Python app. package structure example

18th March 2018 By An7ar35

Source code is available from GitHub.

1 Introduction

2 Project directory structure

```
root ..... Root directory for project
├── setup.py ..... Setup configuration for setuptools
├── README.md ..... Project README file
├── requirements.txt ..... List of python packages to install for running the app.
└── app ..... Main package for the application
    ├── __init__.py
    ├── __main__.py ..... Application's entry point
    ├── app_module.py
    ├── packageA ..... A sub-package (optional)
    │   ├── __init__.py
    │   ├── module1.py
    │   └── module2.py
    └── packageB ..... Another sub-package (optional)
        ├── __init__.py
        └── module1.py
```

3 Setup configuration file

The `setup.py` file is used by `setuptools` for installing the application package into the system's python environment site-packages.

The significant section is contained in lines 20-22. This points to what gets executed first when the application is launched. Without this the program won't run.

Listing 1: setup.py

```
1 from setuptools import setup, find_packages
2
3 with open('README.md') as f:
4   readme = f.read()
```

```
5
6  with open('requirements.txt') as f:
7    requirements = f.read()
8
9  setup(
10   name='python_example_app',
11   version='0.1',
12   author='Author',
13   author_email='author@mail.com',
14   description='Python skeleton app example',
15   long_description=readme,
16   url='https://github.com/an7ar35/python-app-skeleton-structure',
17   license="MIT",
18   install_requires=requirements,
19   packages=find_packages(exclude=('tests', 'docs')),
20   entry_points=dict(
21     console_scripts=['python_example_app=app.__main__:main']
22   ),
23   classifiers=["Environment :: Console",
24          "Operating System :: POSIX :: Linux"],
25 )
```

4 Implementation

4.1 Program entry point

Listing 2: __main__.py of root package

```
1  import sys
2
3  from app.app_module import printThis
4  from app.packageB import module1
5  from app.packageA.module1 import ModulePrinter as ModPrint1
6  from app.packageA.module2 import ModulePrinter as ModPrint2
7
8
9  def main(argv=sys.argv[1:]):
10   print("[__main__:main] args = ", argv)
11   printThis("Hello, world!")        #from app_module.py
12   a = ModPrint1()                       #from packageA.module1
13   b = ModPrint2()                       #from packageA.module2
14   module1.printThis("Goodbye, world!")  #from packageB
15
16
17 if __name__ == '__main__':
18   main(sys.argv[1:])
```

4.2 Nested packages and modules

Listing 3: packageA/module1.py

```
class ModulePrinter():
  def __init__(self):
    print("[packageA.module1.ModulePrinter:__init__()]")
```

Listing 4: packageA/module2.py

```
class ModulePrinter():
  def __init__(self):
    print("[packageA.module2.ModulePrinter:__init__()]")
```

Listing 5: packageB/module1.py

```
def printThis(str):
  print("[packageB.module1:printThis(..)] ", str)
```

Listing 6: app_module.py

```
def printThis(str):
  print("[app_module:printThis(..)] ", str)
```

5 Installing and Running

Inside the root directory of the project:

```
$ sudo pip install -e .
```

Then you can run the application anywhere:

```
$ python_example_app 1 2 3 "abc"
```

The output of that command would be:

```
[__main__:main] args = ['1', '2', '3', 'abc']
[app_module:printThis(..)] Hello, world!
[packageA.module1.ModulePrinter:__init__()]
[packageA.module2.ModulePrinter:__init__()]
[packageB.module1:printThis(..)] Goodbye, world!
```

To remove the application:

```
$ sudo pip uninstall python_example_app
```