# # Good Practices

Consistency is important but knowing when to be inconsistent is important also. When in doubt, use your judgement.

## CODE LAYOUT

1) Indentation

a) Use 4 spaces per indentation level.

b) The 4-space rule is optional for continuation lines.

2) Tab or spaces

a) Spaces are the preferred indentation method

b) Tabs should be used solely to remain consistent with code that is already indented with tabs.

3) Maximum line length

a) Limit all lines to a maximum of 79 characters.

4) Line break before or after a binary operator.

a) As long as convention is consistent, we can break before or after a binary operator.

5) Blank lines

a) Surround top-level function and class definitions with two blank lines.

b) Method definitions inside a class are surrounded by a single blank line.

c) Use blank lines in functions, sparingly, to indicate logical sections.

6) Source file encoding

1. All identifiers in the Python standard library MUST use ASCII-only identifiers, and SHOULD use English words wherever feasible (in many cases, abbreviations and technical terms are used which aren’t English).
2. In addition, string literals and comments must also be in ASCII. The only exceptions are (a) test cases testing the non-ASCII features, and (b) names of authors. Authors whose names are not based on the Latin alphabet (latin-1, ISO/IEC 8859-1 character set) MUST provide a transliteration of their names in this character set.

7) Imports

a) Imports should usually be on separate lines.

b) Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

c) Imports should be grouped in the following order:

Standard library imports.

Related third party imports.

Local application/library specific imports.

8) Module level dunder names

a) Module level “dunders” (i.e. names with two leading and two trailing underscores) such as \_\_all\_\_, \_\_author\_\_, \_\_version\_\_, etc. should be placed after the module docstring but before any import statements except from \_\_future\_\_ imports. Python mandates that future-imports must appear in the module before any other code except docstrings.

String Quotes

[1:48](https://ironhack.slack.com/archives/D012YEMP6GM/p1588592902000700)

## Whitespace is Expressions and Statements

### When to use Trailing Commas

Use Double Quatations for Triple Quote Strings, otherwise use either Double or Single Quatations(Stick to one to make it easier to get used to)Avoid extraneous whitespace inside "parentheses, brackets or braces" or "between a trailing comma and a following close parenthesis" or "before a comma, semicolon, or colon" or "before the open parenthesis that starts the argument list of a function call"Trailing commas are usually optional, except they are mandatory when making a tuple of one element

Avoid extraneous whitespace inside "parentheses, brackets or braces" or "between a trailing comma and a following close parenthesis" or "before a comma, semicolon, or colon" or "before the open parenthesis that starts the argument list of a function call" or "immediately before the open parenthesis that starts an indexing or slicing" or "more than one space around an assignment (or other) operator to align it with another"

## Naming Conventions

1) Overriding Principle

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

2) 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.

3) Prescriptive: Naming Conventions

a) 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.

b) 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.

c) 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.*

d) Type Variable Names

Names of type variables introduced in PEP 484 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:

e) Exception Names

Exception name follow same naming convention as Class name

f) 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.

g) 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.

h) 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.)

i) 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.

j) 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.

4) Designing for Inheritance

Always decide whether a class's methods and instance variables (collectively: "attributes") should be public or non-public. If in doubt, choose non-public; it's easier to make it public later than to make a public attribute non-public.

We don't use the term "private" here, since no attribute is really private in Python (without a generally unnecessary amount of work).

Public attributes should have no leading underscores.

If your public attribute name collides with a reserved keyword, append a single trailing underscore to your attribute name. This is preferable to an abbreviation or corrupted spelling. (However, notwithstanding this rule, 'cls' is the preferred spelling for any variable or argument which is known to be a class, especially the first argument to a class method.)

## Public and internal interfaces

Users be able to clearly distinguish public and internal interfaces

* Public interfaces
* Compatibility issues
* Documented interface except the documentation is to be internal interface or to be provisional
* Internal interfaces
* Undocumented
* internal if the interface contains namespaces (Package,module,class)
* Annotations for module level variables, class and instance variables, and local variables should have a single space after the colon.
* There should be no space before the colon.

If an assignment has a right hand side, then the equality sign should have exactly one space on both sides:

Public and internal interfaces   
Users be able to clearly distinguish public and internal interfaces

* Public interfaces
* Compatibility issues
* Documented interface except the documentation is to be internal interface or to be provisional
* Internal interfaces
* Undocumented
* internal if the interface contains namespaces (Package,module,class)
* [Programming Recommendations](https://www.python.org/dev/peps/pep-0008/#id51)
* Code should be written in a way that does not disadvantage other implementations of Python (PyPy, Jython, IronPython, Cython, Psyco, and such).
* Comparisons to singletons like None should always be done with is or is not, never the equality operators.
* Use is not operator rather than not ... is. While both expressions are functionally identical, the former is more readable and preferred:
* When implementing ordering operations with rich comparisons, it is best to implement all six operations (\_\_eq\_\_, \_\_ne\_\_, \_\_lt\_\_, \_\_le\_\_, \_\_gt\_\_, \_\_ge\_\_) rather than relying on other code to only exercise a particular comparison.
* [Variable Annotations](https://www.python.org/dev/peps/pep-0008/#id53)
* Annotations for module level variables, class and instance variables, and local variables should have a single space after the colon.
* There should be no space before the colon.
* If an assignment has a right hand side, then the equality sign should have exactly one space on both sides