Python Packages

Summary: in this tutorial, you learn about the Python packages and how to use them to structure your application.

Introduction to Python packages

Suppose that you need to develop a large application that handles the sales process from order to cash.

The application will have many modules (https://www.pythontutorial.net/python-basics/python-module/) . When the number of modules grows, it'll become difficult to keep all of them in one location.

And you may want to group modules into something meaningful.

This is where packages come into play.

Packages allow you to organize modules in the hierarchical structure.

The way Python organizes packages and modules like the Operating System structures the folders and files.

To create a package, you create a new folder and place the relevant modules in that folder.

To instruct Python to treat a folder containing files as a package, you need to create a `__init__.py` file in the folder.

> Note that starting with Python 3.3, Python introduced the implicit namespace packages feature. This allows Python to treat a folder as a package without the `__init__.py` .

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:

```
# 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)
```

In addition to initializing package-level data, the `__init__.py` also allows you to automatically import modules from the package.

For example, in the `__init__.py`, if you place the following statement:

```
# __init__.py

# import the order module automatically
```

```
from sales.order import create_sales_order

# default sales tax rate
TAX_RATE = 0.07
```

And import the `sales` package from main.py file, the `create_sales_order` function will be automatically available as follows:

```
# main.py
import sales

sales.order.create_sales_order()
```

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.