Package
&
Module

Introduction

When you work on python projects, it’s not a good practice to have all you python code in one single python file (.py).
You are better off splitting your code, classes, functions and variables thoughtfully in separate python files (.py files), aka **modules**. Python allows you to import code in one module for use in other modules.
This will:

1. Make your code modular, there by make the python objects reusable across modules.
2. Allows you to focus on a small part of problem at a time without disturbing the whole.
3. Makes bug fixing easier.
4. Allow multiple developers to contribute to your project effectively
5. Organize the code and maintain the project a lot more easier.

A package is a container that contains various functions to perform specific tasks. For example, the `math` package includes the `sqrt()` function to perform the square root of a number.

While working on big projects, we have to deal with a large amount of code, and writing everything together in the same file will make our code look messy. Instead, we can separate our code into multiple files by keeping the related code together in packages.

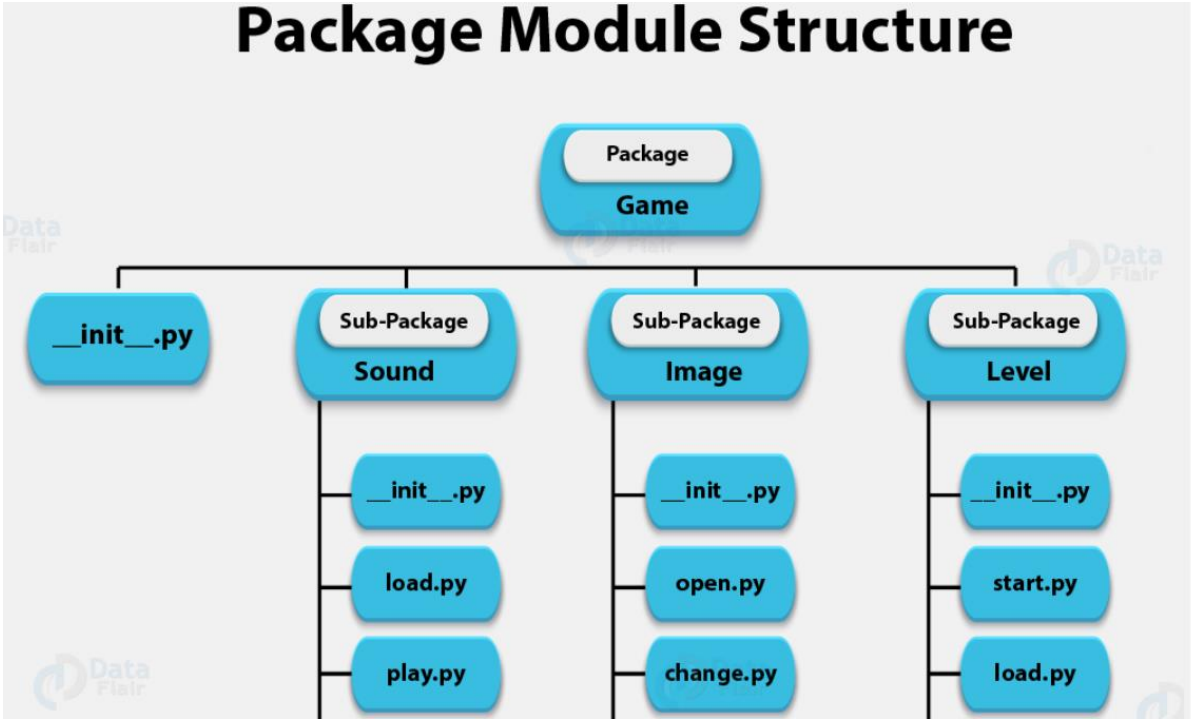
Now, we can use the package whenever we need it in our projects. This way we can also reuse our code.

Structure of Python Packages

As we discussed, a package may hold other Python packages and modules. But what distinguishes a package from a regular directory?
Well, a Python package may have an __init__.py file in the directory.

You may leave it empty, or you may store initialization code in it. But if your directory does not have an __init__.py file, it is namespace a package; it is just a directory with a bunch of Python scripts. Leaving __init__.py empty is indeed good practice.

Take a look at the following structure for a game:



What are Python Modules?

A module is a pythonic statement which contains multiple functions in it. Modules act as a pre-defined library in the code, which is accessible to both coder and user.

The python modules also store pre-defined functions from the library while the execution of code is going on.

