Day 03 – Piscine Python for Data Science

Intro to Python: Package management and virtual environment

Summary: This day will help you to get the basic knowledge about how to manage libraries in Python and work with virtual environments.

Contents

Chapter I

Preamble

10 library rules:

1. Use a level 0-1 voice.

2. Use a shelf marker.

3. Turn the pages from the top corner.

4. Wash your hands before touching a book.

5. Return books on time.

6. Never eat or drink while reading.

7. Keep books dry.

8. Use a bookmark.

9. Do not write or draw in books.

10. Keep books away from babies and pets.

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.

- Here and further we use `Python 3` as the only correct version of Python.

- The python files for python exercises (d01, d02, d03) must have a block in the end:
  `if __name__ == '__main__'`.

- Pay attention to the permissions of your files and directories.

- To be assessed your solution must be in your GIT repository.

- Your solutions will be evaluated by your piscine mates.

- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.

- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.

- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.

- Your reference manual: mates / Internet / Google.

- Remember to discuss on the Intra Piscine forum.

- Read the examples carefully. They may require things that are not otherwise specified in the subject.

- And may the Force be with you!

Chapter III

Specific instructions of the day

- No code in the global scope. Use functions!

- Each file must be ended by a function call in a condition similar to:

```python
if __name__ == '__main__':
    # your tests and your error handling
```

- Any exception not caught will invalidate the work, even in the event of an error that was asked you to test.

- No imports allowed, except those explicitly mentioned in the section "Authorized functions" of the title block of each exercise.

- The interpreter to use is Python 3.

Chapter IV

Exercise 00

Exercise 00 : Virtual Environment
Directory to store your solution : `ex00/`
Files to be in the directory : `venv.py and folder with your virtual env`
Authorized functions : `import os`

Libraries, or in other words packages, are one of the reasons, why coding has been democratized. It is never been easier than now to learn to code and get quick results from this process. Some other programmers wrote pieces of code that can be reused by other coders. And many of these libraries in Python are open-sourced which means everybody can use them. Nobody needs to write already existing classes, methods, functions from scratch. You can reuse them. What you just need to do is `sudo pip install`. Or wait…

This way of installing Python packages is considered as a bad practice. When you do it in the way described above, you install them in the system version of Python. And Python exists on your machine not only to give you the power to code but to run some essential programs for the system. By installing external packages like that you may ruin your system. So almost never do `sudo pip install`.

There is a better way – virtual environments. Think about it as your own little sandbox where you can do whatever you want. If you ruin something, you ruin it only inside this sandbox. Your machine should have a preinstalled package `virtualenv`. If not, please, contact the administrators or install it by yourself if you do it on your personal computer. We will use it in the next exercises and projects.

This exercise is pretty simple just to warm you up and get you acquainted with the concept of virtual environments. What you need to do is:

1. to create a virtual environment with the name equal to your nickname using Python 3
2. to activate it,
3. to run Python 3 from the terminal,
4. to print the virtual env name using os library,
5. to write a small python script that does that thing from calling it in command-line:

   ```
   $ ./venv.py
   Your current virtual env is /Users/McShtuder/shtuder
   ```

6. to deactivate the environment,
7. to run again the script.

If you got a KeyError or an exception when deactivated the env, think about why it happened. You do not have to fix it in the exercise.
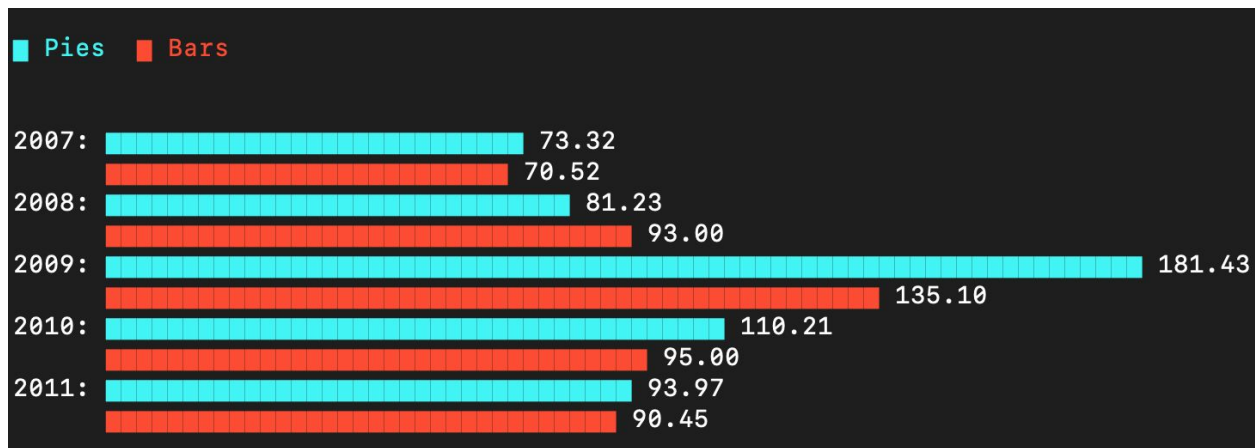
Chapter V

Exercise 01

Exercise 01 : Installing a package
Directory to store your solution : `ex01/`
Files to be in the directory : `pies_bars.sh and folder with your virtual env`
Authorized functions : `no restrictions`

Let us install the first package in your virtual environment!

We will work a little bit with the library `termgraph`. It gives you the power to draw graphs and diagrams right in your terminal. What can be cooler?

Install the library in the virtual environment created in the previous exercise.

Make exact the same visualization as below but with a different color scheme:

Make a shell script file for this purpose with the name `pies_bars.sh`. It contains only the part with making the graph without activation and deactivation of the env.

