Writing your own python modules ¶

You will soon realize that for every project there are always a few lines of code that end up being extremely helpful and handy to keeparound. These lines end up being the ones applied multiple times and for multiple purpuses.

Repeating operations and reusing lines of code is key to programming.

In this tutorial we will learn how to write python modules (https://docs.python.org/3/tutorial/modules.html). A module is nothing more than a file (ending in `.py`) containing collection of functions. A module can be as simple as containing a few functions, or as complicated as `numpy` or `seaborn` .

We have learned so far how to write functions. Functions are a handy way to reuse the same lines of code.

As the data science projects become more complex, or you become more expert at data science projects, the number of functions that end up needing to be carried around can grow fast.

For any sizable project, the number of functions needed to be kept around is larger than the number of functions we are willing to copy and paste in every new script or jupyter notebook.

To avoid copying an pasting dozens of functions we can use python modules. Modules are collections of functions (and other python assertions, such as variables definitions) in a file saved on the current path accessible to `python` .

Just like functions facilitate reusing dozens of lines of code, modules facilitates reusing dozens of functions.

Learning goals:

- Understanding Python Modules
- Practice building Python Modules
- grouped data: aggregation and pivot tables

Our first module

Python offers a convenient way to keep useful code and functions around by writing and importing modules.

What is a module? Python modules are libraries of functions. We have encountered modules all along our tutorials. Indeed everytime we were invoking an `import` statement we were effectively loading a module.

How is a module defined? A module is a python file (ending with exstension `.py`) with a series of functions definitions (i.e., statement starting with `def`) they live in the current path where your python code isrunning and because of that it can be imported.

How does a module work? Python allows importing any `.py` file containing `def` statements. Importing a module file makes the functions in the file callable and usable (for example in Jupyter notebook).

How to write a python module

Let's learn how to write and use Python modules! (we will start simple.)

In a nutshell, the general process to write and use a python modules can be summarized as follows:

To write a module we need to: A) Create a file with exstension `.py` . B) Write functions inside the file. C) Save the file on the path accessible to python (for simplicity say the current working directory).

To use python modules we need to: A) Make sure the module is in the current working directory. B) import the module by typing `import and <moduleName>` C) Call the functions in the module withthe syntax `moduleName.functionName`

MyModule

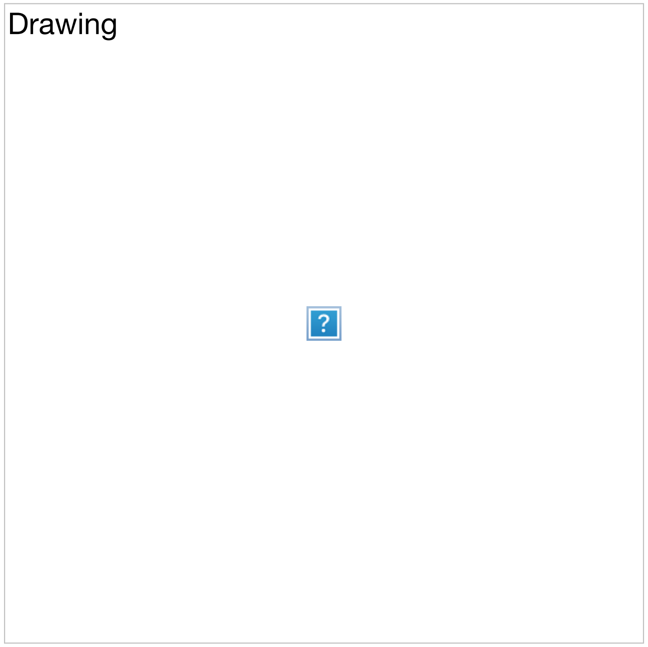
Hereafter, we will practice with the process described above. Write a module and then import and use the module.

This means that we will write a file outside this jupyter notebook. This is something we have not done before and might feel a bit awkward (are we really leaving our safe `Jupyter Notebooks` heaven? Yes).