Example of Python Module:

```
import math
from math import pow
pow(2,8)
print(pow)
```

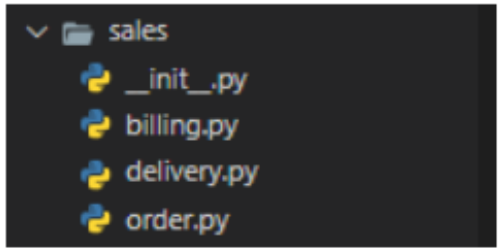
What is a Module and Package?

A Python **module** is any Python files with a .py extension. It can be imported into python without the .py part.

A Python **package** is nothing but a collection of modules along with a __init__.py file. The modules can also be arranged in hierarchy of folders inside a package.

Just by adding an empty __init__.py file to the in the folder, Python knows it is a Package.
In fact, a package is also really a module that contains other modules.
Python provides a wide variety of modules as standard modules.

For example, the following picture shows the `sales` package that contains three modules including `order`, `delivery`, and `billing`:



Importing packages

To import a package, you use the `import` statement like this:

```
import package.module
```

And to access an object from a module that belongs to a package, you use the dot notation:

```
package.module.function
```

The following shows how to use functions in the `order`, `delivery`, and `billing` modules from the `sales` package:

```
# main.py
import sales.order
import sales.delivery
import sales.billing



sales.order.create_sales_order()
sales.delivery.create_delivery()
sales.billing.create_billing()
```

To make the code more concise, you can use the following statement to import a function from a module:

```
from <module> import <function>
```

For example:

```
# main.py
from sales.order import create_sales_order
from sales.delivery import create_delivery
from sales.billing import create_billing



create_sales_order()
create_delivery()
create_billing()
```

It's possible to rename the object when importing it:

```python
# main.py
from sales.order import create_sales_order as create_order
from sales.delivery import create_delivery as start_delivery
from sales.billing import create_billing as issue_billing



create_order()
start_delivery()
issue_billing()
```

In this example, we rename...

- The `create_sales_order` to `create_order`,
- The `create_delivery` to `start_delivery`,
- and the `create_billing` to `issue_billing`.

Initializing a package

By convention, when you import a package, Python will execute the `__init__.py` in that package.

Therefore, you can place the code in the `__init__.py` file to initialize package-level data.

The following example defines a default tax rate in the `__init__.py` of the sales package:

```
# __init__.py

# default sales tax rate
TAX_RATE = 0.07
```

From the `main.py` file, you can access the `TAX_RATE` from the `sales` package like this:

```
# main.py
from sales import TAX_RATE

print(TAX_RATE)
```

from <package> import *

When you use the statement to import all objects from a package:

```
from <package> import *
```

Python will look for the `__init__.py` file.

If the `__init__.py` file exists, it'll load all modules specified in a special list called `__all__` in the file.

For example, you can place the order and delivery modules in the `__all__` list like this:

```
# __init__.py

__all__ = [
    'order',
    'delivery'
]
```

And use the following import statement in the main.py:

```
# main.py
from sales import *



order.create_sales_order()
delivery.create_delivery()

# cannot access the billing module
```

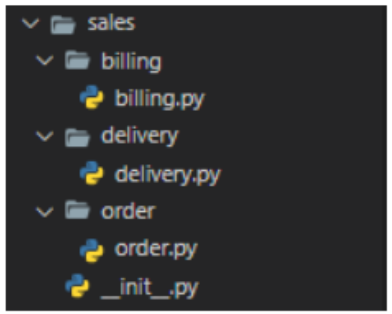
From the main.py, you can access functions defined in the `order` and `delivery` modules. But you cannot see the `billing` module because it isn't in the `__all__` list.

Subpackages

Packages can contain subpackages. The subpackages allow you to further organize modules.

The following shows the `sales` package that contains three subpackages: `order`, `delivery`, and `billing`. Each subpackage has the corresponding module.

For example, you can place all other modules related to the order processing in the `order` subpackage:



Everything you've learned about packages is also relevant to subpackages.

For example, to import a function from the `order` subpackage, you use the following `import` statement:

```
# main.py
from sales.order import create_sales_order


create_sales_order()
```

Summary

- A Python package contains one or more modules. Python uses the folders and files structure to manage packages and modules.
- Use the `__init__.py` file if you want to initialize the package-level data.
- Use `__all__` variable to specify the modules that will load automatically when importing the package.
- A package may contain subpackages.

Regular Package (With __init__.py)

Namespace Package (Without __init__.py)

- **Before 3.3 __init__.py file was needed to add in any directory to create a package**
- **From python 3.3+ , namespace packages were introduced.**
- **Which does not need a init file.**
- **And gives flexibility to have sub_packages on different directories.**