

Exercise 46: A Project Skeleton

This will be where you start learning how to set up a good project "skeleton" directory. This skeleton directory will have all the basics you need to get a new project up and running. It will have your project layout, automated tests, modules, and install scripts. When you go to make a new project, just copy this directory to a new name and edit the files to get started.

Installing Python Packages

Before you can begin this exercise you need to install some software for Python by using a tool called "pip" to install new modules. Here's the problem though. You are at a point where it's difficult for me to help you do that and keep this book sane and clean. There are so many ways to install software on so many computers that I'd have to spend 10 pages walking you through every step, and let me tell you I am a lazy guy.

Rather than tell you how to do it exactly, I'm going to tell you what you should install, and then tell you to figure it out and get it working. This will be really good for you since it will open a whole world of software you can use that other people have released to the world.

Install the following Python packages:

1. pip from <http://pypi.python.org/pypi/pip>
2. distribute from <http://pypi.python.org/pypi/distribute>
3. nose from <http://pypi.python.org/pypi/nose/>
4. virtualenv from <http://pypi.python.org/pypi/virtualenv>

Do not just download these packages and install them by hand. Instead see how other people recommend you install these packages and use them for your particular system. The process will be different for most versions of Linux, OSX, and definitely different for Windows.

I am warning you; this will be frustrating. In the business we call this "yak shaving." Yak shaving is any activity that is mind numbingly, irritatingly boring and tedious that you have to do before you can do something else that's more fun. You want to create cool Python projects, but you can't do that until you set up a skeleton directory, but you can't set up a skeleton directory until you install some packages, but you can't install packages until you install package installers, and you can't install package installers until you figure out how your system installs software in general, and so on.

Struggle through this anyway. Consider it your trial by annoyance to get into the programmer club. Every programmer has to do these annoying tedious tasks before they can do something cool.

Note

Sometimes the Python installer does not add the `C:\Python27\Script` to the system `PATH`. If this is the case for you, go back and add this to the path just like you did for `C:\Python27` in Exercise 0 with:

```
[Environment]::SetEnvironmentVariable("Path",  
"$env:Path;C:\Python27\Scripts", "User")
```

Creating the Skeleton Project Directory

First, create the structure of your skeleton directory with these commands:

```
$ mkdir projects  
$ cd projects/
```

```
$ mkdir skeleton
$ cd skeleton
$ mkdir bin
$ mkdir NAME
$ mkdir tests
$ mkdir docs
```

I use a directory named `projects` to store all the various things I'm working on. Inside that directory I have my `skeleton` directory that I put the basis of my projects into. The directory `NAME` will be renamed to whatever you are calling your project's main module when you use the skeleton.

Next we need to set up some initial files. Here's how you do that on Linux/OSX:

```
$ touch NAME/__init__.py
$ touch tests/__init__.py
```

Here's the same thing on Windows PowerShell:

```
$ new-item -type file NAME/__init__.py
$ new-item -type file tests/__init__.py
```

That creates an empty Python module directories we can put our code in.

Then we need to create a `setup.py` file we can use to install our project later if we want:

```
1  try:
2      from setuptools import setup
3  except ImportError:
4      from distutils.core import setup
5
6  config = {
7      'description': 'My Project',
8      'author': 'My Name',
9      'url': 'URL to get it at.',
10     'download_url': 'Where to download it.',
11     'author_email': 'My email.',
12     'version': '0.1',
13     'install_requires': ['nose'],
14     'packages': ['NAME'],
15     'scripts': [],
16     'name': 'projectname'
17 }
18
19 setup(**config)
```

Edit this file so that it has your contact information and is ready to go for when you copy it.

Finally you will want a simple skeleton file for tests named

tests/NAME_tests.py:

```
1  from nose.tools import *
2  import NAME
3
4  def setup():
5      print "SETUP!"
6
7  def teardown():
8      print "TEAR DOWN!"
9
10 def test_basic():
11     print "I RAN!"
```

Final Directory Structure

When you are done setting all of this up, your directory should look like mine here:

```
$ ls -R
NAME                bin                docs
setup.py            tests
```

```
./NAME:
__init__.py
```

```
./bin:
```

```
./docs:
```

```
./tests:
NAME_tests.py    __init__.py
```