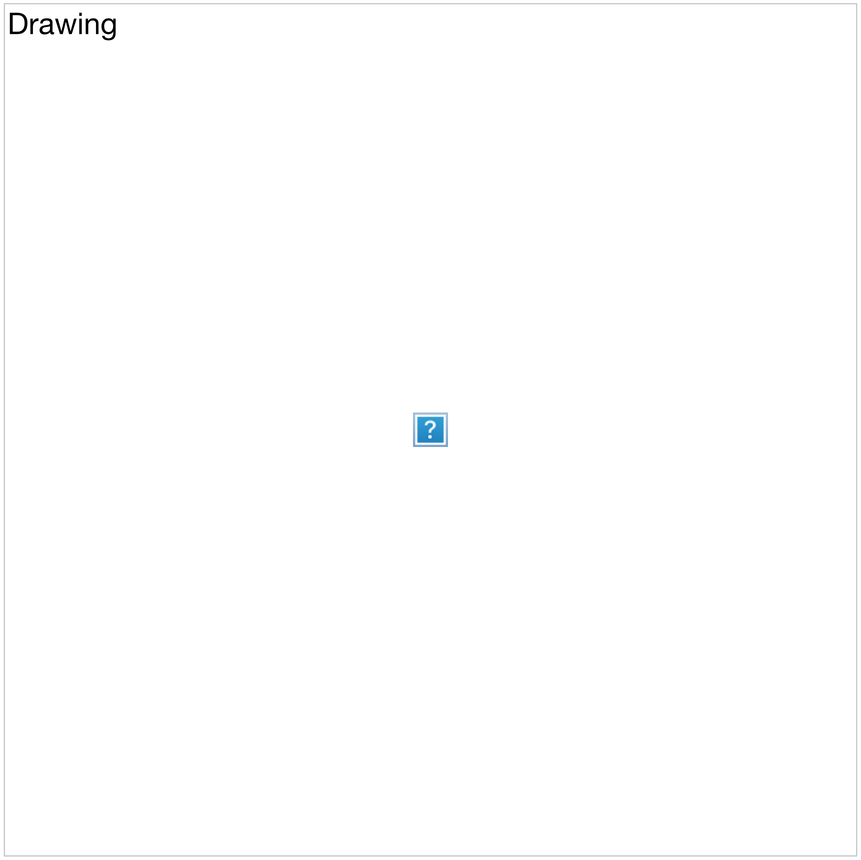
Just as a heads start, our module will be called `mymodule`. The module will contain a function that will print the first few words of [Billie Eilish's song "Ocean Eyes" (https://www.youtube.com/watch?v=viimfQi_pUw&ab_channel=BillieEilish)](https://www.youtube.com/watch?v=viimfQi_pUw&ab_channel=BillieEilish).

So, to learn how to create a module, we will perform the following exercise.

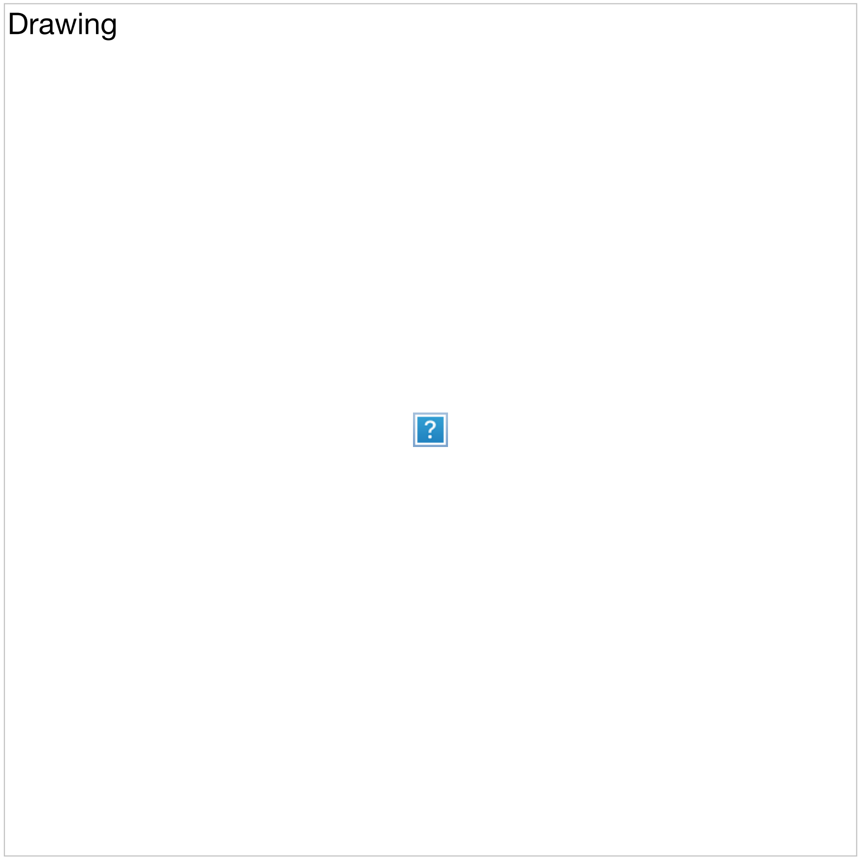
- Open a new Jupyter Notebook (from the File menu, select "New Notebook"



