

Packages (1A)

Copyright (c) 2024 - 2015 Young W. Lim.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Modules and Packages

a **module** in Python is a single file
that contains Python code
– functions, statements, variables, and classes.

a self-contained unit of code
can be imported and used
in other programs or modules.

a **package**, is a collection of modules
organized in a directory

can group multiple related modules together
under a common namespace,

<https://www.sitepoint.com/python-modules-packages/>

Working with Modules

modules can be imported and used
in other **programs**, **modules**, and **packages**.

can make smaller, manageable, and logical units.

improve organization, reusability, and maintainability.

<https://www.sitepoint.com/python-modules-packages/>

Creating a simple module

put a number of related functions, variables, and classes in one **module**, and give the module any name we want

To create a **module** in Python, open up an IDE or text editor, create a file, and give it a descriptive name and a **.py** extension.

For this example, let's call it **sample.py** and enter in the following code:

<https://www.sitepoint.com/python-modules-packages/>

Creating a simple module

```
# sample.py

sample_variable = "This is a string variable in the sample.py module"

# A function in the module
def say_hello(name):
    return f"Hello, {name} welcome to this simple module."

# This is another function in the module
def add(a, b):
    return f"The sum of {a} + {b} is = {a+b}"

print(sample_variable)
print(say_hello("aaa"))
print(add(2, 3))
```

<https://www.sitepoint.com/python-modules-packages/>

Creating a simple module

contains a **variable** named **sample_variable**
whose value is the string
"This is a string variable in the sample.py module".

this module also contains two **function** definitions.

the **say_hello()** **function** takes in a **name** parameter,
and it returns a welcome message if we pass a name to it.

The **add()** **function** returns the sum of two numbers
a, b that have been passed to it.

<https://www.sitepoint.com/python-modules-packages/>

Creating a simple module

To run

`python sample.py`

`python3 sample.py`

*This is a string variable in the sample.py module
Hello, aaa welcome to this simple module.
The sum of 2 + 3 is = 5*

For one-off module usage, we can run it as a standalone, but most modules are made to be used in other modules or other parts of a Python program.

So to use **variables**, **functions**, and **classes** from one module in another module we have to **import** the module.

<https://www.sitepoint.com/python-modules-packages/>

Using the **import** statement

can use the **import** statement to make the contents of one **module** available for use in another module.

to use the contents of **sample.py** in another module, we just import it:

```
# another_module.py
```

```
import sample
```

```
print(sample.sample_variable)  
print(sample.say_hello("John"))  
print(sample.add(2, 3))
```

```
import sample
```

module **sample**
(sample.py)

```
sample.add(2, 3)
```

module **sample** function **add**
(sample.py)

```
# sample.py
```

```
sample_variable = "This is a string variable in the sample.py module"
```

```
# A function in the module
```

```
def say_hello(name):
```

```
    return f"Hello, {name} welcome to this simple module."
```

```
# This is another function in the module
```

```
def add(a, b):
```

```
    return f"The sum of {a} + {b} is = {a+b}"
```

```
print(sample_variable)
```

```
print(say_hello("aaa"))
```

```
print(add(2, 3))
```

<https://www.sitepoint.com/python-modules-packages/>

Using the **import** statement

shows how to **import** the **functions** from the **sample.py** module, making them available for use in the **another_module.py**.

```
# another_module.py
```

```
import sample
```

```
print(sample.sample_variable)  
print(sample.say_hello("John"))  
print(sample.add(2, 3))
```

module **sample**
(**sample.py**)

when we **import** a **module**,
we don't include the **.py** **extension**;

<https://www.sitepoint.com/python-modules-packages/>

Using the **from** statement

can also use the **from** keyword to **import** specific functions or **variables**.

can specify the **functions** or **variables** we want to use, using the **from** keyword:

```
# another_module.py
```

```
from sample import add
```

```
print(add(10, 4))
```

```
from sample import add
```

module **sample**
(**sample.py**)

function add

```
add(2, 3)
```

function add

specifically imported the **add()** function from the **sample module**.

a module has a large number of **functions** and **variables** defined in it 1 and we don't want to use all of them.

<https://www.sitepoint.com/python-modules-packages/>

Using the **from** statement

Another benefit of using the **from** keyword is that we'll run the imported function without **namespacing** it or **prefixing** it with the name of its parent module.

Instead, we'll use the function like we've defined it in the file where we're using it.

add(2, 3)



function add

<https://www.sitepoint.com/python-modules-packages/>

Using the **as** statement (1-1)

can use **as** to provide an alias or an alternate name for the **module**.

```
# another_module.py
```

```
import sample as sp
```

```
result = sp.add(5, 5)
print(result)
print(sp.say_hello("Jason"))
```

```
result = sample.add(5, 5)
print(result)
print(sample.say_hello("Jason"))
```

```
import sample as sp
```

module **sample**
(sample.py)

module alias **sp**
(sample.py)

```
sp.add(2, 3)
```

module alias **sp**
(sample.py)

function **add**

```
# sample.py
```

```
sample_variable = "This is a string variable in the sample.py module"
```

```
# A function in the module
```

```
def say_hello(name):
```

```
    return f"Hello, {name} welcome to this simple module."
```

```
# This is another function in the module
```

```
def add(a, b):
```

```
    return f"The sum of {a} + {b} is = {a+b}"
```

```
print(sample_variable)
```

```
print(say_hello("aaa"))
```

```
print(add(2, 3))
```

<https://www.sitepoint.com/python-modules-packages/>

Using the **as** statement (1-2)

This code shows an import of the sample module, where the module is being given an alternate name **sp**.

So using **sp** is just the same as calling sample.

Therefore, using the **alias**, we have access to the variables and functions, in the same way we could if we were using the original name.

At times, we may define module names that are quite long or unreadable.

Python provides a way of giving the module imports an alternate or alias, which we can use to refer to them in the modules we're importing them into.

sp.add(2, 3)
↑ ↑
module alias **sp** function add
(**sample.py**)

<https://www.sitepoint.com/python-modules-packages/>

Three import examples

Using those three methods, we're able to use the **variables** or **functions** from one **module** in another **module**, enhancing the readability of our application where we don't need to put the code in one file.

import sample

from sample **import** add

import sample **as** sp

from sample **import** add

module sample
(sample.py)

function add

add(2, 3)

function add

import sample

module sample
(sample.py)

sample.add(2, 3)

module sample function add
(sample.py)

import sample **as** sp

module sample
(sample.py)

module alias sp
(sample.py)

sp.add(2, 3)

module alias sp function add
(sample.py)

<https://www.sitepoint.com/python-modules-packages/>

Naming modules

While naming our **modules**, it's good practice to use lowercase letters and separate words with underscores.

