Importing modules

All Versions

Examples

Importing a module

Use the import statement:

```
import random

print(random.randint(1, 10))
>>> 4
```

import module will import a module and then allow you to reference its objects -- values, functions and classes, for example -- using the module.name syntax. In the above example, the random module is imported, which contains the randint function. So by importing random you can call randint with random.randint .

You can import a module and assign it to a different name:

```
import random as rn
print(rn.randint(1, 10))
>>> 4
```

If your python file main.py is in the same folder as custom.py . You can import it like this:

```
import custom
```

It is also possible to import a function from a module:

```
from math import sin
sin(1)
>>> 0.8414709848078965
```

To import specific functions deeper down into a module, the dot operator may be used **only** on the left side of the import keyword:

```
from urllib.request import urlopen
```

In python, we have two ways to call function from top level. One is import and another is from . We should use import when we have a possibility of name collision. Suppose we have hello.py file and world.py files having same function named function . Then import statement will work good.

```
from hello import function
from world import function

function() #world's function will be invoked. Not hello's
```

In general import will provide you a namespace.

```
import hello
import world

hello.function() # exclusively hello's function will be invoked
world.function() # exclusively world's function will be invoked
```

But if you are sure enough, in your whole project there is no way having same function name you should use from statement

Multiple imports can be made on the same line:

```
# Multiple modules
import time, sockets, random
# Multiple functions
from math import sin, cos, tan
# Multiple constants
from math import pi, e

print(pi)
>>> 3.141592653589793
print(cos(45))
>>> 0.5253219888177297
print(time.time())
>>> 1482807222.7240417
```

The keywords and syntax shown above can also be used in combinations:

```
from urllib.request import urlopen as geturl, pathname2url as path2url, getproxies
from math import factorial as fact, gamma, atan as arctan
import random.randint, time, sys

print(time.time())
>>> 1482807222.7240417
print(arctan(60))
>>> 1.554131203080956
filepath = "/dogs/jumping poodle (december).png"
print(path2url(filepath))
>>> /dogs/jumping%20poodle%20%28december%29.png
```

The __all__ special variable

Modules can have a special variable named `__all__` to restrict what variables are imported when using `from mymodule import *` .

Given the following module:

```
# mymodule.py

__all__ = ['imported_by_star']

imported_by_star = 42
not_imported_by_star = 21
```

Only `imported_by_star` is imported when using `from mymodule import *` :

```
>>> from mymodule import *
>>> imported_by_star
42
>>> not_imported_by_star
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'not_imported_by_star' is not defined
```

However, `not_imported_by_star` can be imported explicitly:

```
>>> from mymodule import not_imported_by_star
>>> not_imported_by_star
21
```

Importing all names from a module

```
from module_name import *
```

for example:

```
from math import *
sqrt(2)    # instead of math.sqrt(2)
ceil(2.7)  # instead of math.ceil(2.7)
```

This will import all names defined in the `math` module into the global namespace, other than names that begin with an underscore (which indicates that the writer feels that it is for internal use only).

Warning : If a function with the same name was already defined or imported, it will be **overwritten** . Almost always importing only specific names `from math import sqrt, ceil` is the **recommended way** :

```
def sqrt(num):
    print("I don't know what's the square root of {}.".format(num))

sqrt(4)
# Output: I don't know what's the square root of 4.

from math import *
sqrt(4)
# Output: 2.0
```

Starred imports are only allowed at the module level. Attempts to perform them in class or function definitions result in a `SyntaxError` .

```
def f():
    from math import *
```

and

and

```
class A:
    from math import *
```

both fail with:

```
SyntaxError: import * only allowed at module level
```

Import modules from an arbitrary filesystem location

If you want to import a module that doesn't already exist as a built-in module in the Python Standard Library nor as a side-package, you can do this by adding the path to the directory where your module is found to sys.path . This may be useful where multiple python environments exist on a host.

```
import sys
sys.path.append("/path/to/directory/containing/your/module")
import mymodule
```

It is important that you append the path to the *directory* in which mymodule is found, not the path to the module itself.

