Module Use-cases

So far you have learned about modules, packages and libraries in the context of using modules in Python. The third-party packages in Python are in most cases open-source, free and available for a wide variety of domains. These resources extend the functionality of Python programs beyond the built-in modules and are one of the main reasons why Python is popular today.

Before you learn how to install and use these packages, let's briefly go through the difference between a module, a package and a library.

Modules, libraries and packages

Modules and packages can easily be confused because of their similarities, but there are a few differences. Modules are similar to files, while packages are like directories that contain different files. Modules are generally written in a single file, but that's more of a practice than a definition.

Packages are essentially a type of module. Any module that contains the __path__ definition is a package. Packages, when viewed as a directory, can contain sub-packages and other modules. Modules, on the other hand, can contain classes, functions and data members just like any other Python file.

Library is a term that's used interchangeably with imported packages. But in general practice, it refers to a collection of packages.

Despite the differences between modules, packages and libraries, you can import any of them using import statements.

Third-party package add-ons of Python can be found in the Python Package Index. To install packages that aren't a part of the standard library programmers use ‘pip’ (package installer for Python). It is installed with Python by default. To use pip, you need to be familiar with either the terminal if you're using a Mac or the command line interface if you're using Windows.

Alternatively, you can also use the terminal window present inside your IDE. When you are using the command line or terminal, you must make sure that you are installing packages in the same Python interpreter that you are working with inside your IDE.

While pip usually comes installed with Python by default it might be necessary to check the status of pip installation. If you're using a Mac, run the following command in the terminal:

```
1    python -m ensurepip --upgrade
```

If you're using Windows, you run the following command:

```
1    py -m ensurepip --upgrade
```

Once you verify its installation, you can install packages using pip on your machine.

The standard format for using pip command on MacOS is:

```
1    python3 -m pip install "SomeProject"
```

In the case of Windows, it is:

```
1    py -m pip install "SomeProject"
```

Let's examine an example. If you want to install a third-party package such as ‘requests’, you run the following command on a Mac:

```
python3 -m pip install requests
```

For Windows you'd run this one:

```
py -m pip install requests
```

Note: Package names as well as the command line or terminal are case-sensitive.

Once you have installed the package, you will be able to use the package directly inside your Python code. This is a one-time installation and the package will be present as a part of your Python interpreter until you choose to uninstall it.
The packages that you can install often have a number of classes, functions, sub-packages and members. These can be understood by using the package and finding examples that other programmers have posted on different websites. This will give you a better understanding of what functionality within that package needs more attention than others.
Additionally, it is also a good practice to look up the documentation of the packages. In some cases, you can use the Python Package Index pypi website. In other cases, the packages are built and maintained by open-source communities and you can find their information on a standalone website created for it or on a version control system such as GitHub. Documentation in most of the popular Python libraries today is pretty elaborate and should have good examples to get you started.
Sub-packages
If we are to assume that packages are similar to a folder or directory in our operating system, then the package can also contain other directories. Packages, both built-in and user-defined, can contain other folders within them that need to be accessed. These are named sub-packages. You use dot-notation to access sub-packages within a package you have imported. For example, in a package such as matplotlib, the contents that are used most commonly are present inside the subpackage pyplot. Pyplot eventually may consist of various functions and attributes.
The code for importing a sub-package is:

```
import matplotlib.pyplot
```

To make it even more convenient, it is often imported using an alias. So more commonly, you will come across code such as:

```
import matplotlib.pyplot as plt
```

You could use any other word as an alias instead of plt, but it is a common convention.

You can explore the directory structure of such packages usually by searching for the module index of that package.