This is on Unix, but the structure is the same on Windows, and if I were to draw it out as a tree:

```
setup.py
NAME/
  __init__.py
bin/
docs/
```

```
tests/
  NAME_tests.py
  __init__.py
```

And from now on, you should run your commands from that work with this directory from this point. If you can't do `ls -R` and see this same structure, then you are in the wrong place. For example, people commonly go into the `tests/` directory to try to run files there, which won't work. To run your application's tests, you would need to be *above* `tests/` and this location I have above. So, if you try this:

```
$ cd tests/    # WRONG! WRONG! WRONG!
$ nosetests
```

```
-----
-----
Ran 0 tests in 0.000s
```

OK

Then that is *wrong*! You have to be above tests, so assuming you made this mistake, you would fix it by doing this:

```
$ cd ..    # get out of tests/
$ ls      # CORRECT! you are now in the right spot
NAME      bin      docs
setup.py  tests
$ nosetests
```

```

-----
-----
Ran 1 test in 0.004s
```

OK

Remember this because people make this mistake quite frequently.

Testing Your Setup

After you get all that installed you should be able to do this:

```
$ nosetests
.
-----
-----
Ran 1 test in 0.007s
```

OK

I'll explain what this `nosetests` thing is doing in the next exercise, but for now if you do not see that, you probably got something wrong. Make sure you put `__init__.py` files in your `NAME` and `tests` directories and make sure you got `tests/NAME_tests.py` right.

Using the Skeleton

You are now done with most of your yak shaving. Whenever you want to start a new project, just do this:

1. Make a copy of your skeleton directory. Name it after your new project.
2. Rename (move) the `NAME` module to be the name of your project or whatever you want to call your root module.
3. Edit your `setup.py` to have all the information for your project.
4. Rename `tests/NAME_tests.py` to also have your module name.
5. Double check it's all working by using `nosetests` again.
6. Start coding.

Required Quiz

This exercise doesn't have Study Drills but a quiz you should complete:

1. Read about how to use all of the things you installed.
2. Read about the `setup.py` file and all it has to offer. Warning: it is not a very well-written piece of software, so it will be very strange to use.
3. Make a project and start putting code into the module, then get the module working.
4. Put a script in the `bin` directory that you can run. Read about how you can make a Python script that's runnable for your system.
5. Mention the `bin` script you created in your `setup.py` so that it gets installed.

6. Use your `setup.py` to install your own module and make sure it works, then use `pip` to uninstall it.

Common Student Questions

Do these instructions work on Windows?

They should, but depending on the version of Windows you may need to struggle with the setup a bit to get it working. Just keep researching and trying it until you get it, or see if you can ask a more experienced Python+Windows friend to help out.

It seems I can't run `nosetests` on Windows?

Sometimes the Python installer does not add the `C:\Python27\Script` to the system `PATH`. If this is the case for you, go back and add this to the path just like you did for `C:\Python27` in Exercise 0.

What do I put in the config dictionary in my `setup.py`?

Make sure you read the documentation for `distutils` at <http://docs.python.org/distutils/setupscript.html>.

I can't seem to load the `NAME` module and just get an `ImportError`.

Make sure you made the `NAME/__init__.py` file. If you're on Windows make sure you didn't accidentally name it `NAME/__init__.py.txt` which happens by default with some editors.

Why do we need a `bin/` folder at all?

This is just a standard place to put scripts that are run on the command line, not a place to put modules.

Do you have a real world example project?

There are many projects written in Python that do this, but try this simple one I created: <https://gitorious.org/python-modargs>.

My `nosetests` run only shows one test being run. Is that right?

Yes, that's what my output shows too.

Purchase The Videos For \$29.59

For just \$29.59 you can get access to all the videos for [Learn Python The Hard Way](#), **plus** a PDF of the book and no more popups all in this one location. For \$29.59 you get:

- All 52 videos, 1 per exercise, almost 2G of video.
- A PDF of the book.
- Email help from the author.
- [See a list of everything you get before you buy.](#)

When you buy the videos they will immediately show up **right here** without any hassles.



[Already Paid? Reactivate Your Purchase Right Now!](#)


Buying Is Easy

Buying is easy. Just fill out the form below and we'll get started.

Full Name

Email Address

☒    Pay With Credit Card (by Stripe™)

☐  Use your PayPal™ account.

Buy Learn Python The Hard Way, 3rd Edition



Copyright (C) 2010 Zed. A. Shaw