**How to create a Python library**

[[](https://medium.com/@kiaeisinga?source=post_page-----7d5aea80cc3f--------------------------------)](https://medium.com/@kiaeisinga?source=post_page-----7d5aea80cc3f--------------------------------)

[[Analytics Vidhya](https://medium.com/analytics-vidhya?source=post_page-----7d5aea80cc3f--------------------------------)](https://medium.com/analytics-vidhya?source=post_page-----7d5aea80cc3f--------------------------------)

[Kia Eisinga](https://medium.com/@kiaeisinga?source=post_page-----7d5aea80cc3f--------------------------------)

·

Follow

Published in

[Analytics Vidhya](https://medium.com/analytics-vidhya?source=post_page-----7d5aea80cc3f--------------------------------)

·

7 min read

·

Jan 27, 2020

2.1K

26



Photo by [Iñaki del Olmo](https://unsplash.com/@inakihxz?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral)

Ever wanted to create a Python library, albeit for your team at work or for some open source project online? In this blog you will learn how to!

The tutorial is easiest to follow when you are using the same tools, however it is also possible for you to use different ones.

The tools used in this tutorial are:  
- Linux command prompt  
- Visual Studio Code

**Step 1: Create a directory in which you want to put your library**Open your command prompt and create a folder in which you will create your Python library.

Remember:  
- With pwd you can see your present working directory.  
- With ls you can list the folders and files in your directory.  
- With cd <path> you can change the current present directory you are in.  
- With mkdir <folder> you can create a new folder in your working directory.

In my case, the folder I will be working with is mypythonlibrary. Change the present working directory to be your folder.

**Step 2: Create a virtual environment for your folder**When starting your project, it is always a good idea to create a virtual environment to encapsulate your project. A virtual environment consists of a certain Python version and some libraries.

Virtual environments prevent the issue of running into dependency issues later on. For example, in older projects you might have worked with older versions of the numpy library. Some old code, that once worked beautifully, might stop working once you update its version. Perhaps parts of numpy are no longer compatible with other parts of your program. Creating virtual environments prevents this. They are also useful in cases when you are collaborating with someone else, and you want to make sure that your application is working on their computer, and vice versa.

(Make sure you changed the present working directory to the folder you are going to create your Python library in (cd <path/to/folder>).)

Go ahead and create a virtual environment by typing:

% python3 -m venv venv

Once it is created, you must now activate the environment by using:

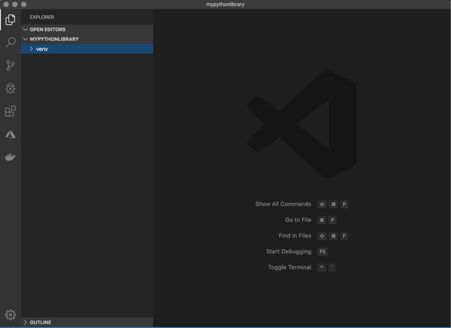
% source venv/bin/activate

Activating a virtual environment modifies the PATH and shell variables to point to the specific isolated Python set-up you created. PATH is an environmental variable in Linux and other Unix-like operating systems that tells the shell which directories to search for executable files (i.e., ready-to-run programs) in response to commands issued by a user. The command prompt will change to indicate which virtual environment you are currently in by prepending (yourenvname).

In your environment, make sure you have pip installed wheel, setuptools and twine. We will need them for later to build our Python library.

% pip install wheel  
% pip install setuptools  
% pip install twine

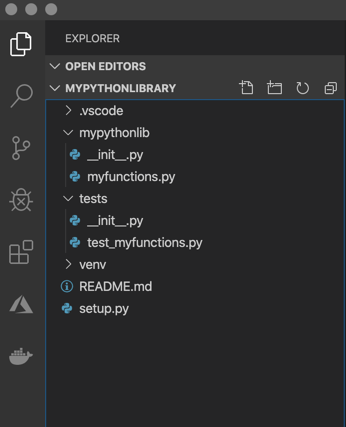
**Step 3: Create a folder structure**In Visual Studio Code, open your folder mypythonlibrary (or any name you have given your folder). It should look something like this:



You now can start adding folders and files to your project. You can do this either through the command prompt or in Visual Studio Code itself.

1. Create an empty file called setup.py. This is one of the most important files when creating a Python library!
2. Create an empty file called README.md. This is the place where you can write markdown to describe the contents of your library for other users.
3. Create a folder called mypythonlib, or whatever you want your Python library to be called when you pip install it. (The name should be unique on pip if you want to publish it later.)
4. Create an empty file inside mypythonlib that is called \_\_init\_\_.py. Basically, any folder that has an \_\_init\_\_.py file in it, will be included in the library when we build it. Most of the time, you can leave the \_\_init\_\_.py files empty. Upon import, the code within \_\_init\_\_.py gets executed, so it should contain only the minimal amount of code that is needed to be able to run your project. For now, we will leave them as is.
5. Also, in the same folder, create a file called myfunctions.py.
6. And, finally, create a folder tests in your root folder. Inside, create an empty \_\_init\_\_.py file and an empty test\_myfunctions.py.