If a **module** name might cause a name clash with a Python built-in keyword or module from a third-party library, consider using a different name or adding a prefix that's relevant to the project.

remember that names are case-sensitive in Python

Overall, using **modules** lets us create and organize our code in a readable and maintainable way.

common Python standard library modules.

<https://www.sitepoint.com/python-modules-packages/>

Package

A **package** in Python is a way of organizing related modules into a **directory**.

to group **modules** that serve a common purpose or are part of the same component.

when structuring larger **projects** or **libraries**.

modules are individual **files** containing Python code:
put related functions, classes, and variables within a **single file**.

In contrast, **packages** are **directories**
that contain multiple **modules** or **subpackages**.

by grouping related **modules**
a higher level of organization
more structured and maintainable projects.

<https://www.sitepoint.com/python-modules-packages/>

Building and managing packages (1)

While **packages** organize related code **modules** in one directory, just putting the **modules** in a directory doesn't make it a package.

For Python to identify a directory as a **package** or a **subpackage**, the directory must contain a special file named `__init__.py`.

This file notifies Python that the directory containing it should be treated as a **package** or a **subpackage**.

This file could be empty, and most of the time it is, but it can also contain initialization code, and it plays a vital role in Python's package structure and import mechanisms.

<https://www.sitepoint.com/python-modules-packages/>

Building and managing packages (2)

So using `__init__.py` tells Python that we are intentionally creating a **package**, thereby helping it differentiate between a **package** and an ordinary directory.

Packages can have a hierarchical structure, meaning we can create **subpackages** within our packages to further organize our code.

This enables finer and more controlled separation of components and functionality.

<https://www.sitepoint.com/python-modules-packages/>

Building and managing packages (3)

Consider the following example:

```
my_package/  
├── __init__.py  
├── module1.py  
└── subpackage/  
    ├── __init__.py  
    ├── submodule1.py  
    └── submodule2.py
```

my_package is the main **package**,
subpackage is a **subpackage** within **my_package**.

Both directories have an **__init__.py** file.

<https://www.sitepoint.com/python-modules-packages/>

Creating packages and sub-packages (1-1)

To create a **package**,
first create a directory
then we create an `__init__.py` file.
and create our **modules** in the directory
along with any **subpackages**.

```
calculator/  
├── __init__.py  
├── add.py  
├── subtract.py  
└── multiply.py
```

```
# add.py  
  
def add(a, b):  
    """  
    adds two numbers and returns the result.  
  
    :param a: First number.  
    :param b: Second number.  
    :return: Sum of a and b.  
    """  
  
    return a + b
```

```
# subtract.py  
  
def subtract(a, b):  
    """  
    subtracts two numbers and returns the result.  
  
    :param a: First number.  
    :param b: Second number.  
    :return: Difference of a and b.  
    """  
  
    return a - b
```

<https://www.sitepoint.com/python-modules-packages/>

Importing from packages - absolute import

Absolute imports are used to directly import **modules** or **subpackages** from the top-level package,

specify the full path to the **module** or **package** we want to import.

importing the **add** module from the **calculator** package:

an **external module** **calculate.py**

imports the **add()** function from the **add** module

using an **absolute import** by specifying the absolute path to the function.
calculator.add

```
calculator/  
├── __init__.py  
├── add.py  
├── subtract.py  
└── multiply.py
```

```
from calculator.add import add
```

package calculator module add (add.py) function add

```
# calculate.py  
  
from calculator.add import add  
  
result = add(5, 9)  
  
print(result)
```

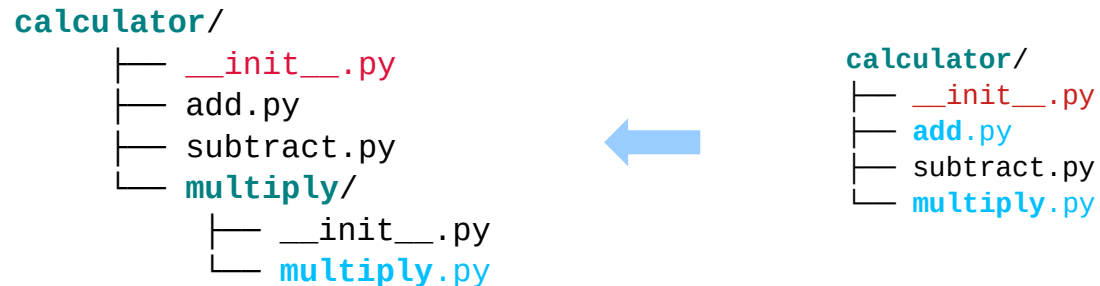
<https://www.sitepoint.com/python-modules-packages/>

Importing from packages - relative import (1)

relative imports are used to import modules or packages relative to the current module's position in the package hierarchy.

relative imports are specified using **dots** (.) to indicate the level of relative positioning.

let's create a subpackage in the **calculator** package,
call the **subpackage multiply**,
then move the **multiply.py** module into that subpackage,



<https://www.sitepoint.com/python-modules-packages/>

Importing from packages - relative import (2)

use **relative imports** to access the **multiply** module from other modules within the **calculator** package or its **subpackages**.

For instance, if we had a **module** inside the **calculator** package that needs to import the **multiply** module

```
calculator/  
├── __init__.py  
├── add.py  
├── subtract.py  
└── multiply/  
    ├── __init__.py  
    └── multiply.py
```

```
from .multiply import multiply
```

```
result = multiply(5, 9)  
print(result)
```

```
from .multiply import multiply
```

package
multiply

module multiply
(multiply.py)

Overall, **relative imports** are particularly useful for imports within a **package** and **subpackage** structure.

<https://www.sitepoint.com/python-modules-packages/>

The `__all__` attribute

when we want to use all **modules** from a **package** or **subpackages**, or all **functions** and **variables** from a **module**, so typing out all names becomes quite cumbersome.

so we want a way to specify that we're importing

all **functions** and **variables** that a **module** has or all **modules** that **package** offers.

Python has the `__all__` attribute, which is used in **modules** or **packages** to control what gets imported when a user uses the **`from module import *`** statement.

`__all__` attribute allows us to specify a list of names that will be considered “public” and will be imported when the wildcard (*) **`import`** is used.

In a **module**, we can define the `__all__` attribute to explicitly specify which names should be imported when the **`from module import *`** statement is used.

this helps prevent unintended imports of internal names, providing a clear way of showing the functions that can be imported publicly and that are for use only inside a module.