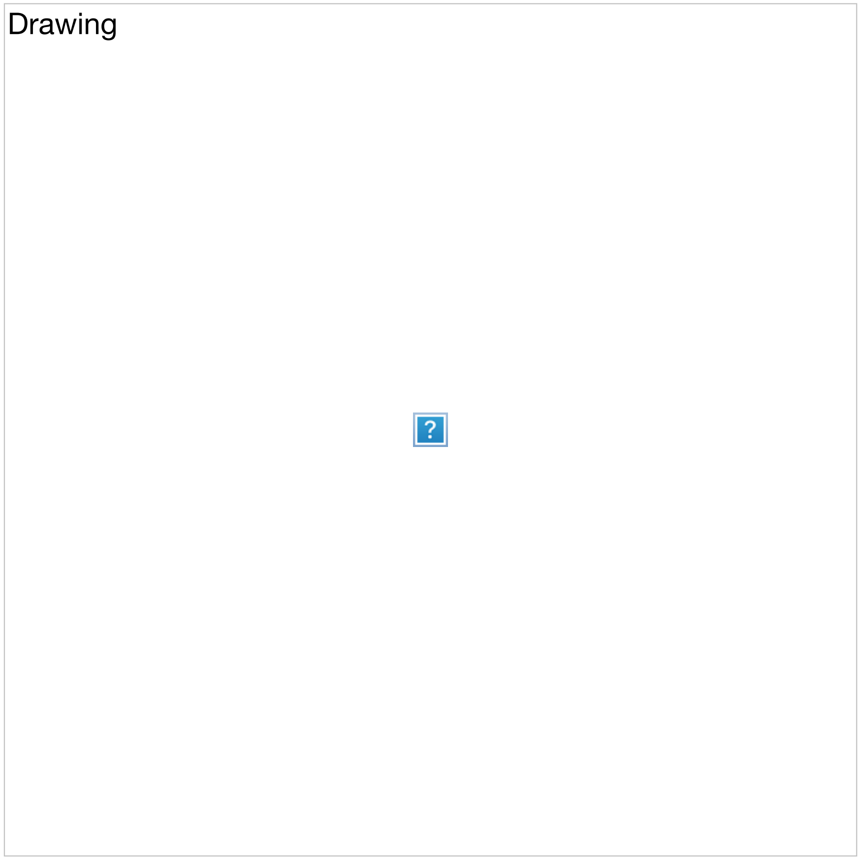
- Edit the name of the new notebook and rename it from "untitled" `mymodule`



- Copy and paste the code for the function provided below (`OceanEyes`) into the `mymodule` Jupyter Notebook. Note. Only create a single cell in the new notebook. Make sure no other cell is there.



- Download the `mymodule` notebook into the current directory with the `.py` file exstension. To do so, from the File menu navigate to "Downloads as" and select the file type "Python *.py*."



- Save the file in the same directory of the current tutorial.

In [29]:

```
def OceanEyes():
    print('Can''t stop starin'' at those ocean eyes')
```

Alright, after following the instructions above, and if all went well, we should be ready to load the module and use its function.

To load the module we will tell python to import it. This is a simple as running the following statements:

In [30]:

```
import mymodule
```

If the previos cell executed withouterrors the module is loaded! (If error were returned, please read the errors and try to repeatthe previous steps.)

Next, let's use the module! The module we created will "only" print the first few words of a song. But let's it.

Our module is called is just like any other modules we have used before. For example, we have used `Pandas` , and `Numpy` , those are also modules.

So, let's take a look at our syntax! Our module is called `mymodule` and the function it contains `OceanEyes` so the call goes as follows

In [31]:

```
mymodule.OceanEyes()
```

```
Cant stop starin at those ocean eyes
```

Did you get it? Did you get the words from the song? If you did, congratulations you just wrote a python module.

