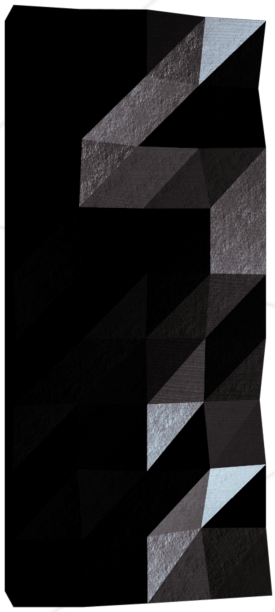


MODULE 03 – Piscine Python for Data Science

Intro to Python: Package management and virtual environment

Summary: Today we will help you acquire basic knowledge about how to manage libraries in Python and work with virtual environments.

Contents

I	Foreword	2
II	Instructions	3
III	Specific instructions for the day	4
IV	Exercise 00 : Virtual Environment	5
V	Exercise 01 : Installing a package	7
VI	Exercise 02 : Installing many libraries	8
VII	Exercise 03 : Very beautiful soup	9
VIII	Exercise 04 : Profiling	10
IX	Exercise 05 : PyTest	12

Chapter I

Foreword

10 library rules:

- Use a level 0-1 voice.
- Use a shelf marker.
- Turn the pages from the top corner.
- Wash your hands before touching a book.
- Return books on time.
- Never eat or drink while reading.
- Keep books dry.
- Use a bookmark.
- Do not write or draw in books.
- Keep books away from babies and pets.

Chapter II

Instructions

- Use this page as the your only reference. Do not listen to any rumors or speculations about how to prepare your solution.
- Here and further on we use Python 3 as the only correct version of Python.
- The solutions for python exercises (module01, module02, module03) must have the following 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 any additional files in your directory other than those explicitly specified in the subject. 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. If that fails, try your neighbor on the left.
- Your reference material: peers / Internet / Google.
- You can ask questions in Slack.
- 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 for the day

- No code in the global scope. Use functions!
- Each file must be ended by a function call in a condition similar to:

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

- Any exception not caught will invalidate your work, even in the event of an error that you were asked you to test.
- No imports are allowed, except those explicitly mentioned in the section “Authorized functions” of the title block of each exercise.

Chapter IV

Exercise 00 : Virtual Environment

	Exercise 00
Virtual Environment	
Turn-in directory : $ex00/$	
Files to turn in : `venv.py and the folder with your virtual env`	
Allowed functions : `import os`	

Libraries, or in other words packages, are one of the means by which coding has been democratized. It has never been easier to learn to code and get quick results from this process. Some programmers have written 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 such already existing classes, methods, or functions from scratch you can reuse them. All you need to do is sudo pip install. Or wait...

This way of installing Python packages is considered bad practice. When you do it as 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 programs that are essential programs for the system. By installing external packages like that you may ruin your system. So you almost never need to sudo pip install.

There is a better way – virtual environments. Think of 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 package called virtualenv preinstalled. If not, please, contact the administrators or install it by yourself if you are working on your personal computer. We will use it in the following exercises and projects.

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

- create a virtual environment with your nickname as its name using Python 3 (you will work with this env here and further on),
- activate it,
- run Python 3 from the terminal,
- print the virtual env name using os library,

- write a small python script that does that thing by calling it in command-line:

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

- deactivate the environment,
- run the script again ...

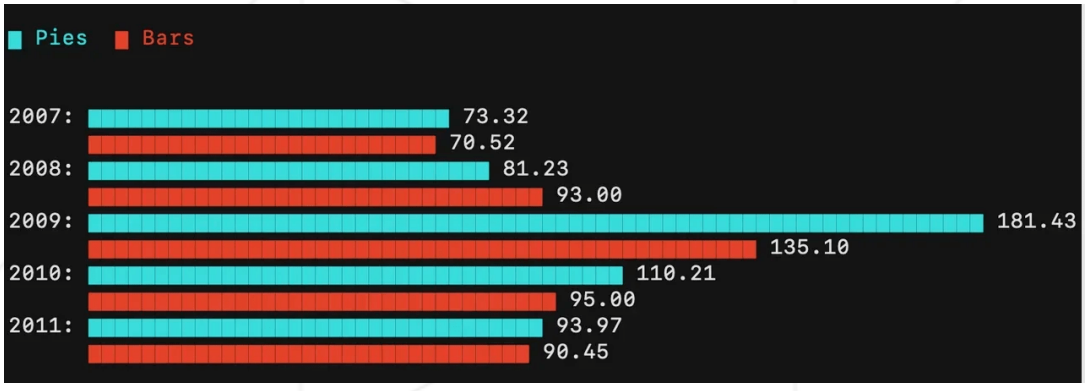
If you got a KeyError or an exception when deactivated the env, consider why it happened. You do not have to fix it in this exercise, but be ready to explain why it happened.