<https://www.sitepoint.com/python-modules-packages/>

Using the `__all__` attribute in modules (1-2)

```
# my_module.py

__all__ = ['public_function', 'public_variable']

def public_function():
    return "This is a public function."

def _internal_function():
    return "This is an internal function."

public_variable = "This is a public variable."
_internal_variable = "This is an internal variable."
```

defines a module named `my_module.py`, and with the `__all__` attribute being set, only the `public_function` and the `public_variable` will be imported when the `from my_module import *` is used.

The function and variable names starting with an underscore the `_internal_function` and the `_internal_variable` won't be imported.

<https://www.sitepoint.com/python-modules-packages/>

Using the `__all__` attribute in modules (2)

If we know the **absolute paths** to the functions starting with an underscore, the **`_internal_function`** and the **`_internal_variable`** we can still import them to our code.

However, that goes against the convention of **encapsulation**, since the **underscore** (`_`) denotes them as **private members** of the **module** and indicates that they shouldn't be used outside the module.

So it's good practice to follow Python programming conventions even if Python doesn't enforce strict encapsulation.

<https://www.sitepoint.com/python-modules-packages/>

Using the `__all__` attribute in packages (1)

The `__all__` attribute can also be used in `__init__.py` files within a **package** or **subpackage** to control the default behavior of **wildcard imports** for **submodules** or **subpackages**.

This can help ensure that only specific modules are imported when using **wildcard imports** on **packages**:

```
# my_package/__init__.py
```

```
__all__ = ['submodule1', 'subpackage']
```

```
from . import submodule1  
from . import subpackage
```

<https://www.sitepoint.com/python-modules-packages/>

Using the `__all__` attribute in packages (2)

This example shows an `__init__.py` file specifying that only `submodule1` and `subpackage1` will be imported when using `from my_package import *`.

Other `submodules` or `subpackages` won't be imported by default.

As in the case of `modules`, we can still import the other modules not specified in the `__all__` attribute list if we know their `absolute paths`.

So the `__all__` attribute acts as a convention rather than as a `strict rule`.

It's meant to communicate what can be used `publicly` from a `module` or a `package`.

It is, however, recommended that `explicit imports` (`import module_name`) be used instead of `wildcard imports` (`from module_name import *`).

<https://www.sitepoint.com/python-modules-packages/>

import <module_name>

```
import <module_name>
```

each **module** has
its own private symbol table,
- the global symbol table
for all **objects** defined in the **module**.

a module creates a separate namespace.

The statement **import** <module_name>
only places <module_name>
in the caller's symbol table.

The **objects** that are defined in the **module**
remain in the **module's** private symbol table.

the **import** statement does not make
the module contents directly accessible to the caller.

From the caller, **objects** in the module
are only accessible when **prefixed**
with <module_name> via **dot notation**

several comma-separated modules
may be specified in a single import statement:

```
import <module_name1>[, <module_name2> ...]
```

<https://realpython.com/lessons/import-statement/>

import <module_name>

```
>>> import mod
>>> mod
<module 'mod' from
'/Users/chris/ModulesAndPackages/mod.py'>
```

But **a**, **s**, and **printy()** remain in the module's private symbol table and are not meaningful in the local context:

```
>>> a
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    a
NameError: name 'a' is not defined
>>> s
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    s
NameError: name 's' is not defined
>>> printy
Traceback (most recent call last):
  File "<input>", line 1, in <module>
    printy
NameError: name 'printy' is not defined
```

To be accessed in the local context, names of objects defined in the module must be **prefixed** by **mod**:

```
>>> mod.a
[100, 200, 300]
>>> mod.s
'Computers are useless. They can only give you answers.'
>>> mod.printy('Hello')
arg = Hello
```

<https://realpython.com/lessons/import-statement/>

from <module_name> import <name(s)>

```
from <module_name> import <name(s)>
```

to allow individual **objects** from the **module**
to be imported directly into the caller's **symbol table**:

<name(s)> can be referenced in the caller's environment
without the <module_name> **prefix**:

because the **object** names are directly placed
into the caller's **symbol table**,

any **objects** that already exist with the same name
will be overwritten:

```
>>> from mod import s, printy
```

```
>>> s
```

```
'Computers are useless. They can only give you answers.'
```

```
>>> printy('Hello')
```

```
arg = Hello
```

```
>>> a = ['abc', 'def', 'ghi']
```

```
>>> a
```

```
['abc', 'def', 'ghi']
```

```
>>> from mod import a
```

```
>>> a
```

```
[100, 200, 300]
```

<https://realpython.com/lessons/import-statement/>

from <module_name> import <name(s)>

also possible to indiscriminately
import everything from a module in one fell swoop:

```
from <module_name> import *
```

the names of all objects from <module_name>
will be placed into the local symbol table,
with the exception of any that begin with
the underscore (_) character:

```
>>> from mod import *
>>> s
'Computers are useless. They can only give you answers.'
>>> a
[100, 200, 300]
>>> printy
<function printy at 0x03B449C0>
>>> Classy
<class 'mod.Classy'>
```

not recommended in large-scale production code.

dangerous because you're entering names
into the local symbol table en masse.

Unless you know them all well and
can be confident there won't be a conflict,
you have a danger of overwriting
an existing name inadvertently.

However, this syntax is quite handy
when you're just working with the interactive interpreter,
for testing or discovery purposes,

because it quickly gives you access to everything
a module has to offer without a lot of typing.

<https://realpython.com/lessons/import-statement/>

from <module_name> import <name> as <alt_name>

```
from <module_name> import <name> as <alt_name>
```

It's also possible to import individual objects but put them into the local symbol table with alternate names:

```
from <module_name> import <name> as <alt_name>  
[, <name> as <alt_name> ...]
```

This makes it possible to place names directly into the local symbol table but avoid conflicts with previously existing names:

```
>>> a = ['abc', 'def', 'ghi']  
>>> s = 'Hello There!'  
>>> from mod import s as string, a as alist  
>>> a  
['abc', 'def', 'ghi']  
>>> s  
'Hello There!'  
>>> string  
'Computers are useless. They can only give you answers.'  
>>> alist  
[100, 200, 300]
```

<https://realpython.com/lessons/import-statement/>

import <module_name> as <alt_name>

```
import <module_name> as <alt_name>
```

You can also import an entire **module** under an alternate **name**:

Module contents can be imported from within a **function definition**.

In that case, the import does not occur until the **function** is **called**:

However, Python 3 does not allow the indiscriminate **import *** syntax from within a **function**:

```
>>> import mod as my_module
>>> my_module.a
[100, 200, 300]
>>> my_module.s
'Computers are useless. They can only give you answers.'
```

<https://realpython.com/lessons/import-statement/>

import <module_name> as <alt_name>

Module contents can be imported from within a function definition.

