|  |  |
| --- | --- |
|  | **2014** |
|  | The Company  Martien Huijsmans |

|  |
| --- |
| **[Python]** |
| Summary |

Inhoud

[Tooling 1](#_Toc452897538)

[Hitchhiker’s guide to python 1](#_Toc452897539)

[Package Structure 2](#_Toc452897540)

[\_\_init\_\_.py includes dependency to other package 2](#_Toc452897541)

[\_\_main\_\_.py 3](#_Toc452897542)

[Building a package 4](#_Toc452897543)

[Is a RPM with all dependencies included feasible? 4](#_Toc452897544)

[User Guide 4](#_Toc452897545)

[Setup.py 4](#_Toc452897546)

[bdist 4](#_Toc452897547)

[Supported formats 4](#_Toc452897548)

[Create a RPM 5](#_Toc452897549)

[Creating a wheel 5](#_Toc452897550)

[Setuptools 6](#_Toc452897551)

[Help 6](#_Toc452897552)

[Build 7](#_Toc452897553)

[Tests 7](#_Toc452897554)

[Install 7](#_Toc452897555)

[Installed packages status 8](#_Toc452897556)

# Tooling

Eclipse Plugin: <http://pydev.org/>

# Hitchhiker’s guide to python

<http://docs.python-guide.org/en/latest/writing/style/>

* include howto check code style

# Package Structure

Source: <https://docs.python.org/3/tutorial/modules.html#packages> (  
Source: <http://effbot.org/pyfaq/what-is-init-py-used-for.htm>

Example directory structure:

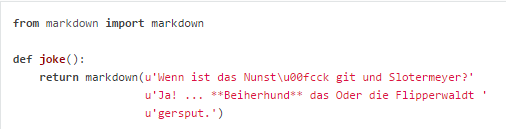
|  |  |  |
| --- | --- | --- |
| mypackage/ |  |  |
|  | \_\_init\_\_.py |  |
|  | file1.py |  |
|  | file2.py |  |
|  | subpackage/ |  |
|  |  | \_\_init\_\_.py |
|  |  | file3.py |
|  |  | file4.py |

Inclusion of a \_\_init\_\_.py file in a directory indicates to the Python interpreter that the directory should be treated like a Python package.

5 simple rules for building python packages:  
<https://axialcorps.com/2013/08/29/5-simple-rules-for-building-great-python-packages/>   
Refers to github project: <https://github.com/axialmarket/fsq>  
What I learned from the article and example is: I don’t understand WHAT they are doing with \_\_init\_\_py. Why all the information in \_\_init\_\_.py. Who uses that information, how and when?

## \_\_init\_\_.py includes dependency to other package

Source: <http://python-packaging.readthedocs.io/en/latest/dependencies.html>   
  
Assume your package **P** depends on other package caller Markdown (available via PyPi), than for **P** the \_\_init\_\_.py could look like:

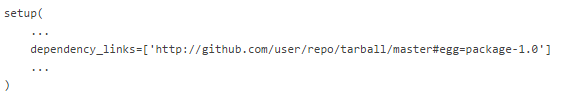


and the setup.py will include a **install\_requires**



When **P** is installed, maker is downloaded and installed.

When Markdown is not on PyPi, setup can include a **dependency\_links** where packages can be downloaded.



## \_\_main\_\_.py

Source: <http://stackoverflow.com/questions/4042905/what-is-main-py>

Often, a Python program is run by naming a .py file on the command line:

$ python my\_program.py

You can also create a directory or zipfile full of code, and include a \_\_main\_\_.py. Then you can simply name the directory or zipfile on the command line, and it executes the \_\_main\_\_.py automatically:

$ python my\_program\_dir

$ python my\_program.zip

# Or, if the program is accessible as a module

$ python -m my\_program

# Building a package

## Is a RPM with all dependencies included feasible?

Source: <http://www.alexhudson.com/2013/05/24/packaging-a-virtualenv-really-not-relocatable/>  
suggests difficult, e.g. patching required.   
Perhaps Docket is the solution.

Source: <https://nylas.com/blog/packaging-deploying-python/>   
Explores a good strategy for server sides build. Looked into Docker, Pex and final went for dh-virtualenv. Docker was dropped for organizational problems.  
Describes that Pip is slow and there is no rollback.  
  
Source: <http://pyfunc.blogspot.nl/2014/09/building-and-packaging-python.html>   
Describes howto create a build using: Pex and Pants and virtualenv.