More complex modules just contains more functions, more complex functions etc. Butthe process (given what we have covered so far) can be summarized as above.

Complete the following exercise.

- Make your module:
  - Pick a song you like
  - Make a new python module that is called with the first two words in the title of the song
  - When invoked, the module should print the first phrase of the song you picked
  - Import the module and show it works

In [32]:

```
import High_Hopes
```

In [33]:

```
High_Hopes.HighHopes()
```

```
Had to have high, high hopes for a living
```

More about modules

Note now that, the file name is also the name of the module (`mymodule`). The file name has the suffix `.py` appended, that suffix is not used in the code, when calling the module (in other words we do not `import mymodule.py` but we `import mymodule`).

The name for modules imported in the current workspace is alwasy available as the value of the global variable **name** (a string).

We can extract themodulename into a string as follows:

```
In [34]: mymodule.__name__ #a built-in variable which evaluates to the name of the current module.
Out[34]: 'mymodule'
```

Just like we have done in the past with Pandas and Numpy also our module can be imported with a different (shorter) name:|

```
In [35]: import mymodule as mm
```

Now the function in mymodule should be called using `mm` , give it a try:

```
In [36]: mm.OceanEyes()
Cant stop starin at those ocean eyes
```

Functions inside a module can be imported directly and assigned a callable name.We have seen this before ...

```
In [37]: from mymodule import OceanEyes as oe
```

Now we can call the function directly, avoiding the sintax `mymodule.<functionName>` . Try the following, it should work:

In [38]:

```
oe()
```

```
Cant stop starin at those ocean eyes
```

Let's break this

OK now let's try something that should breakthings for us, but perhaps also help usunderstand. Move the file `mymodule.py` out of the current directly, for example, move it to your desktop instead.

After doing that try importing the module again.

In [39]:

```
import mymodule
```

Did that work? Why?

Yes, I think it works because we use `.__name__` function, which imprint `mymodule` into this jupyter notebook.

Modules can import other modules.

It is possible to add import operations inside a module. Say for example you want to load `Numpy` every time you load your module. You could add `import numpy as np` at the beginning of your module and the module will automatically add numpy to your current workspace as soon as you call your module.

The standard modules in Python

Python comes with a library of modules called standard: The python standard modules lbrary (https://docs.python.org/3/library/). These modules are shipped with the Python3 distribution. This means that you can simply import them without saving, or moving files. The files are pythonmagically there for you.

A list of standard modules can be found here (https://docs.python.org/3/py-modindex.html). The lis

In sum

Write python modules is as easy as writing a file ending withthe `.py` exstension. The file should contain function definitions. The file could also contain variables definitions or other code statements, an aspect of modules that we have not experimented with in this tutorial.

Make the best rat lab module

To practice with modules we will make an exercise and make a module out of the code from a previous tutorial.

Your goal will be to take these functions sve a the module and demonstrate that it runs from within this jupyter notebook

First of all we will break down the code into the basic steps and make one function per step. After that, we will make a module, save it to disk and call it to use the function.

Let's get started.

In a previous tutorial, we loaded data from files given to us from a lab and performed a series of operations to reorganize the data into a Tidy Data Format. In that tutorial (Tutorial 17 using 'datasets/017DataFile.csv') we performed four independent operations to reorganzie the data.

- We loaded reaction time data into a specific format.
- We organized the labels for the strains of rats into the appropriate format for the data.
- We organized the labels for the sexes of rats into the appropriate format for the data.
- We combined the data and labels into a tidy format (one colum per variable/label)

Below we have four functions written to implement the operations described above and used in the previous tutorial. These functions can now be conveniently called multiple times within this Jupyter notebook. Yet, to call the functions in a new notebooks, or in future (many) notebooks, they must be copied and pasted into each new Jupyter notebook. Boring...

Wouldn't it be easier if we could call them directly from a module? Let's do this.

Below we first describe how we functionalized the code from the previous tutorial. We describe each function and what it does and then use them after loading the data.

After that, we will open a new notebook and save it as a module. We will then repeat the data processing performed with the functions by loading the module we just created.