Importing specific names from a module

Instead of importing the complete module you can import only specified names:

```
from random import randint # Syntax "from MODULENAME import NAME1[, NAME2[, ...]]"
print(randint(1, 10))      # Out: 5
```

from random is needed, because the python interpreter has to know from which resource it should import a function or class and import randint specifies the function or class itself.

Another example below (similar to the one above):

```
from math import pi
print(pi)                  # Out: 3.14159265359
```

The following example will raise an error, because we haven't imported a module:

```
random.randrange(1, 10)    # works only if "import random" has been run before
```

Outputs:

```
NameError: name 'random' is not defined
```

The python interpreter does not understand what you mean with random . It needs to be declared by adding import random to the example:

```
import random
random.randrange(1, 10)
```

__import__() function

The __import__() function can be used to import modules where the name is only known at runtime

```
if user_input == "os":
    os = __import__("os")

# equivalent to import os
```

This function can also be used to specify the file path to a module

```
mod = __import__(r"C:/path/to/file/anywhere/on/computer/module.py")
```

PEP8 rules for Imports

Some recommended PEP8 style guidelines for imports:

1. Imports should on separate lines:

```
from math import sqrt, ceil      # Not recommended
from math import sqrt            # Recommended
from math import ceil
```

2. Order imports as follows at the top of the module:

   - Standard library imports
   - Related third party imports
   - Local application/library specific imports

3. Wildcard imports should be avoided as it leads to confusion in names in the current namespace. If you do `from module import *`, it can be unclear if a specific name in your code comes from `module` or not. This is doubly true if you have multiple `from module import *`-type statements.

4. Avoid using relative imports; use explicit imports instead.

Programmatic importing

Python 2.x ≥ 2.7

To import a module through a function call, use the `importlib` module (included in Python starting in version 2.7):

```
import importlib
random = importlib.import_module("random")
```

The `importlib.import_module()` function will also import the submodule of a package directly:

```
collections_abc = importlib.import_module("collections.abc")
```

For older versions of Python, use the `imp` module.

Python 2.x ≤ 2.7

Use the functions `imp.find_module` and `imp.load_module` to perform a programmatic import.

Taken from standard library documentation

```
import imp, sys
def import_module(name):
    fp, pathname, description = imp.find_module(name)
    try:
        return imp.load_module(name, fp, pathname, description)
    finally:
        if fp:
            fp.close()
```

Do **NOT** use `__import__()` to programmatically import modules! There are subtle details involving `sys.modules`, the `fromlist` argument, etc. that are easy to overlook which `importlib.import_module()` handles for you.

Re-importing a module

When using the interactive interpreter, you might want to reload a module. This can be useful if you're editing a module and want to import the newest version, or if you've monkey-patched an element of an existing module and want to revert your changes.

Note that you **can't** just `import` the module again to revert:

```
import math
math.pi = 3
print(math.pi)    # 3
import math
print(math.pi)    # 3
```

Python 2

Use the reload function:

Python 2.x ≥ 2.3

```
import math
math.pi = 3
print(math.pi)    # 3
reload(math)
print(math.pi)    # 3.141592653589793
```

Python 3

The reload function has moved to importlib :

Python 3.x ≥ 3.0

```
import math
math.pi = 3
print(math.pi)    # 3
from importlib import reload
reload(math)
print(math.pi)    # 3.141592653589793
```

Importing submodules

```
from module.submodule import function
```

This imports function from module.submodule .

Syntax

```
import module_name
import module_name.submodule_name
from module_name import *
from module_name import submodule_name [, class_name , function_name , ...etc]
from module_name import some_name as new_name
from module_name.submodule_name import class_name [, function_name , ...etc]
```

Parameters

Remarks

Importing a module will make Python evaluate all top-level code in this module so it *learns* all the functions, classes, and variables that the module contains. When you want a module of yours to be imported somewhere else, be careful with your top-level code, and encapsulate it into if `__name__ == '__main__':` if you don't want it to be executed when the module gets imported.