In that case, the import does not occur until the function is called:

```
>>> def importer():  
...     from mod import printy  
...     printy('Hello Everyone')  
...
```

```
>>> mod
```

Traceback (most recent call last):

File "<input>", line 1, in <module>
 mod

NameError: name 'mod' is not defined

```
>>> printy
```

Traceback (most recent call last):

File "<input>", line 1, in <module>
 printy

NameError: name 'printy' is not defined

```
>>> importer()
```

arg = Hello Everyone

<https://realpython.com/lessons/import-statement/>

import <module_name> as <alt_name>

can use a **try** statement
with an **except ImportError**
to guard against unsuccessful **import** attempts:

```
>>> try:
...     # Non-existent module
...     import foo
... except ImportError:
...     print('Module not found')
...
```

Module not found

```
>>> try:
...     # Existing module, but non-existent object
...     from mod import bar
... except ImportError:
...     print('Object not found in module')
...
```

Object not found in module

<https://realpython.com/lessons/import-statement/>

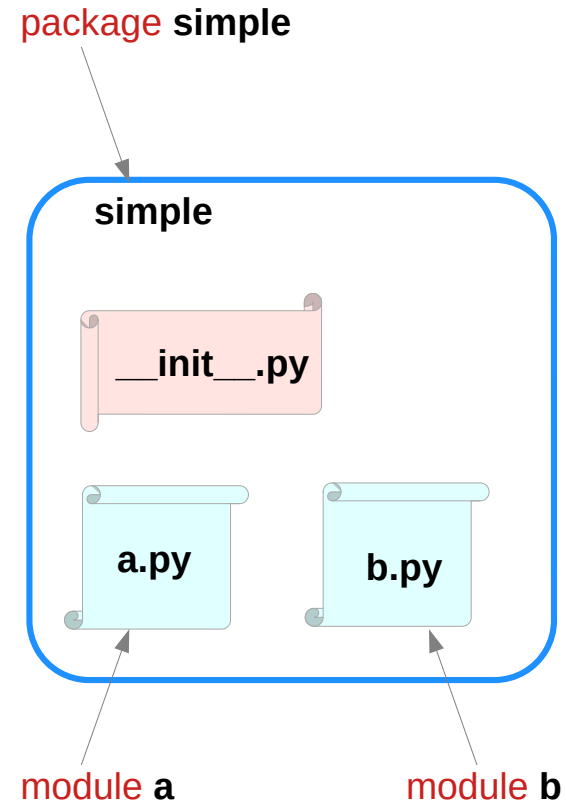
Package (1)

modules are
files containing Python statements and definitions,
like function and class definitions.

to bundle multiple modules together,
create a **package**.

a **package** is
basically a **directory**
with several **Python files (modules)**
and a special file **`__init__.py`**

inside of the **Python path**,
every **directory** contains **`__init__.py`**,
will be treated as a **package** by Python.



<https://python-course.eu/python-tutorial/packages.php>

Submodules in a package

packages are a way of structuring Python's **module namespace** by using "**dotted module names**".

A.B stands for a **submodule** named **B** in a **package** named **A**.

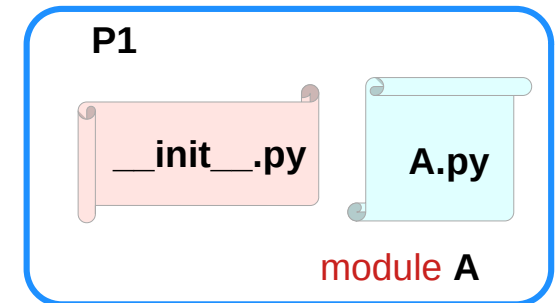
two different **packages** like **P1** and **P2** can both have **modules** with the same name, let's say **A**, for example.

The **submodule A** of the **package P1** and the **submodule A** of the **package P2** can be totally different.

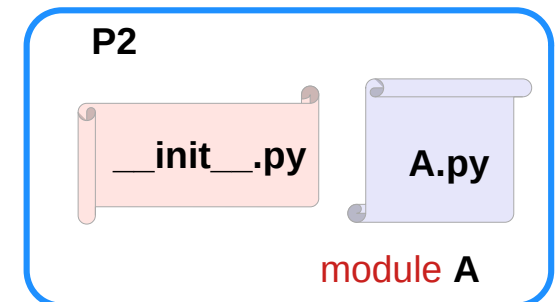
P1.A
P2.A

A **package** is imported like a "normal" **module**.

package P1



package P2



<https://python-course.eu/python-tutorial/packages.php>

Creating a package

to create a **package**, we need a **directory**.

the **name** of this **directory** will be
the **name** of the **package**,

assume we want to create "**simple**" package

must create directory "**simple**"
and this directory needs to contain the "**`__init__.py`**" file

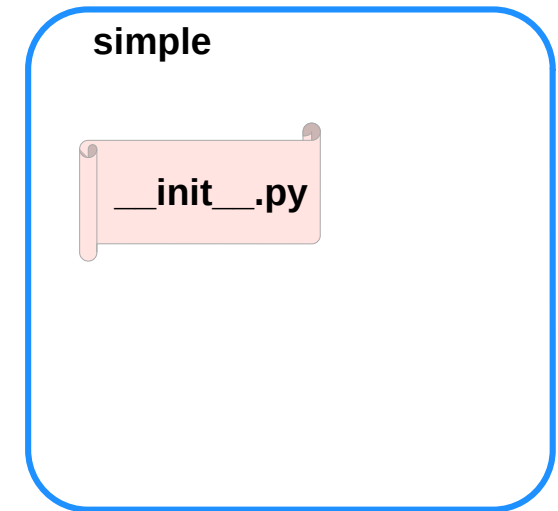
this file can be empty, or
can contain valid Python code.

this code will be executed
when a **package** is imported,

so it can be used to initialize a **package**,

e.g. to make sure that
some other modules are imported or
some values set.

package simple



<https://python-course.eu/python-tutorial/packages.php>

Examples of creating a package (1)

put all of the **Python files** which will be the **submodules** into the **directory** for a **package**.

