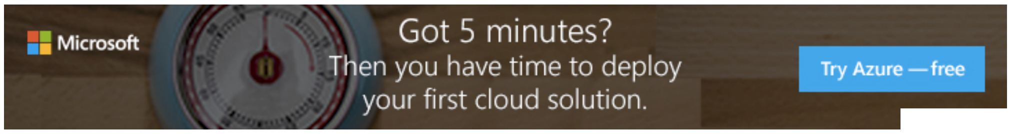
Whats the difference between a module and a library in Python?



I have background in Java and I am new to Python. I want to make sure I understand correctly Python terminology before I go ahead.

My understanding of a **module** is: a script which can be imported by many scripts, to make reading easier. Just like in java you have a class, and that class can be imported by many other classes.

My understanding of a **library** is: A library contains many **modules** which are separated by its use.

My question is: Are libraries like packages, where you have a package e.g. called `food`, then:

> - chocolate.py
> - sweets.py
> - biscuts.py

are contained in the `food` package?

Or do libraries use packages, so if we had another package `drink`:

- milk.py
- juice.py

contained in the package. The `library` contains two packages?

Also, an application programming interface (API) usually contains a set of libraries is this at the top of the hierarchy:

1. API
2. Library
3. Package
4. Module
5. Script

So an API will consist off all from 2-5?

python

edited Feb 16 '16 at 14:17
Dataman
514 3 15

asked Oct 5 '13 at 13:08
joker
321 2 3 13

Python uses the term "package" and not very much "library" (apart from the Standard Library). – John Zwinck Oct 5 '13 at 13:14

2 Answers

From The Python Tutorial - Modules

- **Module**:

  > A module is a file containing Python definitions and statements. The file name is the module name with the suffix `.py` appended.

- **Package**:

  > Packages are a way of structuring Python's module namespace by using "dotted module names".

If you read the documentation for the `import` statement gives more details, for example:

> Python has only one type of **module object**, and all modules are of this type, regardless of whether the module is implemented in Python, C, or something else. To help organize modules and provide a naming hierarchy, Python has a concept of packages.
>
> You can think of packages as the directories on a file system and modules as files within directories, but don't take this analogy too literally since packages and modules need not originate from the file system. For the purposes of this documentation, we'll use this convenient analogy of directories and files. Like file system directories, packages are organized hierarchically, and packages may themselves contain subpackages, as well as regular modules.
>
> It's important to keep in mind that **all packages are modules, but not all modules are packages. Or put another way, packages are just a special kind of module. Specifically, any module that contains a `__path__` attribute is considered a package.**

Hence the term `module` refers to a specific entity: it's a class whose instances are the `module` objects you use in python programs. It is also used, by analogy, to refer to the file in the file system from which these instances "are created".

The term *script* is used to refer to a module whose aim is to be executed. It has the same meaning as "program" or "application", but it is *usually* used to describe simple and small programs(i.e. a single file with at most some hundreds of lines). Writing a script takes minutes or few hours.

The term *library* is simply a generic term for a bunch of code that was designed with the aim of being usable by many applications. It provides some generic functionality that can be used by specific applications.

When a module/package/something else is "published" people often refer to it as a library. Often libraries contain a package or multiple related packages, but it could be even a single module.

Libraries usually do not provide any specific functionality, i.e. you cannot "run a library".

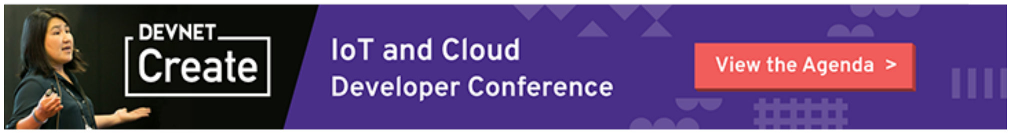
The API can have different meanings depending on the context. For example:

- it can define a protocol like the DB API or the buffer protocol.
- it can define how to interact with an application(e.g. the `Python/C API`)
- when related to a library/package it simply the interface provided by that library for its functionality(set of functions/classes/constants etc.)

In any case an API is *not* python code. It's a description which may be more or less formal.

edited Oct 5 '13 at 13:46 answered Oct 5 '13 at 13:40

Bakuriu
50.2k 7 95 134



Only *package* and *module* have a well-defined meaning specific to Python.

1. An **API** is not a collection of code *per se* - it is more like a "protocol" specification how various parts (usually libraries) communicate with each other. There are a few notable "standard" APIs in python. E.g. the DB API
2. In my opinion, a **library** is anything that is not an *application* - in python, a library is a *module* - usually with *submodules*. The scope of a library is quite variable - for example the python

standard library is vast (with quite a few submodules) while there are lots of single purpose libraries in the PyPi, e.g. a backport of `collections.OrderedDict` for py < 2.7

3. A **package** is a collection of python modules under a common namespace. In practice one is created by placing multiple python modules in a directory with a special `__init__.py` module (file).
4. A **module** is a single file of python code that is meant to be *imported*. This is a bit of a simplification since in practice quite a few modules detect when they are run as script and do something special in that case.
5. A **script** is a single file of python code that is meant to be *executed* as the 'main' program.
6. If you have a set of code that spans multiple files, you probably have an **application** instead of script.

edited Oct 5 '13 at 13:37

answered Oct 5 '13 at 13:32
Kimvais
20.2k 7 71 106