Modules

LaurenceWarne

August 10, 2019

Contents

1 Nomenclature

1. **Module** An object that serves as a unit of python code.
2. **Package** Analogous to directories in a file system, they are special kinds of modules which contain submodules or subpackages. Any module with a `__path__` attribute is a package.
3. **Namespace** A place where a variable is stored. Namespaces are implemented as dictionaries. There are the local, global and built-in namespaces as well as nested namespaces in objects.

2 The Import Statement

We can use the `import` statement to use code from other modules in our module.

```
import numpy as np
import matplotlib.pyplot  # Note this also imports maplotlib

x = np.linspace(0, 10, 1000)
```

```
matplotlib.pyplot.plot(x, x**2)
matplotlib.pyplot.show()
```

When a module is first imported, Python searches for the module and if found, it creates a **module object**. We can now access the module's global namespace from within our module using `mymodule.var`.

Importing a module does automatically import all of its submodules, but a package may choose to do this in its `__init__.py`. For more info see this post.

2.1 From

We can use the `from` statement to import subpackages, classes and functions into our script.

```
from numpy.random import standard_normal  # Import function standard_normal
from matplotlib import pyplot as plt      # Import subpackage plt

plt.scatter(standard_normal(1000), standard_normal(1000))
plt.show()
```

3 The `__init__.py` File

Any directory with an `__init__.py` file is recognised as a **valid package**.

When a regular package is imported, this `__init__.py` file is implicitly executed, and the objects it defines are bound to names in the package's namespace. The `__init__.py` file can contain the same Python code that any other module can contain, and Python will add some additional attributes to the module when it is imported.

```
my_package/
    __init__.py
    my_module.py
    my_other_module.py
```

In `__init__.py`:

```
from .my_module import Foo
from .my_module import foo
```

In `my_script.py`:

```
import my_package

# We can now use Foo and foo
x = my_package.Foo()
my_package.foo()
```

It's common however for an `__init__.py` file to be left completely empty.