create two simple files **a.py** and **b.py**

a.py: ➡ submodule **a**

```
def bar():  
    print("Hello, function 'bar' from module 'a' calling")
```

b.py: ➡ submodule **b**

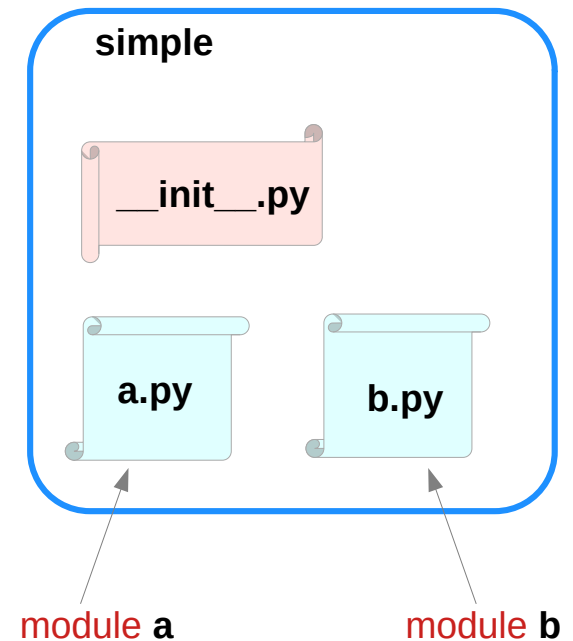
```
def foo():  
    print("Hello, function 'foo' from module 'b' calling")
```

an empty file with the name **__init__.py** inside of simple directory

__init__.py:

empty file

package **simple**



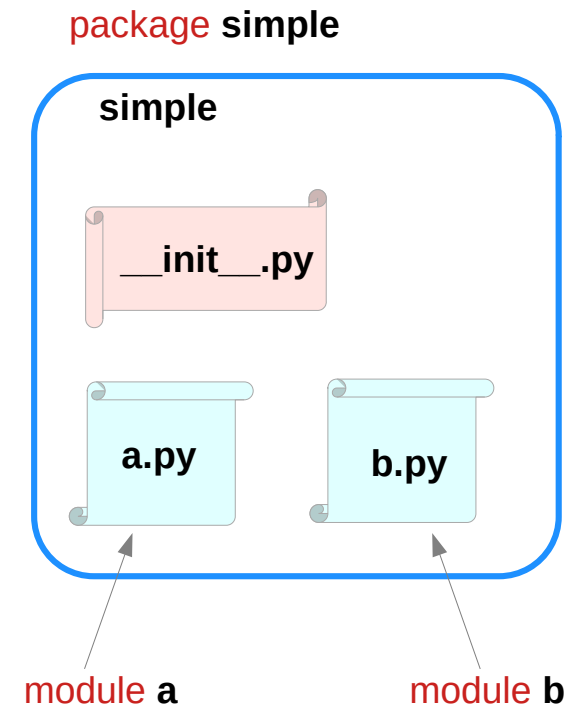
<https://python-course.eu/python-tutorial/packages.php>

Examples of creating a package (2)

`import simple` from the interactive Python shell,

assuming that the directory `simple` is

1. in the directory from which you call the shell or
2. contained in the `search path` or
3. contained `"PYTHONPATH"` environment variable



<https://python-course.eu/python-tutorial/packages.php>

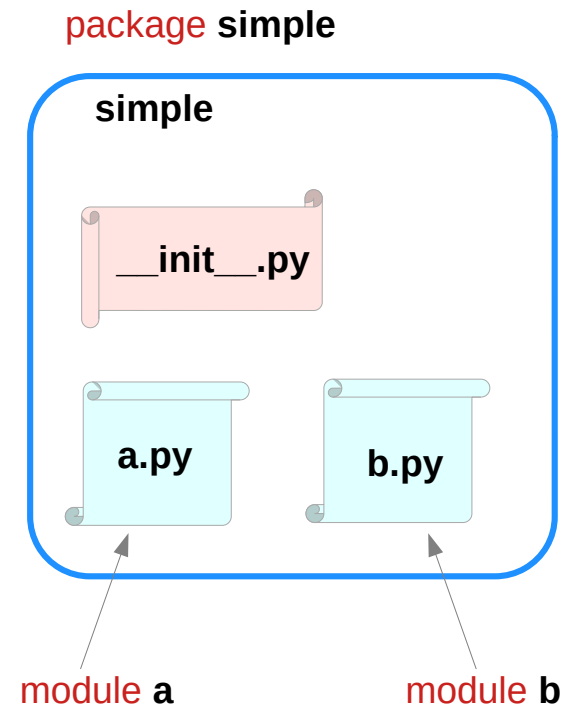
Examples of creating a package (3)

import simple ← **package simple**
simple/a ← **module a**

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-3-347df8a711cc> in <module>  
----> 1 simple/a  
NameError: name 'a' is not defined
```

simple/b ← **module b**

```
-----  
NameError                                Traceback (most recent call last)  
<ipython-input-4-e71d2904d2bd> in <module>  
----> 1 simple/b  
NameError: name 'b' is not defined
```



<https://python-course.eu/python-tutorial/packages.php>

Examples of creating a package (4)

the **package simple** has been loaded
but neither the **module "a"**
nor the **module "b"** has been loaded

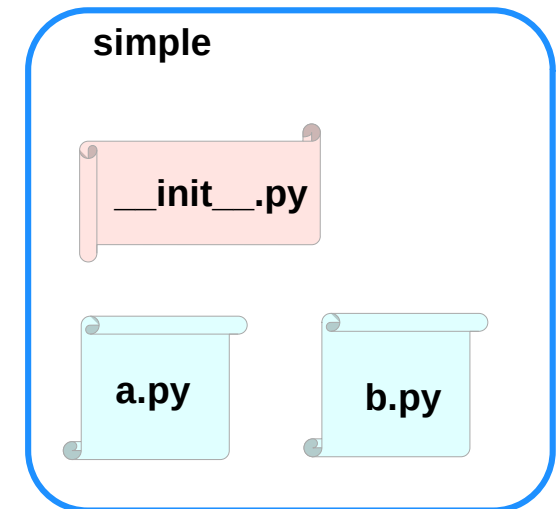
can't access neither **"a"** nor **"b"**
by solely importing simple.

must import the **modules a** and **b** as follows

```
from simple import a, b  
           package       modules
```

```
a.bar()  
b.foo()
```

Hello, function 'bar' from module 'a' calling
Hello, function 'foo' from module 'b' calling



<https://python-course.eu/python-tutorial/packages.php>

Examples of creating a package (5)

to automatically load these **modules**.

add the following lines to the file `__init__.py`:

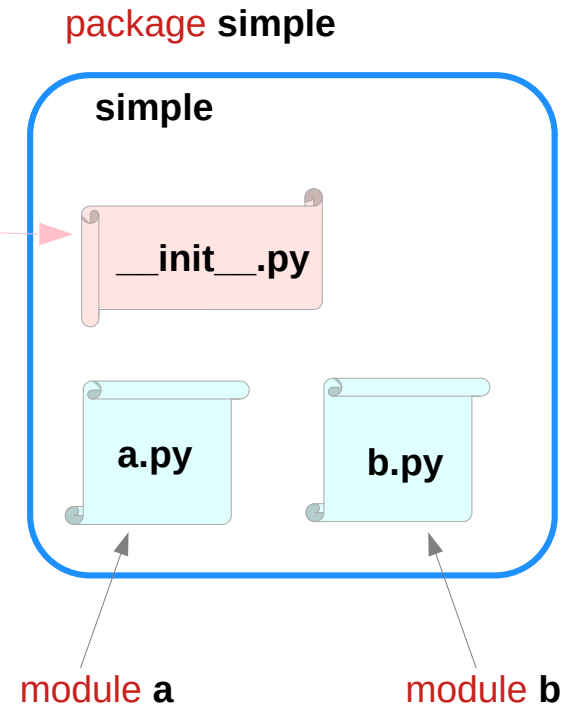
```
import simple.a  
import simple.b
```

package.module

Then

