Modules

- 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.

❖ Advantages of Modularizing

➢ **Simplification:** A module often concentrates on one comparatively small area of the overall problem instead of the full task. We will have a more manageable design problem to think about if we are only concentrating on one module. Program development is now simpler and much less vulnerable to mistakes.

➢ **Flexibility:** Modules are frequently used to establish conceptual separations between various problem areas. It is less likely that changes to one module would influence other portions of the program if modules are constructed in a fashion that reduces interconnectedness. (We might even be capable of editing a module despite being familiar with the program beyond it.) It increases the likelihood that a group of numerous developers will be able to collaborate on a big project.

➢ **Reusability:** Functions created in a particular module may be readily accessed by different sections of the assignment (through a suitably established api). As a result, duplicate code is no longer necessary.

➢ **Scope:** Modules often declare a distinct namespace to prevent identifier clashes in various parts of a program.

❖ Create a Python Module

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

Example:

```
# 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.

Syntax:

```
import module
```

Example:

```
# importing module calc.py
import calc


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

Output:

```
12
```

❖ Locating Python Modules

Whenever a module is imported in Python the interpreter looks for several locations.

First, it will check for the built-in module, if not found then it looks for a list of directories defined in the sys.path. Python interpreter searches for the module in the following manner –

- First, it searches for the module in the current directory.
- If the module isn’t found in the current directory, Python then searches each directory in the shell variable PYTHONPATH. The PYTHONPATH is an environment variable, consisting of a list of directories.
- If that also fails python checks the installation-dependent list of directories configured at the time Python is installed.

❖ Renaming the Python module

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

Syntax:

*Import **Module_name** as **Alias_name***

Example:

```
# 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))
```

Output:

```
4.0

720
```

❖ Directories List for Modules

Here, sys.path is a built-in variable within the sys module. It contains a list of directories that the interpreter will search for the required module.

Example:

```
# importing sys module
import sys


# importing sys.path
print(sys.path)
```