Modules in Python

A Python module is a file containing Python definitions and statements.

A module can define functions, classes, and variables.

A module can also include runnable code.

Grouping related code into a module makes the code easier to understand and use.

It also makes the code logically organized.

Create a Python Module

create a simple calc.py in which we define two functions, one **add** and another **subtract**.

```
# A simple module, calc.py
def add(x, y):
    return (x+y)

def subtract(x, y):
    return (x-y)
```

Import module in Python

We can import the functions, and classes defined in a module to another module using the import statement in some other Python source file.

When the interpreter encounters an import statement, it imports the module if the module is present in the search path. A search path is a list of directories that the interpreter searches for importing a module.

Syntax

```
import module
```

Note: This does not import the functions or classes directly instead imports the module only. To access the functions inside the module the dot(.) operator is used.

```
# importing  module calc.py
import calc

print(calc.add(10, 2))
```

Variables in Module

The module can contain functions, as already described, but also variables of all types (arrays, dictionaries, objects etc):

Save this code in the file `mymodule.py`

```
person1 = {
  "name": "John",
  "age": 36,
  "country": "Norway"
}
```

Import the module named mymodule, and access the person1 dictionary:

```
import mymodule

a = mymodule.person1["age"]
print(a)
```

Python Import From Module

Python's *from* statement lets you import specific attributes from a module without importing the module as a whole.

```
# importing sqrt() and factorial from the
# module math
from math import sqrt, factorial

# if we simply do "import math", then
# math.sqrt(16) and math.factorial()
# are required.
print(sqrt(16))
print(factorial(6))
```

Import all Names

The * symbol used with the import statement is used to import all the names from a module to a current namespace.

```
from module_name import *
```

The use of * has its advantages and disadvantages. If you know exactly what you will be needing from the module, it is not recommended to use *, else do so.

```
# importing sqrt() and factorial from the
# module math
from math import *

# if we simply do "import math", then
# math.sqrt(16) and math.factorial()
# are required.
print(sqrt(16))
print(factorial(6))
```

Renaming the Python module

We can rename the module while importing it using the keyword.

Syntax: *Import* ***Module_name*** *as* ***Alias_name***

```
# importing sqrt() and factorial from the
# module math
import math as mt

# if we simply do "import math", then
# math.sqrt(16) and math.factorial()
# are required.
print(mt.sqrt(16))
print(mt.factorial(6))
```

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.