```
import simple  
simple.a.bar()  
simple.b.foo()
```

Hello, function 'bar' from module 'a' calling
Hello, function 'foo' from module 'b' calling



<https://python-course.eu/python-tutorial/packages.php>

Package Examples (1)

```
sound
|-- effects
|   |-- __init__.py
|   |-- echo.py
|   |-- reverse.py
|   \-- surround.py
|-- filters
|   |-- __init__.py
|   |-- equalizer.py
|   |-- karaoke.py
|   \-- vocoder.py
|-- formats
|   |-- __init__.py
|   |-- aiffread.py
|   |-- aiffwrite.py
|   |-- auread.py
|   |-- auwrite.py
|   |-- wavread.py
|   \-- wavwrite.py
\-- __init__.py
```

sound *package*
__init__.py *module*

effects *package*
__init__.py *module*
echo.py *module*
reverse.py *module*
surround.py *module*

filters *package*
__init__.py *module*
equalizer.py *module*
karaoke.py *module*
vocoder.py *module*

formats *package*
__init__.py *module*
aiffread.py *module*
aiffwrite.py *module*
auread.py *module*

aurwrite.py *module*
wavred.py *module*
wavwrite.py *module*

<https://python-course.eu/python-tutorial/packages.php>

Package Examples **sound1**

sound1

`__init__.py`

effects

`__init__.py`

`echo.py`
`reverse.py`
`surround.py`

formats

`__init__.py`

`aiffread.py` `aurwrite.py`
`aifwrite.py` `wavred.py`
`auread.py` `wavwrite.py`

filters

`__init__.py`

`equalizer.py`
`karaoke.py`
`vocoder.py`

```
import sound1
print(sound1) ... OK
print(sound1.effects) ... Error
```

```
import sound1.effects
print(sound1.effects) ... OK
```

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound2

sound2

__init__.py

import sound2.effects

effects

__init__.py

echo.py
reverse.py
surround.py

formats

__init__.py

aiffread.py
aifwrite.py
auread.py

aurwrite.py
wavred.py
wavwrite.py

filters

__init__.py

equalizer.py
karaoke.py
vocoder.py

```
import sound2
print(sound2) ... OK
print(sound2.effects) ... OK
```

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound3

sound3

`__init__.py`

`from . import effects`

effects

`__init__.py`

`echo.py`
`reverse.py`
`surround.py`

formats

`__init__.py`

`aiffread.py`
`aifwrite.py`
`auread.py`
`aurwrite.py`
`wavred.py`
`wavwrite.py`

filters

`__init__.py`

`equalizer.py`
`karaoke.py`
`vocoder.py`

```
import sound3
print(sound3)           ... OK
print(sound3.effects)  ... OK
```

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound4

sound4

`__init__.py`

`from . import effects`

effects

`__init__.py`

`from .. import formats`

`echo.py`
`reverse.py`
`surround.py`

formats

`__init__.py`

`aiffread.py`
`aifwrite.py`
`auread.py`

`aurwrite.py`
`wavred.py`
`wavwrite.py`

filters

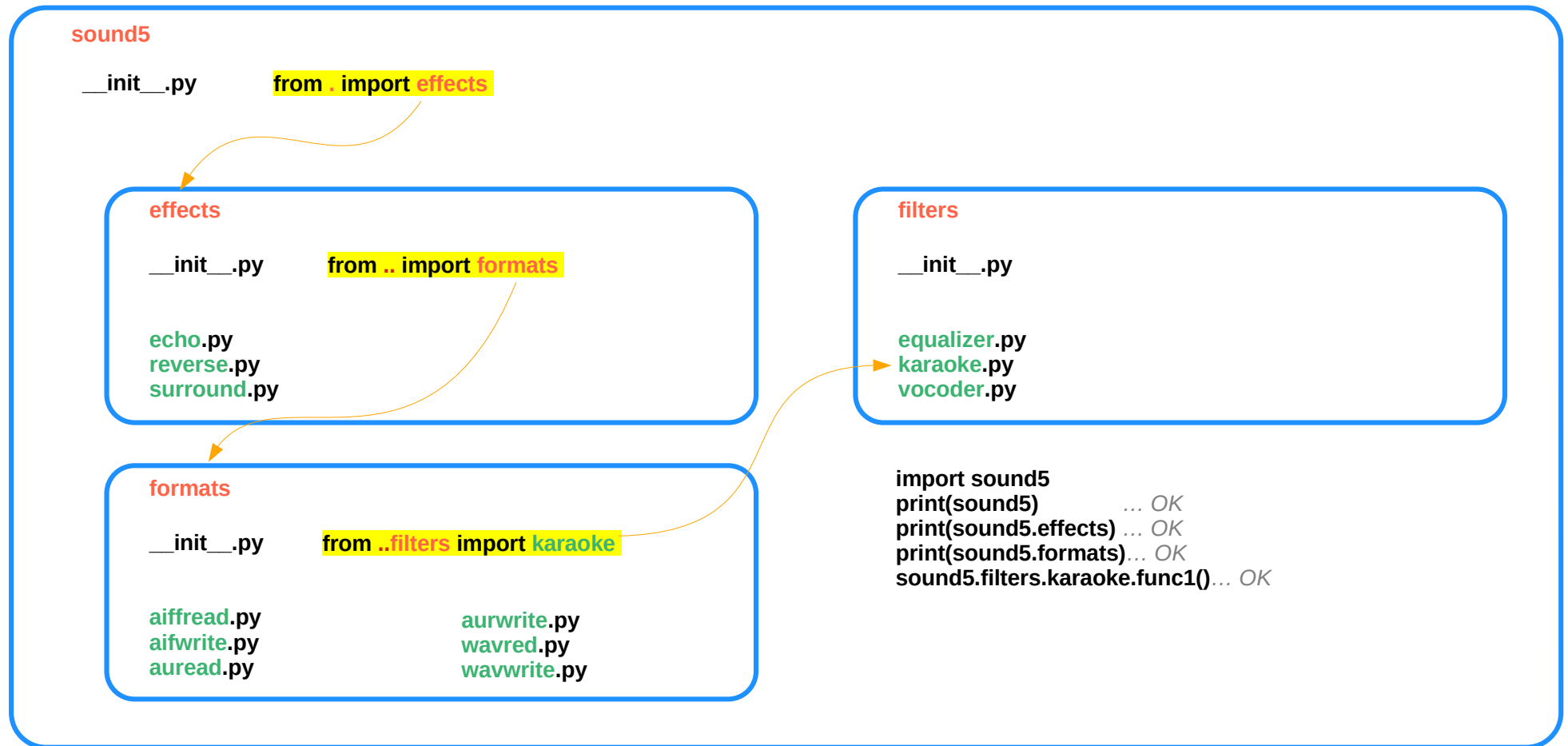
`__init__.py`

`equalizer.py`
`karaoke.py`
`vocoder.py`

```
import sound4
print(sound4)           ... OK
print(sound4.effects) ... OK
print(sound4.formats)... OK
```

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound5



<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound6

sound6

`__init__.py`

`foobar.py` *empty file*

effects

`__init__.py`

`echo.py`
`reverse.py`
`surround.py`

formats

`__init__.py`

`aiffread.py`
`aifwrite.py`
`auread.py`

`aurwrite.py`
`wavred.py`
`wavwrite.py`

filters

`__init__.py`

`equalizer.py`
`karaoke.py`
`vocoder.py`

`from sound6 import *`
sound6 package is getting imported!

`for mod in ['foobar', 'effects', 'filters', 'formats']:`
`print(mod, mod in dir())`
foobar False
effects False
filters False
formats False

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound7

sound7

`__init__.py` `__all__ = ["effects", "filters", "formats", "foobar"]`

`foobar.py` *empty file*

effects

`__init__.py`

`echo.py`
`reverse.py`
`surround.py`

*sound6 package
effects package
filters package
formats package
foobar module*

filters

`__init__.py`

`equalizer.py`
`karaoke.py`
`vocoder.py`

formats

`__init__.py`

`aiffread.py` `aurwrite.py`
`aifwrite.py` `wavred.py`
`auread.py` `wavwrite.py`

`from sound7 import *`
sound7 package is getting imported!

`for mod in ['foobar', 'effects', 'filters', 'formats']:`
`print(mod, mod in dir())`
foobar True
effects True
filters True
formats True

<https://python-course.eu/python-tutorial/packages.php>

Package Examples sound8

sound7

`__init__.py` `__all__ = ["effects", "filters", "formats", "foobar"]`

`foobar.py` *empty file*

effects

`__init__.py`

`__all__ = ["echo", "reverse", "surround"]`

`echo.py`

`reverse.py`

`surround.py`

filters

`__init__.py`

`__all__ = ["equalizer", "karaoke", "vocoder", "__init__"]`

`equalizer.py`

`karaoke.py`

`vocoder.py`

formats

`__init__.py`

`__all__ = ["aiffread", "aifwrite", "auread",
"aurwrite", "wavred", "wavwrite"]`

`aiffread.py`

`aifwrite.py`

`auread.py`

`aurwrite.py`

`wavred.py`

`wavwrite.py`

`from sound8 import *`

sound8 package is getting imported!

`from sound8.effects import *`

xxx package is getting imported!

`from sound8.filters import *`

xxx package is getting imported!

`from sound8.formats import *`

xxx package is getting imported!

<https://python-course.eu/python-tutorial/packages.php>

Package sound1 (1)

```
__init__.py
print("sound1 package is getting imported!")
```

```
effects/__init__.py
print("effects package is getting imported!")
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

If we import the package **sound1**
by using the statement **import sound1**,
Only the package **sound1** is imported
but none of the subpackages
will be imported
effects, **filters** and **formats**

because the file **__init__.py**
doesn't contain any code
for importing subpackages:

```
import sound1
print(sound1)           ... OK
print(sound1.effects)   ... Error
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound1 (2)

```
import sound1
print(sound1)
print(sound1.effects)
```

OUTPUT:

```
<module 'sound1' from '/data/Dropbox (Bodenseo)/
Bodenseo Team Folder/melisa/notebooks_en/
sound1/__init__.py'>
```

```
AttributeError                                Traceback (most recent call last)
<ipython-input-2-0b6d7fed3b24> in <module>
      3 print(sound1)
      4
----> 5 print(sound1.effects)
AttributeError: module 'sound1' has no attribute 'effects'
```

`print(sound1)`

`print(sound1.effects)`

If you also want to use the package **effects**, you have to import it explicitly with `import sound.effects`:

```
import sound1.effects
print(sound1.effects)
```

```
<module 'sound1.effects' from '/data/Dropbox (Bodenseo)/
Bodenseo Team Folder/melisa/notebooks_en/
sound1/effects/__init__.py'>
```

It is possible to have the submodule importing done automatically when importing the sound1 module.

We will change now to **sound2** to demonstrate how to do this.

We use the same files as in **sound1**, but we will add the code line `import sound2.effects` into the file `__init__.py` of the directory **sound2**.

```
print("sound2 package is getting imported!")
import sound2.effects
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound2

```
__init__.py
print("sound2 package is getting imported!")
import sound2.effects
```

```
effects/__init__.py
print("effects package is getting imported!")
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
import sound2.effects
in __init__.py of the package sound2
```

when the package `sound2` is imported,
the subpackage `effects` will also
be automatically loaded:

```
import sound2
sound2 package is getting imported!
effects package is getting imported!
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound3

```
__init__.py
print("sound3 package is getting imported!")
from . import effects
```

```
effects/__init__.py
print("effects package is getting imported!")
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

Instead of using an **absolute path** we could have imported the **effects** package **relative** to the **sound2** package.

```
import sound2.effects    # absolute path
```

```
from . import effects    # relative path
```

```
import sound3
sound3 package is getting imported!
effects package is getting imported!
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound4

```
__init__.py
print("sound4 package is getting imported!")
from . import effects
```

```
effects/__init__.py
print("effects package is getting imported!")
from .. import formats
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
import sound4
sound4 package is getting imported!
effects package is getting imported!
formats package is getting imported!
```

in the *__init__.py* file of **sound4** directory

```
from . import effects
```

in the *__init__.py* file of **effects** directory

