**[Naming Conventions](https://peps.python.org/pep-0008/" \l "naming-conventions)**

The naming conventions of Python’s library are a bit of a mess, so we’ll never get this completely consistent – nevertheless, here are the currently recommended naming standards. New modules and packages (including third party frameworks) should be written to these standards, but where an existing library has a different style, internal consistency is preferred.

[Overriding Principle](https://peps.python.org/pep-0008/#overriding-principle)

Names that are visible to the user as public parts of the API should follow conventions that reflect usage rather than implementation.

[Descriptive: Naming Styles](https://peps.python.org/pep-0008/#descriptive-naming-styles)

There are a lot of different naming styles. It helps to be able to recognize what naming style is being used, independently from what they are used for.

The following naming styles are commonly distinguished:

* b (single lowercase letter)
* B (single uppercase letter)
* lowercase
* lower\_case\_with\_underscores
* UPPERCASE
* UPPER\_CASE\_WITH\_UNDERSCORES
* CapitalizedWords (or CapWords, or CamelCase – so named because of the bumpy look of its letters [[4]](https://peps.python.org/pep-0008/#id8)). This is also sometimes known as StudlyCaps.

Note: When using acronyms in CapWords, capitalize all the letters of the acronym. Thus HTTPServerError is better than HttpServerError.

* mixedCase (differs from CapitalizedWords by initial lowercase character!)
* Capitalized\_Words\_With\_Underscores (ugly!)

There’s also the style of using a short unique prefix to group related names together. This is not used much in Python, but it is mentioned for completeness. For example, the os.stat() function returns a tuple whose items traditionally have names like st\_mode, st\_size, st\_mtime and so on. (This is done to emphasize the correspondence with the fields of the POSIX system call struct, which helps programmers familiar with that.)

The X11 library uses a leading X for all its public functions. In Python, this style is generally deemed unnecessary because attribute and method names are prefixed with an object, and function names are prefixed with a module name.

In addition, the following special forms using leading or trailing underscores are recognized (these can generally be combined with any case convention):

* \_single\_leading\_underscore: weak “internal use” indicator. E.g. from M import \* does not import objects whose names start with an underscore.
* single\_trailing\_underscore\_: used by convention to avoid conflicts with Python keyword, e.g. :
* tkinter**.**Toplevel**(**master**,** class\_**=**'ClassName'**)**
* \_\_double\_leading\_underscore: when naming a class attribute, invokes name mangling (inside class FooBar, \_\_boo becomes \_FooBar\_\_boo; see below).
* \_\_double\_leading\_and\_trailing\_underscore\_\_: “magic” objects or attributes that live in user-controlled namespaces. E.g. \_\_init\_\_, \_\_import\_\_ or \_\_file\_\_. Never invent such names; only use them as documented.

[Prescriptive: Naming Conventions](https://peps.python.org/pep-0008/#prescriptive-naming-conventions)

[Names to Avoid](https://peps.python.org/pep-0008/#names-to-avoid)

Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable names.

In some fonts, these characters are indistinguishable from the numerals one and zero. When tempted to use ‘l’, use ‘L’ instead.

[ASCII Compatibility](https://peps.python.org/pep-0008/#ascii-compatibility)

Identifiers used in the standard library must be ASCII compatible as described in the [policy section](https://peps.python.org/pep-3131/#policy-specification) of [PEP 3131](https://peps.python.org/pep-3131/).

[Package and Module Names](https://peps.python.org/pep-0008/#package-and-module-names)

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.

When an extension module written in C or C++ has an accompanying Python module that provides a higher level (e.g. more object oriented) interface, the C/C++ module has a leading underscore (e.g. \_socket).

[Class Names](https://peps.python.org/pep-0008/#class-names)

Class names should normally use the CapWords convention.

The naming convention for functions may be used instead in cases where the interface is documented and used primarily as a callable.

Note that there is a separate convention for builtin names: most builtin names are single words (or two words run together), with the CapWords convention used only for exception names and builtin constants.

[Type Variable Names](https://peps.python.org/pep-0008/#type-variable-names)

Names of type variables introduced in [PEP 484](https://peps.python.org/pep-0484/) should normally use CapWords preferring short names: T, AnyStr, Num. It is recommended to add suffixes \_co or \_contra to the variables used to declare covariant or contravariant behavior correspondingly:

**from** typing **import** TypeVar

VT\_co **=** TypeVar**(**'VT\_co'**,** covariant**=True)**

KT\_contra **=** TypeVar**(**'KT\_contra'**,** contravariant**=True)**

[Exception Names](https://peps.python.org/pep-0008/#exception-names)

Because exceptions should be classes, the class naming convention applies here. However, you should use the suffix “Error” on your exception names (if the exception actually is an error).

[Global Variable Names](https://peps.python.org/pep-0008/#global-variable-names)

(Let’s hope that these variables are meant for use inside one module only.) The conventions are about the same as those for functions.

Modules that are designed for use via from M import \* should use the \_\_all\_\_ mechanism to prevent exporting globals, or use the older convention of prefixing such globals with an underscore (which you might want to do to indicate these globals are “module non-public”).

[Function and Variable Names](https://peps.python.org/pep-0008/#function-and-variable-names)

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

Variable names follow the same convention as function names.

mixedCase is allowed only in contexts where that’s already the prevailing style (e.g. threading.py), to retain backwards compatibility.

[Function and Method Arguments](https://peps.python.org/pep-0008/#function-and-method-arguments)

Always use self for the first argument to instance methods.

Always use cls for the first argument to class methods.

If a function argument’s name clashes with a reserved keyword, it is generally better to append a single trailing underscore rather than use an abbreviation or spelling corruption. Thus class\_ is better than clss. (Perhaps better is to avoid such clashes by using a synonym.)

[Method Names and Instance Variables](https://peps.python.org/pep-0008/#method-names-and-instance-variables)

Use the function naming rules: lowercase with words separated by underscores as necessary to improve readability.

Use one leading underscore only for non-public methods and instance variables.

To avoid name clashes with subclasses, use two leading underscores to invoke Python’s name mangling rules.

Python mangles these names with the class name: if class Foo has an attribute named \_\_a, it cannot be accessed by Foo.\_\_a. (An insistent user could still gain access by calling Foo.\_Foo\_\_a.) Generally, double leading underscores should be used only to avoid name conflicts with attributes in classes designed to be subclassed.

Note: there is some controversy about the use of \_\_names (see below).

[Constants](https://peps.python.org/pep-0008/#constants)

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL.