

PEP 8 – Style Guide for Python Code

Naming Conventions	
Overriding Principle	<p>Names that are visible to the user should follow conventions that reflect usage rather than implementation.</p> <ul style="list-style-type: none">• As public parts of the API.
Descriptive: Naming Styles	<p>Commonly distinguished naming styles:</p> <ul style="list-style-type: none">• b (single lowercase letter)• B (single uppercase letter)• lowercase• lower_case_with_underscores• UPPERCASE• UPPER_CASE_WITH_UNDERSCORES• CapitalizedWords (or CapWords, or CamelCase).<ul style="list-style-type: none">◦ Capitalize all the letters of the acronym.• mixedCase• Capitalized_Words_With_Underscores (ugly!) <p>Also a style of using a short unique prefix to group related names together.</p> <ul style="list-style-type: none">• E.g., os.stat() -> st_mode, st_size, st_mtime, etc. <p>The following special forms using leading or trailing underscores are recognized:</p> <ul style="list-style-type: none">• <code>_single_leading_underscore</code>: weak “internal use” indicator.<ul style="list-style-type: none">◦ E.g., from M import * -> no import on names that start with an underscore.• <code>single_trailing_underscore_</code>: used by convention to avoid conflicts with Python keywords.<ul style="list-style-type: none">◦ E.g., class_attributes• <code>__double_leading_underscore</code>: when naming a class attribute, invokes name mangling.<ul style="list-style-type: none">◦ E.g., <code>__boo</code> inside class FooBar -> <code>_FooBar__boo</code>• <code>__double_leading_and_trailing_underscore__</code>: “magic” objects or attributes that live in user-controlled namespaces.<ul style="list-style-type: none">◦ E.g., <code>__init__</code>, <code>__import__</code>, or <code>__file__</code> -> only use them as documented.
Prescriptive: Naming Conventions	<p>Names to Avoid:</p> <ul style="list-style-type: none">• Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye).<ul style="list-style-type: none">◦ In some fonts, these characters are indistinguishable from the numerals one and zero.• When tempted to use ‘l’, use ‘L’ instead. <p>ASCII Compatibility:</p> <ul style="list-style-type: none">• Identifiers used in the standard library must be ASCII

compatible as described in the policy section of PEP 3131.

Package and Module Names

- **Modules** should have short, all-lowercase names.
 - Underscores can be used in the module name if it improves readability.
- **Packages** should also have short, all-lowercase names, although the use of underscores is discouraged.
 - C/C++ modules that extend and accompany Python modules with higher level interface have a leading underscore (e.g., `_socket`).

Class Names

- Should normally use the CapWords convention.
 - Would apply the naming conventions for functions if the interface is documented and used primarily as a callable.
- Different with most **builtin names** (single words or two words run together, with the CapWords convention used only for exception names and built in constants).

Type Variable Names

- Should normally use CapWords preferring short names: T, AnyStr, Num.
- Recommended to add suffixes `_co` or `_contra` to declare covariant or contravariant behavior.

Exception Names

- Generally follow the naming conventions for classes.
- Should use the suffix "Error" on your exception names.

Global Variable Names

- Almost 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.
 - Or to indicate such globals as "module non-public".

Function and Variable Names

- **Function** names should be lowercase, with words separated by underscores.
- **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`).

Function and Method Arguments

- Always use "self" for the first argument to **instance methods**.

- Always use “cls” for the first argument to class methods.
 - Append a single trailing underscore instead of an abbreviation if a function argument’s name clashes with a reserved keyword.

Method Names and Instance Variables

- Use the function naming rules: lowercase with words separated by underscores.
 - Use one leading underscore only for non-public methods and instance variables.
 - Use two leading underscores to avoid name clashes with subclasses.
 - Python mangles these names with the class name: if class Foo has an attribute named `__a`, it cannot be accessed by `Foo.__a`.
 - Only to avoid name conflicts with attributes in classes designed to be subclassed.

Constants

- Usually defined on a module level and written in all capital letters with underscores separating words.

Designing for Inheritance

- Always decide whether a class’s methods and instance variables (collectively: “attributes”) should be public or non-public.
 - Public attributes are those that you expect unrelated clients of your class to use, with your commitment to avoid backwards incompatible changes.
 - Non-public attributes are those that are not intended to be used by third parties; you make no guarantees that non-public attributes won’t change or even be removed.
- If in doubt, choose non-public; it’s easier to make it public later than to make a public attribute non-public.
- Another category of attributes are those that are part of the “subclass API” (often called “protected” in other languages).
 - Some classes are designed to be inherited from, either to extend or modify aspects of the class’s behavior.
- Public attributes should have no leading underscores.
 - Append a single trailing underscore to your attribute name if it collides with a reserved keyword.
- Simple public attributes is best to be exposed just the attribute name, without complicated accessor/mutator methods.
- Attributes that reject the use from subclassed classes should name them with double leading underscores and no trailing underscores.
 - To invoke name mangling algorithm.

<p>Public and Internal Interfaces</p>	<p>Any backwards compatibility guarantees apply only to public interfaces.</p> <ul style="list-style-type: none">• Documented interfaces are considered public without explicit declarations.• To better support introspection, modules should declare the names in their public API using the <code>__all__</code> attribute.<ul style="list-style-type: none">◦ Setting <code>__all__</code> to an empty list indicates that the module has no public API.• Internal interfaces (packages, modules, classes, functions, attributes or other names) should still be prefixed with a single leading underscore.<ul style="list-style-type: none">◦ Also considered internal if any containing namespace (package, module or class) is considered internal.• Imported names should always be considered an implementation detail.
---------------------------------------	---