```
from .. import formats
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound5

```
__init__.py
print("sound5 package is getting imported!")
from . import effects
```

```
effects/__init__.py
print("effects package is getting imported!")
from .. import formats
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
import sound4
sound5 package is getting imported!
effects package is getting imported!
formats package is getting imported!
filters package is getting imported!
Module karaoke.py has been loaded!
```

```
from ..filters import karaoke
into the __init__.py file of formats directory
```

can access the functions of karaoke :

```
sound5.filters.karaoke.func1()
Function func1 has been called!
```

```
formats/__init__.py
print("formats package is getting imported!")
from ..filters import karaoke
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound6 (1)

```
__init__.py
print("sound6 package is getting imported!")
```

```
foobar.py
empty file
```

```
effects/__init__.py
print("effects package is getting imported!")
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
from sound6 import *
    sound6 package is getting imported!

for mod in \
['foobar', 'effects', 'filters', 'formats']:
    print(mod, mod in dir())
    foobar False
    effects False
    filters False
    formats False
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound6 (2)

add a module (file) **foobar** (filename: **foobar.py**) to the **sound** directory.

want to import all the **submodules** and **subpackages** of the package **sound6**.

from sound6 import *

sound6 package is getting imported!

Yet, if we check with the **dir** function, we see that neither the **module foobar** nor the **subpackages effects, filters and formats** have been imported:

```
for mod in ['foobar', 'effects', 'filters', 'formats']:
    print(mod, mod in dir())
```

```
foobar False
effects False
filters False
formats False
```

```
sound7
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound7 (1)

```
__init__.py
print("sound5 package is getting imported!")
all = \
["formats", "filters", "effects", "foobar"]
```

`foobar.py`
empty file

```
effects/__init__.py
print("effects package is getting imported!")
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound7
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

```
filters/__init__.py
print("filters package is getting imported!")
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
from sound7 import * ... OK
sound7 package is getting imported!
formats package is getting imported!
filters package is getting imported!
effects package is getting imported!
foobar module is getting imported

from sound7.effects import * ... not OK
sound7 package is getting imported!
effects package is getting imported!
```

```
formats/__init__.py
print("formats package is getting imported!")
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound7 (2)

explicit index for the **subpackages** and **modules** of a **package**, which should be imported.

define a **list** named **__all__**.
to the **__init__.py** file of the **sound** directory.

: the **list** of **module** and **package** names to be imported when **from package import *** is encountered.

```
__all__ = ["formats", "filters", "effects", "foobar"]
```

```
from sound7 import *
```

sound7 package is getting imported!
formats package is getting imported!
filters package is getting imported!
effects package is getting imported!
foobar module is getting imported

```
sound7
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

check with **dir** again:

```
for mod in ['foobar', 'effects', 'filters', 'formats']:  
    print(mod, mod in dir())
```

```
foobar True  
effects True  
filters True  
formats True
```

if we use ***** in a **subpackage effects**

```
from sound7.effects import *  
sound7 package is getting imported!  
effects package is getting imported!
```

```
dir()  
['__builtins__', '__doc__', '__loader__', '__name__',  
 '__package__', '__spec__']
```

Like expected the modules inside of **effects**
have not been imported automatically.

<https://python-course.eu/python-tutorial/packages.php>

Package sound8 (1)

```
__init__.py
print("sound8 package is getting imported!")
all_ = \
["formats", "filters", "effects", "foobar"]
```

`foobar.py`
empty file

```
effects/__init__.py
print("effects package is getting imported!")
all_ = ["echo", "surround", "reverse"]
```

```
effects/echo.py
def func1():
    print("Function func1 has been called!")
    print("Module echo.py has been loaded!")
```

```
effects/reverse.py
def func1():
    print("Function func1 has been called!")
    print("Module reverse.py has been loaded!")
```

```
effects/surround.py
def func1():
    print("Function func1 has been called!")
```

```
sound8
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

```
filters/__init__.py
print("filters package is getting imported!")
all_ = ["equalizer", "__init__", \
"karaoke", "vocoder"]
```

```
filters/equalizer.py
def func1():
    print("Function func1 has been called!")
    print("Module equalizer.py has been loaded!")
```

```
filters/karaoke.py
def func1():
    print("Function func1 has been called!")
    print("Module karaoke.py has been loaded!")
```

```
filters/vocoder.py
def func1():
    print("Function func1 has been called!")
    print("Module vocoder.py has been loaded!")
```

```
from sound8 import * ... OK
from sound8.effects import * ... OK
from sound8.filters import * ... OK
from sound8.formats import * ... OK
```

```
formats/__init__.py
print("formats package is getting imported!")
all_ = ["aiffread", "aiffwrite", "auread", \
"auwrite", "wavread", "wavwrite"]
```

```
formats/aiffread.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffread.py has been loaded!")
```

```
formats/aiffwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module aiffwrite.py has been loaded!")
```

```
formats/auread.py
def func1():
    print("Function func1 has been called!")
    print("Module auread.py has been loaded!")
```

```
formats/auwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module auwrite.py has been loaded!")
```

```
formats/wavread.py
def func1():
    print("Function func1 has been called!")
    print("Module wavread.py has been loaded!")
```

```
formats/wavwrite.py
def func1():
    print("Function func1 has been called!")
    print("Module wavwrite.py has been loaded!")
```

<https://python-course.eu/python-tutorial/packages.php>

Package sound8 (2)

`__all__` list in the `__init__` file of each sub-package

effects.py

`__all__` = ["echo", "surround", "reverse"]

filter.py

`__all__` = ["equalizer", "__init__", "karaoke", "vocoder"]

formats.py

`__all__` = ["aiffread", "aiffwrite", "auread", "auwrite", "wavread", "wavwrite"]

```
sound8
|-- effects
|-- filters
|-- formats
|-- __init__.py
|-- foobar.py
```

from sound8 import *

sound8 package is getting imported!
formats package is getting imported!
filters package is getting imported!
effects package is getting imported!
foobar module is getting imported

from sound8.effects import *

Module **echo.py** has been loaded!
Module **surround.py** has been loaded!
Module **reverse.py** has been loaded!

from sound8.filters import *

Module **equalizer.py** has been loaded!
Module **karaoke.py** has been loaded!
Module **vocoder.py** has been loaded!

from sound8.formats import *

Module **aiffread.py** has been loaded!
Module **aiffwrite.py** has been loaded!
Module **auread.py** has been loaded!
Module **auwrite.py** has been loaded!
Module **wavread.py** has been loaded!
Module **wavwrite.py** has been loaded!

<https://python-course.eu/python-tutorial/packages.php>

Package sound6 (3)

Although certain modules are designed to export only names that follow certain patterns when you use **import** , it is still considered bad practice.

The recommended way is to import specific modules from a package instead of using *

<https://python-course.eu/python-tutorial/packages.php>