Your set-up should now look something like this:



**Step 4: Create content for your library**To put functions inside your library, you can place them in the myfunctions.py file. For example, copy the [haversine function](https://stackoverflow.com/questions/4913349/haversine-formula-in-python-bearing-and-distance-between-two-gps-points) in your file:

from math import radians, cos, sin, asin, sqrt  
  
def haversine(lon1: float, lat1: float, lon2: float, lat2: float) -> float:  
 """  
 Calculate the great circle distance between two points on the   
 earth (specified in decimal degrees), returns the distance in  
 kilometers.  
 All arguments must be of equal length.  
 :param lon1: longitude of first place  
 :param lat1: latitude of first place  
 :param lon2: longitude of second place  
 :param lat2: latitude of second place  
 :return: distance in kilometers between the two sets of coordinates  
 """  
 # Convert decimal degrees to radians  
 lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])  
  
 # Haversine formula  
 dlon = lon2 - lon1  
 dlat = lat2 - lat1  
 a = sin(dlat/2)\*\*2 + cos(lat1) \* cos(lat2) \* sin(dlon/2)\*\*2  
 c = 2 \* asin(sqrt(a))  
 r = 6371 # Radius of earth in kilometers  
 return c \* r

This function will give us the distance in meters between two latitude and longitude points.

Whenever you write any code, it is highly encouraged to also write tests for this code. For testing with Python you can use the libraries pytest and pytest-runner. Install the library in your virtual environment:  
> pip install pytest==4.4.1  
> pip install pytest-runner==4.4

Let’s create a small test for the haversine function. Copy the following and place it inside the test\_myfunctions.py file:

from mypythonlib import myfunctions  
  
def test\_haversine():  
 # Amsterdam to Berlin  
 assert myfunctions.haversine(  
 4.895168, 52.370216, 13.404954, 52.520008  
 ) == 576.6625818456291

Finally, let’s create a setup.py file, that will help us to build the library. A limited version of setup.py will look something like this:

from setuptools import find\_packages, setup  
  
setup(  
 name='mypythonlib',  
 packages=find\_packages(),  
 version='0.1.0',  
 description='My first Python library',  
 author='Me',  
)

The name variable in setup holds whatever name you want your package wheel file to have. To make it easy, we will gave it the same name as the folder.

**Set the packages you would like to create**While in principle you could use find\_packages() without any arguments, this can potentially result in unwanted packages to be included. This can happen, for example, if you included an \_\_init\_\_.py in your tests/ directory (which we did). Alternatively, you can also use the exclude argument to explicitly prevent the inclusion of tests in the package, but this is slightly less robust. Let’s change it to the following:

from setuptools import find\_packages, setup  
  
setup(  
 name='mypythonlib',  
 packages=find\_packages(include=['mypythonlib']),  
 version='0.1.0',  
 description='My first Python library',  
 author='Me',  
)

**Set the requirements your library needs**Note that pip does not use requirements.yml / requirements.txt when your project is installed as a dependency by others. Generally, for that, you will have to specify dependencies in the install\_requires and tests\_require arguments in your setup.py file.

Install\_requires should be limited to the list of packages that are absolutely needed. This is because you do not want to make users install unnecessary packages. Also note that you do not need to list packages that are part of the standard Python library.

However, since we have only defined the haversine function so far and it only uses the math library (which is always available in Python), we can leave this argument empty.

Maybe you can remember us installing the pytest library before. Of course, you do not want to add pytest to your dependencies in install\_requires: it isn’t required by the users of your package. In order to have it installed automatically only *when you run tests*you can add the following to your setup.py:

from setuptools import find\_packages, setup  
  
setup(  
 name='mypythonlib',  
 packages=find\_packages(include=['mypythonlib']),  
 version='0.1.0',  
 description='My first Python library',  
 author='Me',  
 install\_requires=[],  
 setup\_requires=['pytest-runner'],  
 tests\_require=['pytest==4.4.1'],  
 test\_suite='tests',  
)

Running:

% python setup.py pytest

will execute all tests stored in the ‘tests’ folder.

**Step 5: Build your library**Now that all the content is there, we want to build our library. Make sure your present working directory is /path/to/mypythonlibrary (so the root folder of your project). In your command prompt, run:

% python setup.py bdist\_wheel

Your wheel file is stored in the “dist” folder that is now created. You can install your library by using:

% pip install /path/to/wheelfile.whl

Note that you could also publish your library to an internal file system on intranet at your workplace, or to the official PyPI repository and install it from there.

Once you have installed your Python library, you can import it using:

import mypythonlib  
from mypythonlib import myfunctions

[Python](https://medium.com/tag/python?source=post_page-----7d5aea80cc3f---------------python-----------------)

[Pip](https://medium.com/tag/pip?source=post_page-----7d5aea80cc3f---------------pip-----------------)

[Python Programming](https://medium.com/tag/python-programming?source=post_page-----7d5aea80cc3f---------------python_programming-----------------)