Module

A module is a file containing Python definitions and statements.

A module is a file containing group of variables, methods, function and classes etc.

They are executed only the first time the module name is encountered in an import statement.

The file name is the module name with the suffix .py appended.

Ex:- mymodule.py

Type of Modules:-

- User-defined Modules
- Built-in Modules

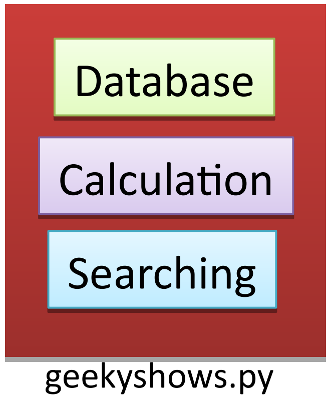
    Ex:- array, math, numpy, sys

When and Why use Module

Assume that you are building a very large project, it will be very difficult to manage all logic within one single file so If you want to separate your similar logic to a separate file, you can use module.

It will not only separate your logics but also help you to debug your code easily as you know which logic is defined in which module.

When a module is developed, it can be reused in any program that needs that module.

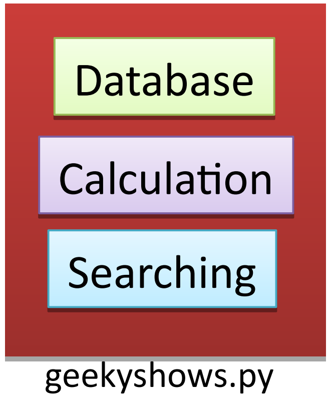


db.py

cal.py

sarch.py

Creating a Module



db.py

cal.py

sarch.py

```
def add(a, b):
        return a+b
def sub(a, b):
        return a-b
```

cal.py

How to use Module

import statement is used to import modules.

Syntax:-

import module_name

import module_name as alias_name

from module_name import function_name1, function_name2, ... , function_nameN

from module_name import f_name as alias_f_name

from module_name import *

Note - Modules can import other modules.

import module_name

This does not enter the names of the functions defined in module directly in the current symbol table; it only enters the module name there.

Ex:- import cal

How to access Methods, Functions, Variable and Classes ?

Using the module name you can access the functions.

Syntax:- module_name.function_name()

Ex:-

```
cal.add(10, 20)
cal.sub(20, 10)
add = cal.add
add(10, 20)
```

When 2 modules having same function name then This import module is good approach to use.

import module_name as alias_name

This does not enter the names of the functions defined in module directly in the current symbol table; it only enters the module name there. If the module name is followed by *as*, then the name following as is bound directly to the imported module.

Ex:- import cal as c

How to access Methods, Functions, Variable and Classes ?

Using the alias_name you can access the functions.

Ex:-

```
c.add(10, 20)
c.sub(20, 10)
add = c.add
add(10, 20)
```

from module_name import function_name

There is a variant of the import statement that imports names from a module directly into the importing module's symbol table.

Ex:- from cal import add, sub

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it's name.

Ex:-

add(10, 20)

sub(20, 10)

<u>from module_name import f_name as a_name</u>

Ex:- from cal import add as s

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it's alias name.

Ex:-

s(10, 20)

from module_name import *

This imports all names except those beginning with an underscore (_).

Ex:- from cal import *

How to access Methods, Functions, Variable and Classes ?

You can access the functions directly by it’s name.

Ex:-

add(10, 20)

sub(20, 10)

Module Search Path

When a module named cal is imported, the interpreter first searches for a built-in module with that name. If not found, it then searches for a file named cal.py in a list of directories given by the variable sys.path.

sys.path is initialized from these locations:

- Current Directory
- If not found then searches each directory in the Shell variable PYTHONPATH
- If not found then searches installation-dependent default path.

PYTHONPATH is a list of directory names, with the same syntax as the shell variable PATH