**What are modules in Python?**

Modules refer to a file containing Python statements and definitions.

A file containing Python code, for example: example.py, is called a module, and its module name would be example.

We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.

We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

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

# Python Module example

def add(a, b):

"""This program adds two

numbers and return the result"""

result = a + b

return result

Here, we have defined a [function](https://www.programiz.com/python-programming/function) add() inside a module named example. The function takes in two numbers and returns their sum.

**How to 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(4,5.5)

9.5

Python has tons of standard modules. You can check out the full list of [Python standard modules](http://docs.python.org/3/py-modindex.html) and their use cases. These files are in the Lib directory inside the location where you installed Python.

Standard modules can be imported the same way as we import our user-defined modules.

There are various ways to import modules. They are listed below..

**Python import statement**

We can import a module using the import statement and access the definitions inside it using the dot operator as described above. Here is an example.

# import statement example

# to import standard module math

import math

print("The value of pi is", math.pi)

When you run the program, the output will be:

The value of pi is 3.141592653589793

**Import with renaming**

We can import a module by renaming it as follows:

# import module by renaming it

import math as m

print("The value of pi is", m.pi)

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. Here is an example.

# import only pi from math module

from math import pi

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

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

In such cases, we don't use the dot operator. We can also import multiple attributes as follows:

>>> from math import pi, e

>>> pi

3.141592653589793

>>> e

2.718281828459045

**Import all names**

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.

**Python Module Search Path**

While importing a module, Python looks at several places. Interpreter first looks for a built-in module. Then(if built-in module not found), Python looks into a list of directories defined in sys.path. The search is in this order.

* The current directory.
* PYTHONPATH (an environment variable with a list of directories).
* The installation-dependent default directory.

>>> import sys

>>> sys.path

['',

'C:\\Python33\\Lib\\idlelib',

'C:\\Windows\\system32\\python33.zip',

'C:\\Python33\\DLLs',

'C:\\Python33\\lib',

'C:\\Python33',

'C:\\Python33\\lib\\site-packages']

We can add and modify this list to add our own path.

**Reloading a module**

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works.

Suppose we have the following code in a module named my\_module.

# This module shows the effect of

# multiple imports and reload

print("This code got executed")

Now we see the effect of multiple imports.

>>> import my\_module

This code got executed

>>> import my\_module

>>> import my\_module

We can see that our code got executed only once. This goes to say that our module was imported only once.

Now if our module changed during the course of the program, we would have to reload it.One way to do this is to restart the interpreter. But this does not help much.

Python provides a more efficient way of doing this. We can use the reload() function inside the imp module to reload a module. We can do it in the following ways:

>>> import imp

>>> import my\_module

This code got executed

>>> import my\_module

>>> imp.reload(my\_module)

This code got executed

<module 'my\_module' from '.\\my\_module.py'>

**The dir() built-in function**

We can use the dir() function to find out names that are defined inside a module.

For example, we have defined a function add() in the module example that we had in the beginning.

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

>>> 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\_\_

'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

>>> dir()

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', 'a', 'b', 'math', 'pyscripter']

## What are packages?

We don't usually store all of our files on our computer in the same location. We use a well-organized hierarchy of directories for easier access.

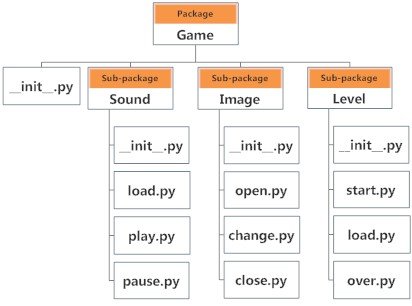
Similar files are kept in the same directory, for example, we may keep all the songs in the "**music**" directory. Analogous to this, Python has packages for directories and [modules](https://www.programiz.com/python-programming/modules) for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similarly, as a directory can contain subdirectories and files, a Python package can have sub-packages and modules.

A directory must contain a file named \_\_init\_\_.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

Here is an example. Suppose we are developing a game. One possible organization of packages and modules could be as shown in the figure below.

Package Module Structure in Python Programming

## Importing module from a package

We can import modules from packages using the dot (.) operator.

For example, if we want to import the start module in the above example, it can be done as follows:

import Game.Level.start

Now, if this module contains a [function](https://www.programiz.com/python-programming/function) named select\_difficulty(), we must use the full name to reference it.

Game.Level.start.select\_difficulty(2)

If this construct seems lengthy, we can import the module without the package prefix as follows:

from Game.Level import start

We can now call the function simply as follows:

start.select\_difficulty(2)

Another way of importing just the required function (or class or variable) from a module within a package would be as follows:

from Game.Level.start import select\_difficulty

Now we can directly call this function.

select\_difficulty(2)

Although easier, this method is not recommended. Using the full [namespace](https://www.programiz.com/python-programming/namespace) avoids confusion and prevents two same identifier names from colliding.

While importing packages, Python looks in the list of directories defined in sys.path, similar as for [module search path](https://www.programiz.com/python-programming/modules#search).