This course has already ended.

« Libraries and Packages Course materials Exercises »

CS-E4002 / Using Data in Python / Importing Package Functions

Importing Package Functions

1. Installing and importing those libraries:

Installing the different libraries is explained in the linked documentation: make sure to pick the method that fits your Python installation.

Importing libraries or functions can be done in different ways:

- The most common being **import library_name**.

```
import numpy
import pandas
```

- If the library is used a lot, or if the name is really long, it can be shortened to simplify the call of functions. Note that some libraries have conventional shortened names that are used by the vast majority of people using Python. Make sure to follow the conventions so that your code can be easily understood. The name can also be changed without necessarily shortening it to match the current use and make the code more understandable.

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

- Finally, if a function from a library is used very often and the library is otherwise rarely used, a single function or a single class can be imported with: **from library_name import function_name**.

```
from pandas import DataFrame
from numpy import linspace
from numpy.random import randint
```

2. Define a Python variable from the dataset:

Once the libraries are installed and imported, and once the type of the file we want to use is known, it is very easy with pandas to define a Python variable from the file, using one of the functions read_excel, read_csv, read_json, ...

These functions are from the pandas library, so they are called using **pandas.read_csv**, or **pd.read_csv** if the name was shortened on import.

```
my_df = pd.read_csv('my_directory/my_file.csv')
```

Calling one of these functions creates a DataFrame, which is a data type defined in pandas.

```
.   col1  col2
0     1     3
1     2     4
```

A DataFrame is, roughly speaking, a two dimensional table (that can be heterogeneous). The different attributes and main functions are listed in the documentation, but let us notice the head() method that plots the n first lines of the table, and will allow you to get a first nice and comprehensive view of your data. Note that the method is directly called "on the object" e.g. **my_dataframe.head()**.

```
>>> df.head(2)
    animal
0      cat
1      dog
2    hamster
```

3. Important note:

When you need to do something specific with Python, before starting to implement your own functions right away, maybe check if a package already exists for it. Note that external libraries (besides the ones explicitly named in this course) cannot be used for the course exercises. If you find a package that could help you, you need to check the following things:

- It has all the functions and functionalities you require, so you won't have to install a gazillion packages to do what you want (too many dependencies is bad, as people will struggle using your code)
- Getting familiar with the library won't take you more time than to implement the thing directly (even though library functions are very often optimized, and so faster and more efficient memory wise),
- Check ratings (it is a bad sign if you can't find anyone who used the package before you) to make sure it works well and can be trusted.
- Check that there are not too many dependencies on packages you don't have.
- Check that it is well documented, with lots of examples and the documentation is easily navigable. If the documentation of a library is awful, using it might very quickly become time consuming and ineffective.

Once you decide which one you want to use, make sure:

- You import it right and use the documentation all along the way.
- Mention in your own code documentation which libraries you use, why and how to install them. Indeed, someone using your code needs to know which libraries to install and how (and this person can be you, a few months in the future).

WARNING: we will be talking a lot about modifying a dataset but, through this entire process, it is possible to make mistakes. Also, it is possible that later on, you want to use some data that you got rid of in the first place because you didn't need it then. To avoid losing precious data (that can sometimes be difficult to gather), it is important to keep an untouched backup copy of the original dataset, in a separate directory. Then, make a copy of it in a new directory and work on this one to clean and unify it. Finally, once the data is cleaned and unified, you don't want to lose all this hard work, so it is important to create a third directory, with a copy of the clean and unified data you will actually work on! So, working on a specific dataset, you should have **three different directories**: one with the original dataset, one with the cleaned and unified dataset and one working directory.