In [ ]:

```
def get_data(filename) :
    '''
    get_data()
    Loads the data from a filename.
    Organizes the data and retunrs key data values
    '''
    import numpy as np
    import pandas as pd

    my_input_data = pd.read_csv(filename)  # read the data

    raw_data   = my_input_data.to_numpy()                    # convert to numpy array
    obs, grps  = raw_data.shape                              # get the number of rows and co
    new_length = obs*grps                                    # compute total number of obser
    values_col = np.reshape(raw_data, (new_length, 1),
                            order = 'F')                     # reshape the array
    values_col = np.squeeze(values_col)                      # squeeze to make 1D

    return values_col, obs
```

In [ ]:

```python
def get_strains(obs=10, names=['wildtype', 'mutant']) :
    '''
    get_strains()
    Takes names of rat types (e.g., names=['wildtype', 'mutant']) and
    the number of observation per group (obs_per_grp=10).
    Returns the variable `strain` containing.
    User specifies a filename string.
    '''
    import pandas as pd

    strain = pd.Series(names)                   # make the short series
    strain = strain.repeat([2*obs])     # repeat each over two cell's worth of data
    strain = strain.reset_index(drop=True)      # reset the series's index value

    return strain
```

In [ ]:

```python
def get_sexes(obs, sexLabels=['male', 'female']) :
    '''
    tidyMyData() Takes one-column-per-cell rat reaction time data as input.
    Returns tidy one-column-per-variable data.
    User specifies a filename string.
    '''
    import pandas as pd

    sexes = pd.Series(sexLabels)                      # make the short series
    sexes = sexes.repeat(obs)                 # repeat each over one cell's worth of data
    sexes = pd.concat([sexes]*2, ignore_index=True)   # stack or "concatonate" two copies

    return sexes
```

In [ ]:

```
def tidy_data(values_col,strain,sexes) :
    '''
    tidyMyData() Takes
    1. A one-column-per-cell rat reaction time data (values_col).
    2. A sexes variables labelling each entry in values_col by rat-sex
    3. A strain variable labelling entries in values_col by rat strain

    Returns one-column-per-variable data adhering to the tidy format.

    '''

    import pandas as pd

    # construct the data frame
    my_new_tidy_data = pd.DataFrame(
        {
            "RTs": values_col,                               # make a column named RTs and pu
            "sex": sexes,                                    # ditto for sex
            "strain": strain                                 # and for genetic strain
        }
    )

    return my_new_tidy_data
```

Complete the following exercise.

- Your goal is to make a module called `bestratlab.py` out of the above functions and to demonstrate that it can run from this notebook.

```
In [1]:  import bestratlab
```

```
In [2]:  bestratlab.get_data('datasets/017DataFile.csv')
```

```
Out[2]: (array([10.48545088, 11.74794775, 13.41258004, 12.91009526, 10.36777045,
                11.69842177, 11.58315277, 11.44734892, 10.85227619, 11.28589742,
                 8.2500131 ,  8.45383932,  9.70660484,  9.52211638,  8.58321246,
                 9.83500171, 10.53209602,  9.39416641,  8.73947266, 10.89239399,
                20.12706278, 20.06814699, 21.21514789, 20.70641578, 18.07479515,
                20.36762403, 20.15252058, 19.39247581, 18.52434071, 20.32502629,
                25.94638414, 23.46487013, 22.98948034, 25.32437595, 22.60748688,
                23.05218737, 25.3690367 , 23.37270897, 25.21564644, 24.99050453]),
         10)
```

```
In [3]:  bestratlab.get_strains(obs=10, names=['wildtype', 'mutant'])
```

```
21        mutant
22        mutant
23        mutant
24        mutant
25        mutant
26        mutant
27        mutant
28        mutant
29        mutant
30        mutant
31        mutant
32        mutant
33        mutant
34        mutant
35        mutant
36        mutant
37        mutant
38        mutant
39        mutant
dtype: object
```

```
In [4]: bestratlab.get_sexes(obs = 10, sexLabels=['male', 'female'])
```

```
Out[4]: 0         male
        1         male
        2         male
        3         male
        4         male
        5         male
        6         male
        7         male
        8         male
        9         male
        10      female
        11      female
        12      female
        13      female
        14      female
        15      female
        16      female
        17      female
        18      female
        19      female
```

