Python Modules

Summary: in this tutorial, you'll learn about Python modules, how to import objects from a module, and how to develop your modules.

Introduction to Python modules

A module is a piece of software that has a specific functionality. A Python module is a file that contains Python code.

For example, when building a shopping cart application, you can have one module for calculating prices and another module for managing items in the cart. Each module is a separate Python source code file.

A module has a name specified by the filename without the `.py` extension. For example, if you have a file called `pricing.py`, the module name is `pricing`.

Writing Python modules

First, create a new file called `pricing.py` and add the following code:

```
# pricing.py

def get_net_price(price, tax_rate, discount=0):
    return price * (1 + tax_rate) * (1-discount)


def get_tax(price, tax_rate=0):
    return price * tax_rate
```

The pricing module has two functions that calculate the net price and tax from the selling price, tax rate, and discount.

Importing module objects

To use objects defined in a module from another file, you can use the `import` statement.

The `import` statement has several forms that we will discuss in the next sections.

1) import <module_name>

To use objects defined in a module, you need to import the module using the following `import` statement:

```
import module_name
```

For example, to use the `pricing` module in the `main.py` file, you use the following statement:

```
import pricing
```

When you import a module, Python executes all the code from the corresponding file. In this example, Python executes the code from the `pricing.py` file. Also, Python adds the module name to the current module.

This module name allows you to access functions, variables, etc. from the imported module in the current module. For example, you can call a function (https://www.pythontutorial.net/python-basics/python-functions/) defined in the imported module using the following syntax:

```
module_name.function_name()
```

The following shows how to use the `get_net_price()` function defined in the `pricing` module in the `main.py` file:

```
# main.py
import pricing


net_price = pricing.get_net_price(
    price=100,
    tax_rate=0.01
)

print(net_price)
```

Output:

```
101.0
```

2) import <module_name> as new_name

If you don't want to use the `pricing` as the identifier in the `main.py`, you can rename the module name to another as follows:

```
import pricing as selling_price
```

And then you can use the `selling_price` as the identifier in the `main.py`:

```
net_price = selling_price.get_net_price(
    price=100,
    tax_rate=0.01
)
```

3) from <module_name> import <name>

If you want to reference objects in a module without prefixing the module name, you can explicitly import them using the following syntax:

```
from module_name import fn1, fn2
```

Now, you can use the imported functions without specifying the module name like this:

```
fn1()
fn2()
```

This example imports the `get_net_price()` function from the `pricing` module:

```
# main.py
from pricing import get_net_price
```

and use the `get_net_price()` function from the `pricing` module:

```
# main.py
from pricing import get_net_price

net_price = get_net_price(price=100, tax_rate=0.01)
print(net_price)
```

4) from <module_name> import <name> as <new_name>: rename the imported objects

It's possible to rename an imported name to another by using the following `import` statement:

```
from <module_name> import <name> as <new_name>
```

The following example renames the `get_net_price()` function from the `pricing` module to `calculate_net_price()` function:

```
from pricing import get_net_price as calculate_net_price

net_price = calculate_net_price(
    price=100,
    tax_rate=0.1,
    discount=0.05
)
```

5) from <module_name> import * : import all objects from a module

To import every object from a module, you can use the following syntax:

```
from module_name import *
```

This `import` statement will import all public identifiers including variables (https://www.pythontutorial.net/python-basics/python-variables/) , constants (https://www.pythontutorial.net/python-basics/python-constants/) , functions (https://www.pythontutorial.net/python-basics/python-functions/) , classes (https://www.pythontutorial.net/python-oop/python-class/) , etc., to the program.

However, it's not a good practice because if the imported modules have the same object, the object from the second module will override the first one. The program may not work as you would expect.

Let's see the following example.

First, create a new file called `product.py` and define the `get_tax()` function:

```
# product.py
def get_tax(price):
    return price * 0.1
```

Both `product` and `pricing` modules have the `get_tax()` function. However, the `get_tax()` function from the `product` module has only one parameter while the `get_tax()` function from the `pricing` module has two parameters.

Second, import all objects from both `pricing` and `product` modules and use the `get_tax()` function:

```
from pricing import *
from product import *

tax = get_tax(100)
print(tax)
```

Since the `get_tax()` function from the `product` module overrides the `get_tax` () function from the `pricing` module, you get the `tax` with a value of 10.

Summary

- A module is a Python source code file with the `.py` extension. The module name is the Python file name without the extension.
- To use objects from a module, you import them using the `import` statement.