# Coding Standard

Version 1.0

## Contents

# Acknowledgement

The template of this document is taken from jcs coding standard document.

# General principles

While it is important to write software that performs well, many other issues should concern the professional developer. All *good* software performs well. But *great* software, written with style, is predictable, robust, maintainable, supportable and extensible.

## Adhere to the style of the original

When modifying existing software, your changes should follow the style of the original code. Do not introduce a new coding style in a modification, and do not attempt to rewrite the old software just to make it match the new style. The use of different styles within a single source file produces code that is more difficult to read and comprehend. Rewriting old code simply to change its style may result in the introduction of costly yet avoidable defects.

## Adhere to the Principle of Least Astonishment

The *Principle of Least Astonishment* suggests you should avoid doing things that will surprise a user of your software. This implies the means of interaction and the behavior exhibited by your software must be predictable and consistent and, if not, the documentation must clearly identify and justify any unusual patterns of use or behavior.

To minimize the chances that a user will encounter something surprising in your software, you should emphasize the following characteristics in the design, implementation and documentation of your software:

| | |
|---|---|
| Simplicity | Build simple classes and simple methods. Determine how much you need to do to meet the expectations of your users. |
| Clarity | Ensure each class, interface, method, variable and object has a clear purpose. Explain where, when, why and how to use it. |
| Completeness | Provide the minimum functionality that any reasonable user will expect to find and use. Create complete documentation: document all features and functionality. |
| Consistency | Similar entities should look and behave the same; dissimilar entities should look and behave differently. Create and apply standards whenever possible. |
| Robustness | Provide predictable documented behavior in response to errors and exceptions. Do not hide errors and do not force clients to detect errors. |

## Do it right the first time

Apply these rules to any code you write, not just the code destined for production. More often than not, some piece of prototype or experimental code will make its way into a finished product, so you should anticipate this eventuality. Even if your code never makes it into production, someone else may still have to read it. Anyone who must look at your code will appreciate your professionalism and foresight at having consistently applied these rules from the start.

## Document any deviations

No standard is perfect and no standard is universally applicable. Sometimes you will find yourself in a situation where you need to deviate from an established standard.

Before you decide to ignore a rule, you should first make sure you understand why the rule exists and what the consequences are if it is not applied. If you decide you must violate a rule, then document shy you have done so.

# Formatting conventions

1. Follow PEP8 standard for python code. Read here. To test the code against PEP8 standards click here.
2. Use single quotes, double quotes should be used only when necessary (for eg. if text itself contains single or double quote).
3. Use absolute path, relative path should be rarely used.

# Naming conventions

1. Use meaningful names.
2. Names of files (code files) created in lowercase with underscore as delimiter between words.
3. ClassName in upper camelcase (eg. CapWords, HTTPServerError).
4. Function/Method name must be lower camelcase (eg. findArea(square, side)).
5. Variables/Arguments name must be lowercase with underscore as delimiter between words (eg. customer_name).
6. Constants in capital letters with underscore delimiter (eg. PI, MAX_OVERFLOW).

# Documentation conventions

### Write documentation for those who must use your code and those who must maintain it

Document the public programming interface of your code so others can use it correctly and effectively. Document the private interface and internal implementation details of your code so others can maintain and enhance it.

Always assume someone who is completely unfamiliar with your code will eventually have to read and understand it. In fact, if enough time passes, your own code may become unfamiliar, so this person may even be you!