Common mistakes

If you accidentally import a module from itself, the code of the module will be executed twice

Another common mistake is name shadowing. For example, you have created a local module that has the same name as some built-in module. In this case, you won't be able to import anything from the original module, because the import system will search names in your custom module.

What is a package and how to get one?

Each package is a collection of typically related modules and sub-packages stored under the same directory that contains an \_\_init\_\_.py file. The purpose of the \_\_init\_\_.py file is to mark the directory as a package. Packages are extremely handy as they enhance code organization in a hierarchical manner, promote code reuse, distribution, and encapsulation.

The standard package manager for Python is called pip. It is used for installing, upgrading, and generally managing Python packages from the Python Package Index (PyPI). PyPI is the software repository for the Python programming language where all Python packages are shared, you can think of it as of a central storage place.

You can use the following command to verify whether pip is installed in your system:

pip --version # or –v

**You can use the following command to install:**

**python get-pip.py**

**or**

**python3 get-pip.py #, depending on your Python version**

**To install a package using pip use the following command:**

**pip install package\_name**

**For example, to install the numpy package you would use:**

**pip install numpy**

**Using an installed package: import**

**In order to incorporate an installed package into your program, you need to import it. Assuming we have installed the numpy package, we can use the import keyword to include it like so:**

**import numpy**

Alias can be declared using the askeyword, as shown below:

**import numpy as np**

**import long\_package\_name as short\_package**

**import long\_package\_name.long\_subpackage\_name as short\_subpackage**

**Using an installed package: from … import**

Packages can also be imported using from keyword combined with import: it allows to import specific modules or [subpackages](https://hyperskill.org/learn/step/6384" \o "In Python, a subpackage is a package contained within another package, typically organized in a hierarchical manner. | Subpackages help in enhancing code organization, promoting code reuse, distribution, and encapsulation. In the provided context, 'apple' is an example of a subpackage, which contains two modules, 'macbookair.py' and 'macbookpro.py'. Subpackages can be imported and used similarly to packages and modules. They are essential for large-scale projects, enabling better code management and version control when working with multiple packages." \t "_blank) without clogging up the [namespace](https://hyperskill.org/learn/step/6384) with unnecessary declarations. If we already know which functions or classes we'll need for our program, we can explicitly import only those, without importing any other declarations from the same module or subpackage. Let's consider our example from the previous section and import only the specific functions and classes we'll definitely use:

from numpy import array

from long\_package\_name import some\_function1

from long\_package\_name.long\_subpackage\_name import other\_function

from long\_package\_name import \*

Such imports are called [wildcard imports](https://hyperskill.org/learn/step/6384). It is really handy in terms of writing code, since you no longer need to specify the package name to which an entity belongs. However, it could make debugging quite difficult since pinpointing the specific function with the error is a hassle.

## Relative imports

[Relative imports](https://hyperskill.org/learn/step/6384) allow you to import modules or packages that are located in the same directory and depend on the current location of the module or package to be imported. They make use of the [dot notation](https://hyperskill.org/learn/step/6384): a single dot means the module or package being imported is **in the same directory** as the current directory, while two dots would mean it is **in the parent directory** of the current location.

From our laptop example, for us to import the dell.py module into the hp.py module we would use

from . import dell

since they are located in the same directory. At the same time, in order to import the dell.py file into the macbookpro.py file, we would use

from .. import dell

to print the current local time

import time

print(time.ctime())

## PEP Time!

Please note that, according to PEP 8, using **wildcard imports** is considered bad practice, as they make it unclear which names are present in the namespace, confusing both readers and automated tools. **Absolute imports are recommended**, as they are usually more readable. They also give better error messages if something goes wrong.

import package.subpackage.amateurs

from package.subpackage import amateurs

Explicit relative imports are also acceptable, especially when dealing with complex package layouts where using [absolute imports](https://hyperskill.org/learn/step/6384) would be unnecessarily verbose:

from . import animate # in amazing.py, for example

from .barriers import function # in animate.py, for example

Standard library code should avoid complex package layouts and always use absolute imports.

## Multiple packages and Virtual environment

When you check the Python Package Index (PyPI), you will notice that most packages have different versions. This is due to constant upgrades made to the packages to improve their functionality. It is therefore a good convention to make use of the [virtual environment](https://hyperskill.org/learn/step/6384) when using multiple packages. A virtual environment gives you the ability to separate packages for your different Python projects, as well as isolate them from the global Python environment. Let's say you have two different projects that use the numpy package. Project 1 uses numpy 1.24, while Project 2 uses numpy 1.19. A virtual environment makes it possible to have these two different versions of the same package **coexist** without conflicts. It acts like a barrier that isolates one version from the other.