Python Modules

As our program grows bigger, it may contain many lines of code. Instead of putting everything in a single file, we can use modules to separate codes in separate files as per their functionality. This makes our code organized and easier to maintain.

Module is a file that contains code to perform a specific task. A module may contain variables, functions, classes etc. Let's see an example,

Let us create a module. Type the following and save it as `example.py`.

```
# Python Module addition

def add(a, b):

   result = a + b
   return result
```

Here, we have defined a function `add()` inside a module named `example`. The function takes in two numbers and returns their sum.

Import modules in Python

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the `import` keyword to do this. To import our previously defined module `example`, we type the following in the Python prompt.

```
import example
```

This does not import the names of the functions defined in `example` directly in the current symbol table. It only imports the module name `example` there.

Using the module name we can access the function using the dot `.` operator. For example:

```
example.add(4,5) # returns 9
```

Note:

- Python has tons of standard modules. You can check out the full list of Python standard modules and their use cases.
- Standard modules can be imported the same way as we import our user-defined modules.

Import Python Standard Library Modules

The Python standard library contains well over **200** modules. We can import a module according to our needs.

Suppose we want to get the value of `pi`, first we import the math module and use `math.pi`. For example,

```
# import standard math module
import math

# use math.pi to get value of pi
print("The value of pi is", math.pi)
```

Output

```
The value of pi is 3.141592653589793
```

Python import with Renaming

In Python, we can also import a module by renaming it. For example,

```
# import module by renaming it
import math as m

print(m.pi)

# Output: 3.141592653589793
```

Here, We have renamed the `math` module as `m`. This can save us typing time in some cases.

Note that the name `math` is not recognized in our scope. Hence, `math.pi` is invalid, and `m.pi` is the correct implementation.

Python from...import statement

We can import specific names from a module without importing the module as a whole. For example,

```
# import only pi from math module
from math import pi

print(pi)

# Output: 3.141592653589793
```

Here, we imported only the `pi` attribute from the `math` module.

Import all names

In Python, we can import all names(definitions) from a module using the following construct:

```
# import all names from the standard module math
from math import *

print("The value of pi is", pi)
```

Here, we have imported all the definitions from the math module. This includes all names visible in our scope except those beginning with an underscore(private definitions).

Importing everything with the asterisk (*) symbol is not a good programming practice. This can lead to duplicate definitions for an identifier. It also hampers the readability of our code.

The dir() built-in function

In Python, we can use the dir() function to list all the function names in a module.

For example, earlier we have defined a function `add()` in the module `example`.

We can use `dir` in `example` module in the following way:

```
print(dir(example))
```

```
['__builtins__',
'__cached__',
'__doc__',
'__file__',
'__initializing__',
'__loader__',
'__name__',
'__package__',
'add']
```

Here, we can see a sorted list of names (along with `add`). All other names that begin with an underscore are default Python attributes associated with the module (not user-defined).

For example, the `__name__` attribute contains the name of the module.

```
import example
```

```
example.__name__

# Output: 'example'
```

All the names defined in our current namespace can be found out using the `dir()` function without any arguments.

```
a = 1
b = "hello"

import math

print(dir())
```

```
['__builtins__', '__doc__', '__name__', 'a', 'b', 'math', 'pyscripter']
```

Table of Contents

- Introduction
- Import modules in Python
- Import Python Standard Library Modules
- Python import with Renaming
- Python from...import statement
- Import all names
- The dir() built-in function