcf = get current figure
fig.savefig('myplot.png')
```

the plot is saved to a png file called myplot.png in the current working folder

---

Advanced Python

- how to get help! try [*help(command)*] (remember if a function is inside a library, you'll have to call it with library.function, like math.pi in library math); also you can check the Python documentation on Google (https://docs.python.org/3/index.html)
- Why do we comment our codes? https://www.c-sharpcorner.com/blogs/why-comments-are-important-while-writing-a-code
- What do we mean when we say "best practice"? Simply put, be kind to your future self or to different users that will have to read your code in the future! There are some acutal guidelines (see the link below), but let's say it is some kind of coder's *common sense* https://en.wikipedia.org/wiki/Best_coding_practices
- Functions: let's have a look at the syntax, check the keyword 'def', the colon ':' and the indent
- [*def function_name(input):*
  - ''' Function description (this is a special comment, see below)
  - *output = (something something) # can be many lines, it includes all the operations you want to perform*
  - *return output*]
- All the variables defined inside the functions (included input) are local! They are not saved anywhere unless we don't call the function and we assign them to a global variable (stored in the notebook itself). To call a function you can write [*function_name(input)*], to store its value into a variable called temp we can write [*temp = function_name(input)*]
- e.g. let's write a function to transform Farhenheit degrees to Celsius
- [*def fahr_to_celsius(temp_F):*
  - ''' This function provides the temperature in Celsius given the temperature in Farenheit'''
  - *temp_C = ((temp_F-32)*(5/9))*
  - *return temp_C*
- *temp = fahr_to_celsius(100) # assign output for input = 100 to variable temp*]
  - If we want to display some help for our function if called ([*help(function_name)*]), we have to start the comment that we want to display with ['''] in the body of the function (see example, it's optional!). Good practice is to specify what the function does, what are the input and output, if there are any dependancies (do you use functions from other libraries?)
  - We can plug the call to a function inside another function (nesting)
- Let's put altogether now! Pandas and functions!

- [*def load_gapminderdata(continent):*
  - *''' This function loads data from gapminder datasets*
  - *Continent is a string indicating the continent we want to load the dataset of'''*
  - *dataDirectory = 'python-novice-gapminder-data/data/' # the folder where our tabular data is*
  - *baseFilename = 'gapminder_gdp_' # the base filename of all files*
  - *ext = '.csv' # file extension*
  - *data = pd.read_csv(dataDirectory + baseFilename + continent + ext, index_col = 'country')*
  - *return data*
- *gdData = load_gapminderdata('europe')*]

Exercise
Write a function celsius_to_kelvin that converts a T in Celsius to Kelvin (T_K = T_C+273.15).
Write a function fahr_to_kelvin that converts a T in Fahrenheit to Kelvin.
Provide a description of each function that can accessed through the help()
Paste the solution here

```
def celsius_to_kelvin(temp_C):
    '''This function provides the temperature in Kelvin given the temperature in Celsius'''
    temp_K = temp_C+273.15
    return temp_K
celsius_to_kelvin(100)

def celsius_to_kelvin(temp_C):
    temp_K = (temp_C+273.15)
    return temp_K
celsius_to_kelvin (100)
-
def fahr(temp_F):
    temp_K = (temp_F+ 459.67)*5/9
    return temp_K
celsius_to_kelvin (100)

def celsius_to_kelvin(Temp_C):
    Temp_K = Temp_C + 273.25
    return(Temp_K)



def celsius_to_kelvin(temp_C):
    '''This function provides the temperature in Kelvin given the temperature in Celcius'''
    temp_K = temp_C+273.15
    return temp_K
def fahr_to_kelvin(temp_F):
     '''This function provides the temperature in Kelvin given the temperature in Fahrenheit'''
    temp_K = (temp_F-32)*(5/9) + 273.15
    return temp_K
```

```
    def cel_to_kelv(temp_C):
    '''This function calculates temperature in kelvin from celsius'''
    temp_K = (temp_C+273.15)
    return temp_K

    def celcius_to_kelvin(temp_C):
    T_K = ((temp_C+273.15))
    return T_K
    celcius_to_kelvin(80)
```

Exercise 2:

```
def fahr_to_kelvin(Temp_F):

    Temp_C = ((Temp_F-32)*(5/9))
    Temp_K = Temp_C + 273.25
    return(Temp_K)

def fahr_to_kelvin(temp_F):
    temp_C = ((temp_F-32)*(5/9))
    T_K = ((temp_C+273.15))
    return T_K
fahr_to_kelvin(100)

def fahr_to_kelvin(temp_F):
'''This function converts temperature values from the Fahrenheit scale to outputs in the Kelvin scale.'''
    temp_K = (((temp_F-32)*(5/9))+273.15)
    return temp_K
fahr_to_kelvin(0)

def conv_F_to_C(temp_F):
    '''This function is a temperature convertor
                Parameters:
                    Float temp_F

                Returns:
                    Float temp_C

                    Errors:
                    Dependencies:
    '''
    temp_C = ((temp_F - 32) * (5/9))
    return temp_C

def conv_C_to_K(temp_C):
    temp_K = temp_C + 273.15
    assert temp_K >= 0 , 'Impossible Value'
```

```
    return temp_K

def conv_F_to_K(temp_F):
    temp_K = conv_C_to_K(conv_F_to_C(temp_F))
    return temp_K

print(conv_C_to_K(0))
print(conv_C_to_K(-300))
print(conv_F_to_C(90))
print(conv_F_to_K(65))
```

Debugging

Finding errors in the code (i.e., debugging) is very hard, especially when you have been looking at it for a long time. Sometimes, just standing up, going for a coffee, short walk (or sleep over it!) will help you in finding where the problem is. If this doesn't work, you can try to explain the problem to a colleague, and if no colleague is available you can always rely on a rubber duck (no joking: https://en.wikipedia.org/wiki/Rubber_duck_debugging): just talking about the logic behind your code/function may help you to spot the mistake. Also, doing a pen and paper simulation of your code line-by-line may be very helpful (especially if you code is doing something that looks weird).
If also this doesn't work, you can Google the error, or post in some specialised forum (e.g., https://stackoverflow.com/, but do not expect everyone to be kind there). Finally, you can always try to write the same piece of code from scratch without looking at the old one, hoping of avoiding the error.
Of course, don't forget to look for misspelled functions/variables :)

How did you find out about the Software Carpentries "Introduction to Python" Workshop?

(Thanks in advance!)

Via the Modern Languages mailing list
Via personal email
Department newsletter (which dept? - NR)
History Department mailing list +
KCL website+
Twitter+
Via the Modern Languages mailing list
Arts & Humanities Events Bulletiin Newsletter Thing
Arts & Humanities PGR mailing list +++

Flyer (4th floor New hunts house, Guys Campus) + 1
via Arts&Humanities PGR mailing list