## User Guide

<https://packaging.python.org/en/latest/>

## Setup.py

Source: <https://docs.python.org/2/distutils/setupscript.html>

*The setup script is the centre of all activity in building, distributing, and installing modules using the Distutils. The main purpose of the setup script is to describe your module distribution to the Distutils, so that the various commands that operate on your modules do the right thing.*

## bdist

### Supported formats

Check supported formats bdist

python setup.py bdist --help-formats

Example output:  
./setup.py bdist --help-formats  
List of available distribution formats:  
 --formats=rpm RPM distribution  
 --formats=gztar gzip'ed tar file  
 --formats=bztar bzip2'ed tar file  
 --formats=ztar compressed tar file  
 --formats=tar tar file  
 --formats=wininst Windows executable installer  
 --formats=zip ZIP file  
 --formats=msi Microsoft Installer

### Create a RPM

Following files exists at command execution:

* hello\_world.py
* setup.py

python setup.py bdist\_rpm

Creates

* dir: build  
  holds the artifacts from which a build is created.
* dir: dist  
  holds the distributables:
  + hello\_world-1.0-1.noarch.rpm
  + hello\_world-1.0-1.src.rpm
  + hello\_world-1.0.tar.gz
* file: MANIFEST  
  Role: ??

noarch rpm contains   
/usr/lib/python2.7/site-packages/  
which contains

* hello\_world.py
* hello\_world.pyc
* hello\_world.pyo
* hello\_world-1.0-py2.7.egg-info

## Creating a wheel

Source: <http://pythonwheels.com/>   
[*Wheels*](https://pypi.python.org/pypi/wheel)*are*[*the new standard*](http://www.python.org/dev/peps/pep-0427)*of python distribution and are intended to replace eggs. Support is offered in pip >= 1.4 and setuptools >= 0.8*.  
  
source: <https://pypi.python.org/pypi/wheel>  
*Wheel A built-package format for Python.  
A wheel is a ZIP-format archive with a specially formatted filename and the .whl extension. It is designed to contain all the files for a PEP 376 compatible install in a way that is very close to the on-disk format. Many packages will be properly installed with only the “Unpack” step (simply extracting the file onto sys.path), and the unpacked archive preserves enough information to “Spread” (copy data and scripts to their final locations) at any later time.*

Wheel requires separate installation:

* sudo pip --proxy http://130.144.240.237:8080 install wheel   
  *Failed. Proxy ?*

## Setuptools

### Help

Help: ./setup.py --help-commands

Example output:  
Standard commands:  
 build build everything needed to install  
 build\_py "build" pure Python modules (copy to build directory)  
 build\_ext build C/C++ extensions (compile/link to build directory)  
 build\_clib build C/C++ libraries used by Python extensions  
 build\_scripts "build" scripts (copy and fixup #! line)  
 clean clean up temporary files from 'build' command  
 install install everything from build directory  
 install\_lib install all Python modules (extensions and pure Python)  
 install\_headers install C/C++ header files  
 install\_scripts install scripts (Python or otherwise)  
 install\_data install data files  
 sdist create a source distribution (tarball, zip file, etc.)  
 register register the distribution with the Python package index  
 bdist create a built (binary) distribution  
 bdist\_dumb create a "dumb" built distribution  
 bdist\_rpm create an RPM distribution  
 bdist\_wininst create an executable installer for MS Windows  
 upload upload binary package to PyPI  
 check perform some checks on the package

Extra commands:  
 rotate delete older distributions, keeping N newest files  
 develop install package in 'development mode'  
 setopt set an option in setup.cfg or another config file  
 saveopts save supplied options to setup.cfg or other config file  
 egg\_info create a distribution's .egg-info directory  
 upload\_docs Upload documentation to PyPI  
 nosetests Run unit tests using nosetests  
 alias define a shortcut to invoke one or more commands  
 easy\_install Find/get/install Python packages  
 bdist\_egg create an "egg" distribution  
 install\_egg\_info Install an .egg-info directory for the package  
 test run unit tests after in-place build  
  
usage: setup.py [global\_opts] cmd1 [cmd1\_opts] [cmd2 [cmd2\_opts] ...]  
 or: setup.py --help [cmd1 cmd2 ...]  
 or: setup.py --help-commands  
 or: setup.py cmd –help

### Build

Source: # Copied from: <https://pythonhosted.org/an_example_pypi_project/setuptools.html>  
With setuptools setup is imported from setuptools and not from distutils as shown in the following two examples.

|  |  |
| --- | --- |
| tool | import line |
| bdist | from distutils.core import setup |
| setuptools | from setuptools import setup, find\_packages |

At moment of writing it is unclear if setup parameters are the same.

Creating a rpm (same compared to using bdist)

./setup.py bdist\_rpm

# Tests

Execute the test cases for this package

./setup.py test

# Install

source: <http://stackoverflow.com/questions/3472430/how-can-i-make-setuptools-install-a-package-thats-not-on-pypi>

Link shows many options, e.g.

pip install svn+svn://svn.myproject.org/svn/MyProject

# Installed packages status

Following command gives list of current installed packages in global namespace:  
$ pip freeze

# Python inside a Docket Container

Source: <https://www.digitalocean.com/community/tutorials/docker-explained-how-to-containerize-python-web-applications>