13-Modules-and-Packages

September 28, 2016

1 Modules and Packages

One feature of Python that makes it useful for a wide range of tasks is the fact that it comes "batteries included" – that is, the Python standard library contains useful tools for a wide range of tasks. On top of this, there is a broad ecosystem of third-party tools and packages that offer more specialized functionality. Here we'll take a look at importing standard library modules, tools for installing third-party modules, and a description of how you can make your own modules.

1.1 Loading Modules: the `import` Statement

For loading built-in and third-party modules, Python provides the `import` statement. There are a few ways to use the statement, which we will mention briefly here, from most recommended to least recommended.

1.1.1 Explicit module import

Explicit import of a module preserves the module's content in a namespace. The namespace is then used to refer to its contents with a "`.`" between them. For example, here we'll import the built-in `math` module and compute the sine of pi:

```
In [1]: import math
        math.cos(math.pi)

Out[1]: -1.0
```

1.1.2 Explicit module import by alias

For longer module names, it's not convenient to use the full module name each time you access some element. For this reason, we'll commonly use the "`import ... as ...`" pattern to create a shorter alias for the namespace. For example, the NumPy (Numerical Python) package, a popular third-party package useful for data science, is by convention imported under the alias `np`:

```
In [2]: import numpy as np
        np.cos(np.pi)

Out[2]: -1.0
```

1.1.3 Explicit import of module contents

Sometimes rather than importing the module namespace, you would just like to import a few particular items from the module. This can be done with the "`from ... import ...`" pattern. For example, we can import just the `cos` function and the `pi` constant from the `math` module:

```
In [3]: from math import cos, pi
        cos(pi)
```

```
Out[3]: -1.0
```

1.1.4 Implicit import of module contents

Finally, it is sometimes useful to import the entirety of the module contents into the local namespace. This can be done with the "`from ... import *`" pattern:

```
In [4]: from math import *
        sin(pi) ** 2 + cos(pi) ** 2
```

```
Out[4]: 1.0
```

This pattern should be used sparingly, if at all. The problem is that such imports can sometimes overwrite function names that you do not intend to overwrite, and the implicitness of the statement makes it difficult to determine what has changed.

For example, Python has a built-in `sum` function that can be used for various operations:

```
In [5]: help(sum)
```

```
Help on built-in function sum in module builtins:

sum(...)
    sum(iterable[, start]) -> value

    Return the sum of an iterable of numbers (NOT strings) plus the value
    of parameter 'start' (which defaults to 0).  When the iterable is
    empty, return start.
```

We can use this to compute the sum of a sequence, starting with a certain value (here, we'll start with `-1`):

```
In [6]: sum(range(5), -1)
```

```
Out[6]: 9
```

Now observe what happens if we make the *exact same function call* after importing `*` from `numpy`:

```
In [7]: from numpy import *
```

```
In [8]: sum(range(5), -1)

Out[8]: 10
```

The result is off by one! The reason for this is that the `import *` statement *replaces* the built-in `sum` function with the `numpy.sum` function, which has a different call signature: in the former, we're summing `range(5)` starting at `-1`; in the latter, we're summing `range(5)` along the last axis (indicated by `-1`). This is the type of situation that may arise if care is not taken when using "`import *`" – for this reason, it is best to avoid this unless you know exactly what you are doing.

1.2 Importing from Python's Standard Library

Python's standard library contains many useful built-in modules, which you can read about fully in Python's documentation. Any of these can be imported with the `import` statement, and then explored using the help function seen in the previous section. Here is an extremely incomplete list of some of the modules you might wish to explore and learn about:

- `os` and `sys`: Tools for interfacing with the operating system, including navigating file directory structures and executing shell commands
- `math` and `cmath`: Mathematical functions and operations on real and complex numbers
- `itertools`: Tools for constructing and interacting with iterators and generators
- `functools`: Tools that assist with functional programming
- `random`: Tools for generating pseudorandom numbers
- `pickle`: Tools for object persistence: saving objects to and loading objects from disk
- `json` and `csv`: Tools for reading JSON-formatted and CSV-formatted files.
- `urllib`: Tools for doing HTTP and other web requests.

You can find information on these, and many more, in the Python standard library documentation: https://docs.python.org/3/library/.

1.3 Importing from Third-Party Modules

One of the things that makes Python useful, especially within the world of data science, is its ecosystem of third-party modules. These can be imported just as the built-in modules, but first the modules must be installed on your system. The standard registry for such modules is the Python Package Index (*PyPI* for short), found on the Web at http://pypi.python.org/. For convenience, Python comes with a program called `pip` (a recursive acronym meaning "pip installs packages"), which will automatically fetch packages released and listed on PyPI (if you use Python version 2, `pip` must be installed separately). For example, if you'd like to install the `supersmoother` package that I wrote, all that is required is to type the following at the command line:

```
$ pip install supersmoother
```

The source code for the package will be automatically downloaded from the PyPI repository, and the package installed in the standard Python path (assuming you have permission to do so on the computer you're using).

For more information about PyPI and the `pip` installer, refer to the documentation at http://pypi.python.org/.