Chapter VI

Exercise 02

Exercise 02 : Installing many libraries
Directory to store your solution : `ex02/`
Files to be in the directory : `librarian.py and archive with your virtual env`
Authorized functions : `no restrictions`

During the next exercises, you will work with several different libraries. In this exercise, you need to prepare your virtual environment for them.

Install the latest release of BeautifulSoup and PyTest. It is prohibited to install them one by one (`pip install x, pip install y`). It is prohibited to use loops. Find a clever way to do it, use installation via requirements.

Write a python script `librarian.py` that:

1. checks that it runs inside the correct env,

2. installs the libraries,

3. displays all the installed libraries at the end like this (doesn't have to be exactly the same list):

   ```
   six==1.14.0
   soupsieve==2.0
   termgraph==0.2.0
   wcwidth==0.1.9
   zipp==3.1.0
   ```

4. saves it to `requirements.txt`.

Put an archive of your env in the folder. You can put archivation in your code or you can do it from the command line. The archive may be compressed if you think it is useful. If the script was called from the wrong env, there should be an exception.

Chapter VII

Exercise 03

Exercise 03 : Very beautiful soup
Directory to store your solution : `ex03/`
Files to be in the directory : `financial.py`
Authorized functions : `no restrictions`

Ok, so you have installed 2 libraries in the previous exercise. Let us work with one of them – BeautifulSoup. It is very useful when you need to parse a website that does not have an API (like it was with HeadHunter in day00). The problem is that when you parse a webpage, you get not only useful information but HTML markup as well that gives you pain. The package helps you navigate in different blocks and classes in HTML making easier to extract from them what you really need. But keep in mind that it is not a parser itself, it just helps you navigate in the mess of HTML or XML (it means that you need to install an HTTP-library to your taste in your env).

In this exercise, you will parse Yahoo Finance (yeah, it has an API, but for the learning purposes let us forget about it). You will need to visit a [page like this](#) and to get some data for a specific field of a specific company.

Write a Python script that:

- **gets**: as the arguments the ticker symbol and the field of the table (for example, `MSFT`, `Total Revenue`),

- **returns**: the tuple that contains the requested information,

- **special conditions**: put a sleep for 5 seconds inside your script (we will need it later).

The example:

```
$ ./financial.py 'MSFT' 'Total Revenue'
('Total Revenue', '134,249,000', '125,843,000', '110,360,000', '89,950,000',
'85,320,000')
```

If the URL does not exist, raise an exception. If the requested field does not exist, raise an exception.

Chapter VIII

Exercise 04

Exercise 04 : Profiling
Directory to store your solution : `ex04/`
Files to be in the directory : `financial.py`, `financial_enhanced.py`, `profiling-sleep.txt`, `profiling-tottime.txt`, `profiling-http.txt`, `profiling-ncalls.txt`, `profiling-percall.txt`
Authorized functions : `no restrictions`

No chance that you in the future will write code 100% perfectly without any chance of improvement. Likely, you will have to find out why your scripts work not as fast as you want. And there is a thing for such purposes – profilers. According to Wikipedia, profiling is a form of dynamic program analysis that measures, for example, the space or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.

Remember your script from the previous exercise? Let us optimize it. Even if you are a guru of programming, there was one structure that was not very effective (we asked you to do it that way).

1. Applying cProfile to your script `financial.py`, get a table of used functions sorted descending by total time spent on their execution. Save it to the file `profiling-sleep.txt`.

```
('Total Revenue', '134,249,000', '125,843,000', '110,360,000', '89,950,000', '85,320,000')
        244845 function calls (236544 primitive calls) in 7.452 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    5.005    5.005    5.005    5.005 {built-in method time.sleep}
      511    1.946    0.004    1.946    0.004 {method 'read' of '_ssl._SSLSocket' objects}
        1    0.155    0.155    0.155    0.155 {method 'do_handshake' of '_ssl._SSLSocket' objects}
        1    0.056    0.056    0.056    0.056 {method 'connect' of '_socket.socket' objects}
```

2. Delete from your script the line with `time.sleep(5)` and run profiling again. You should get a new table without `{built-in method time.sleep}`. Save it to the file `profiling-tottime.txt`.

3. Try using another HTTP-client library to see if your script got faster. Save the new script to `financial_enhanced.py`. Save the result of profiling to the file `profiling-http.txt`.

4. Get the same table but descendingly sorted by number of calls. Sometimes it is useful to know: maybe you can optimize those functions to make them call fewer times. Save the table to the file `profiling-ncalls.txt`.

5. Get the same table but descendingly sorted by the time per call. Sometimes it is useful to know: maybe you can optimize those functions to make them call faster. Save the table to the file `profiling-percall.txt`.

Chapter IX

Exercise 05

Exercise 05 : PyTest
Directory to store your solution : `ex05/`
Files to be in the directory : `financial_test.py`
Authorized functions : `no restrictions`

Well, the speed of your script is not the only issue to consider. Your script may work as not intended by you from the first place. To be sure that the script works properly you need to conduct unit tests: for example, to give as the input different things and to make sure that it returns what expected.

We are sure that in `ex03` you used one or more functions. For each of the functions, you need to create at least 3 tests using the library PyTest. For example, check if your script gives the correct information for the request: if I ask for Total Revenue, do I get the total revenue for the given ticker. Is the type of the return a tuple? If I give an invalid ticker name, do I get an exception?

Modify your script `financial.py` by adding in the code the tests. Put the file in your directory with the name `financial_test.py`. Run PyTest. Your tests should be passed. If not, work on your script to make it ready.