**Introduction**

This document contains the coding standards for the Train project for Group 4. The coding standards listed below will be enforced using the tool “pylint.”

**Indentation**

Use 4 spaces per indentation level.

Continuation lines should align wrapped elements either vertically using Python's implicit line joining inside parentheses, brackets and braces, or using a hanging indent. When using a hanging indent the following should be considered; there should be no arguments on the first line and further indentation should be used to clearly distinguish itself as a continuation line.

**Tabs/Spaces**

Spaces are the preferred indentation method.

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

Python 3 disallows mixing the use of tabs and spaces for indentation.

**Maximum Line Length**

Limit all lines to a maximum of 79 characters.

For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to 72 characters.

Limiting the required editor window width makes it possible to have several files open side-by-side, and works well when using code review tools that present the two versions in adjacent columns.

The default wrapping in most tools disrupts the visual structure of the code, making it more difficult to understand. The limits are chosen to avoid wrapping in editors with the window width set to 80, even if the tool places a marker glyph in the final column when wrapping lines. Some web based tools may not offer dynamic line wrapping at all.

Backslashes may still be appropriate at times. For example, long, multiple with-statements cannot use implicit continuation, so backslashes are acceptable.

**Blank Lines**

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

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

Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).

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

**Imports**

Imports should usually be on separate lines.

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

Imports should be grouped in the following order:

1. Standard library imports.
2. Related third party imports.
3. Local application/library specific imports.

You should put a blank line between each group of imports.

Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as when a directory inside a package ends up on sys.path).

Wildcard imports (from <module> import \*) should be avoided, as they make it unclear which names are present in the namespace, confusing both readers and many automated tools. There is one defensible use case for a wildcard import, which is to republish an internal interface as part of a public API (for example, overwriting a pure Python implementation of an interface with the definitions from an optional accelerator module and exactly which definitions will be overwritten isn't known in advance).

**Strings**

In Python, single-quoted strings and double-quoted strings are the same. This PEP does not make a recommendation for this. Pick a rule and stick to it. When a string contains single or double quote characters, however, use the other one to avoid backslashes in the string. It improves readability.

**Whitespace**

Avoid extraneous whitespace in the following situations:

* Immediately inside parentheses, brackets or braces.
* Between a trailing comma and a following close parenthesis.
* Immediately before a comma, semicolon, or colon.
* However, in a slice the colon acts like a binary operator, and should have equal amounts on either side (treating it as the operator with the lowest priority). In an extended slice, both colons must have the same amount of spacing applied. Exception: when a slice parameter is omitted, the space is omitted.
* Immediately before the open parenthesis that starts the argument list of a function call.
* More than one space around an assignment (or other) operator to align it with another.

**Comments**

Comments that contradict the code are worse than no comments. Always make a priority of keeping the comments up-to-date when the code changes!

Comments should be complete sentences. The first word should be capitalized, unless it is an identifier that begins with a lower case letter (never alter the case of identifiers!).

**Block Comments**

Block comments generally consist of one or more paragraphs built out of complete sentences, with each sentence ending in a period.

You should use two spaces after a sentence-ending period in multi- sentence comments, except after the final sentence.

Ensure that your comments are clear and easily understandable to other speakers of the language you are writing in.

Block comments generally apply to some (or all) code that follows them, and are indented to the same level as that code. Each line of a block comment starts with a # and a single space (unless it is indented text inside the comment).

Paragraphs inside a block comment are separated by a line containing a single #.

**Inline Comments**

Use inline comments sparingly.

An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with a # and a single space.

**Documentation Strings**

Write docstrings for all public modules, functions, classes, and methods. Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the def line.

**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.

**Function/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.

**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.