In [10]:

```python
def get_data2(filename) :
    '''
    get_data()
    Loads the data from a filename.
    Organizes the data and retunrs key data values
    '''
    import numpy as np
    import pandas as pd

    my_input_data = pd.read_csv(filename)  # read the data

    raw_data   = my_input_data.to_numpy()                    # convert to numpy array
    obs, grps  = raw_data.shape                               # get the number of rows and co
    new_length = obs*grps                                     # compute total number of obser
    values_col = np.reshape(raw_data, (new_length, 1),
                            order = 'F')                     # reshape the array
    values_col = np.squeeze(values_col)                       # squeeze to make 1D

    return values_col
```

In [13]:

```python
values_col = get_data2('datasets/017DataFile.csv')
values_col
```

Out[13]:

```
array([10.48545088, 11.74794775, 13.41258004, 12.91009526, 10.36777045,
       11.69842177, 11.58315277, 11.44734892, 10.85227619, 11.28589742,
        8.2500131 ,  8.45383932,  9.70660484,  9.52211638,  8.58321246,
        9.83500171, 10.53209602,  9.39416641,  8.73947266, 10.89239399,
       20.12706278, 20.06814699, 21.21514789, 20.70641578, 18.07479515,
       20.36762403, 20.15252058, 19.39247581, 18.52434071, 20.32502629,
       25.94638414, 23.46487013, 22.98948034, 25.32437595, 22.60748688,
       23.05218737, 25.3690367 , 23.37270897, 25.21564644, 24.99050453])
```

In [15]:

```
strain = bestratlab.get_strains(obs=10, names=['wildtype', 'mutant'])
strain
```

Out[15]:

```
0     wildtype
1     wildtype
2     wildtype
3     wildtype
4     wildtype
5     wildtype
6     wildtype
7     wildtype
8     wildtype
9     wildtype
10    wildtype
11    wildtype
12    wildtype
13    wildtype
14    wildtype
15    wildtype
16    wildtype
17    wildtype
18    wildtype
19    wildtype
```

In [17]:

```
sexes = bestratlab.get_sexes(obs = 10, sexLabels=['male', 'female'])
sexes
```

Out[17]:

```
0       male
1       male
2       male
3       male
4       male
5       male
6       male
7       male
8       male
9       male
10    female
11    female
12    female
13    female
14    female
15    female
16    female
17    female
18    female
19    female
```

In [18]:

```
bestratlab.tidy_data(values_col,strain,sexes)
```

Out[18]:

	RTs	sex	strain
0	10.485451	male	wildtype
1	11.747948	male	wildtype
2	13.412580	male	wildtype
3	12.910095	male	wildtype
4	10.367770	male	wildtype
5	11.698422	male	wildtype
6	11.583153	male	wildtype
7	11.447349	male	wildtype

8	10.852276	male	wildtype
9	11.285897	male	wildtype
10	8.250013	female	wildtype
11	8.453839	female	wildtype
12	9.706605	female	wildtype
13	9.522116	female	wildtype
14	8.583212	female	wildtype
15	9.835002	female	wildtype
16	10.532096	female	wildtype
17	9.394166	female	wildtype
18	8.739473	female	wildtype
19	10.892394	female	wildtype
20	20.127063	male	mutant
21	20.068147	male	mutant
22	21.215148	male	mutant
23	20.706416	male	mutant
24	18.074795	male	mutant
25	20.367624	male	mutant
26	20.152521	male	mutant
27	19.392476	male	mutant
28	18.524341	male	mutant
29	20.325026	male	mutant
30	25.946384	female	mutant
31	23.464870	female	mutant
32	22.989480	female	mutant

33	25.324376	female	mutant
34	22.607487	female	mutant
35	23.052187	female	mutant
36	25.369037	female	mutant
37	23.372709	female	mutant
38	25.215646	female	mutant
39	24.990505	female	mutant

A note on recycling code

We have learned early in our journey towards Data Science that it is convenient to keep helpful code around and recycle it. So far, we have learned of at least three ways to recycle code:

- *Loops.* Loops facilitate reusing hundreds of operations. Loops allow repeating the same operations over and over avoiding actually copying and pasting the same lines of code.
- *Functions.* Functions facilitate reusing hundreds of lines of code. Functions allow reusing the same lines of code for different instances ofthe same situation.
- *Modules.* Modules allow facilitates reusing hundreds of functions. Modules provide a convenient way to save good work, functions, in an accessible file. Module files can be loaded the, or better imported in the current working python stack and that allow accessing and using the functions saved in the module.