# Python - Packages

We organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in Python takes the concept of the modular approach to next logical level. As you know, a [module](https://www.tutorialsteacher.com/python/python-module) can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules. Physically, a package is actually a folder containing one or more module files.

Let us create a package named mypackage, using the following steps:

* Create a new folder named D:\MyApp.
* Inside MyApp, create a subfolder with the name 'mypackage'.
* Create an empty \_\_init\_\_.py file in the mypackage folder.
* Using a Python-aware editor like IDLE, create modules greet.py and functions.py with the following code:

**greet.py**

def SayHello(name):

print("Hello ", name)

**functions.py**

def sum(x,y):

return x+y

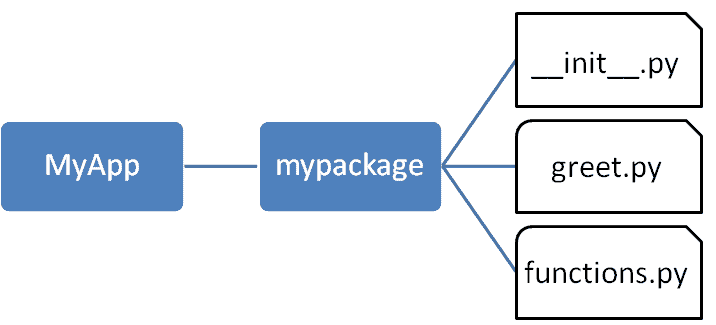
def average(x,y):

return (x+y)/2

def power(x,y):

return x\*\*y

That's it. We have created our package called mypackage. The following is a folder structure:

[](https://www.tutorialsteacher.com/Content/images/python/package.png)Package Folder Structure

## Importing a Module from a Package

Now, to test our package, navigate the command prompt to the MyApp folder and invoke the Python prompt from there.

D:\MyApp>python

Import the functions module from the mypackage package and call its power() function.

>>> from mypackage import functions  
>>> functions.power(3,2)  
9

It is also possible to import specific functions from a module in the package.

>>> from mypackage.functions import sum  
>>> sum(10,20)  
30  
>>> average(10,12)  
Traceback (most recent call last):  
File "<pyshell#13>", line 1, in <module>  
NameError: name 'average' is not defined

## \_\_init\_\_.py

The package folder contains a special file called \_\_init\_\_.py, which stores the package's content. It serves two purposes:

1. The Python interpreter recognizes a folder as the package if it contains \_\_init\_\_.py file.
2. \_\_init\_\_.py exposes specified resources from its modules to be imported.

An empty \_\_init\_\_.py file makes all functions from the above modules available when this package is imported. Note that \_\_init\_\_.py is essential for the folder to be recognized by Python as a package. You can optionally define functions from individual modules to be made available.

 Note:

We shall also create another Python script in the MyApp folder and import the mypackage package in it. It should be at the same level of the package to be imported.

The \_\_init\_\_.py file is normally kept empty. However, it can also be used to choose specific functions from modules in the package folder and make them available for import. Modify \_\_init\_\_.py as below:

**\_\_init\_\_.py**

from .functions import average, power

from .greet import SayHello

The specified functions can now be imported in the interpreter session or another executable script.

Create test.py in the MyApp folder to test mypackage.

test.py

from mypackage import power, average, SayHello

SayHello()

x=power(3,2)

print("power(3,2) : ", x)

Note that functions power() and SayHello() are imported from the package and not from their respective modules, as done earlier. The output of the above script is:

D:\MyApp>python test.py  
Hello world  
power(3,2) : 9

## Install a Package Globally

Once a package is created, it can be installed for system-wide use by running the setup script. The script calls setup() function from the setuptools module.

Let's install mypackage for system-wide use by running a setup script.

Save the following code as setup.py in the parent folder MyApp. The script calls the setup() function from the setuptools module. The setup() function takes various arguments such as name, version, author, list of dependencies, etc.

The zip\_safe argument defines whether the package is installed in compressed mode or regular mode.

Example: setup.py

from setuptools import setup

setup(

name='mypackage',

version='0.1',

description='Testing installation of Package',

url='#',

author='auth',

author\_email='your@email’,

license='HITACHI',

packages=['mypackage'],

zip\_safe=False

)

Now execute the following command to install mypackage using the [pip](https://www.tutorialsteacher.com/python/pip-in-python) utility. Ensure that the command prompt is in the parent folder, in this case D:\MyApp.

D:\MyApp>pip install mypackage  
Processing d:\MyApp  
Installing collected packages: mypack  
Running setup.py install for mypack ... done  
Successfully installed mypackage-0.1

Now mypackage is available for system-wide use and can be imported in any script or interpreter.

D:\>python  
>>> import mypackage  
>>>mypackage.average(10,20)  
15.0  
>>>mypackage.power(10,2)  
100

You may also want to publish the package for public use. [PyPI](https://pypi.org/) (stands for Python Package Index) is a repository of Python packages.

Visit <https://packaging.python.org/distributing> to know more about the procedure of uploading a package to PyPI.