Chapter V

Exercise 01 : Installing a package

	Exercise 01
Installing a package	
Turn-in directory : $ex01/$	
Files to turn in : `pies_bars.sh, the file with the data and the folder with your virtual env`	
Allowed functions : `no restrictions`	

- Let us install the first package in your virtual environment!
- We will work with the library termgraph a bit. It gives you the power to draw graphs and diagrams right in your terminal. What could be cooler?
- Install the library in the virtual environment created in the previous exercise.
- Make exactly the same visualization as below but with a different color scheme (create a file for the visualization by yourself):



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

Chapter VI

Exercise 02 : Installing many libraries

	Exercise 02
Installing many libraries	
Turn-in directory : $ex02/$	
Files to turn in : `librarian.py and the archive with your virtual env`	
Allowed functions : `no restrictions`	

During the following 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 called librarian.py that:

- checks that it runs inside the correct env
- installs the libraries
- 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
```

- 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 that would be useful. If the script was called from the wrong env, there should be an exception.

Chapter VII

Exercise 03 : Very beautiful soup

	Exercise 03
Very beautiful soup	
Turn-in directory : $ex03/$	
Files to turn in : `financial.py`	
Allowed 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 (As was the case with HeadHunter on day00). The problem is that when you parse a webpage, you get not only useful information but also HTML markup that will be a pain for you. This package helps you navigate in different blocks and classes in HTML, making easier to extract what you really need from them. But keep in mind that it is not a parser itself, it just helps you navigate in the mess of HTML or XML (meaning that you need to install an HTTP-library as per your own taste in your env).

In this exercise, you will parse Yahoo Finance (yeah, it has an API, but for learning purposes let us forget about that). You will need to visit a page like this and 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: add 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 : Profiling

	Exercise 04
Profiling	
Turn-in directory : $ex04/$	
Files to turn in : `financial.py, financial_enhanced.py, profiling-sleep.txt, profiling-tottime.txt, profiling-http.txt, profiling-ncalls.txt`	
Allowed functions : `no restrictions`	

There is no chance that you will write code 100% perfectly in the future without any scope for improvement. You will likely have to figure out why your scripts don't work as fast as you want. And we have the thing for such purposes - profilers. According to Wikipedia, profiling is a form of dynamic program analysis that measures, for example, the spatial or temporal 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 programming guru, there was one structure that was not very effective (we asked you to do it that way).

- Applying cProfile to your script financial.py, get a table of the functions used sorted in descending order 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}
```

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

- Try using another HTTP-client library to see if your script got any faster. Save the new script to financial_enhanced.py. Save the result of the profiling to the file profiling-http.txt
- Get the same table but sorted in descendingly order by number of calls. Sometimes it is useful to know: that you can choose to optimize those functions to make them call fewer times. Save the table to the file profiling-ncalls.txt
- This time use the library pstats. Sort by cumulative time and get the top 5. Save it to the file pstats-cumulative.txt

Chapter IX

Exercise 05 : PyTest

	Exercise 05
PyTest	
Turn-in directory : $ex05/$	
Files to turn in : `financial_test.py`	
Allowed functions : `no restrictions`	

Well, the speed of your script is not the only issue to consider. Your script may not work as you intended from the start. To be sure that the script works properly, you need to conduct unit tests: for example, to give different things as the input and 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. 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 the tests into the code. Put the file in your directory with the name financial_test.py. Run PyTest. Your tests should have passed. If not, work on your script to make it ready.