Python coding standard

T level students 2020/2021

Contents

[Introduction 2](#_Toc55046723)

[Clean Code 2](#_Toc55046724)

[Implications of having a bad code. 2](#_Toc55046725)

[What is a clean code? 2](#_Toc55046726)

[Smells of bad code. 3](#_Toc55046727)

[Conventions and style guide 5](#_Toc55046728)

[Indentation 5](#_Toc55046729)

[Maximum line length 5](#_Toc55046730)

[Comments 5](#_Toc55046731)

[Do not use a comment when you can use a function or a variable. 5](#_Toc55046732)

[Inline comments 6](#_Toc55046733)

[Block Comments 6](#_Toc55046734)

[Naming styles 6](#_Toc55046735)

[Names to Avoid. 7](#_Toc55046736)

[Strings quote 7](#_Toc55046737)

Leo Anderson-Leslie ………………

Mohamad Azizii ………………

Eniola Balogun ………………

Kevin Camargo ………………

Oliver Ellingham ………………

Harry Fletcher - Signed 11/01/2022

Ryan Gray ………………

Jake Grice JGrice

Dillon Mackney ………………

Joe Matongo ………………

Jai Rai - Signed 11/01/2022

James Wheeler – Signed 11/01/2022

Pat Wigley ………………

Jack Williamson JackW

Konrad Zajac KZajac

# Introduction

This document will outline the standards python coding convention that should be followed by T level students during the academic year 2020/2020.

# Clean Code

Having a clean code is critical while developing software; therefore, it is the first thing that will be introduced in this document. And it will be mentioned again and again throughout the document. Any code written by the programmers who follow this coding convention should be clean.

Most of the topics that covered in this section are available in a book called “Clean Code by Robert C. Martin”. It is a book that almost all software developers read.

## Implications of having a bad code.

A bad code will make it harder for developers to maintain the software in the future. Moreover, a bad code is more likely to contain more logical problems. Bad code also can decrease your productivity as a programmer; because you might spend a lot of time debugging and figuring out what you need to implement and where it needs to be implemented.

Yes, you might spend some time to ensure that your code is clean, but a clean code will save you a lot of time, I promise.

## What is a clean code?

Here are some very well-known and deeply experienced programmers thought.

**Bjarne Stroustrup, inventor of C++ and author of The C++ Programming Language:**

I like my code to be elegant and efficient. The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance, error handling complete according to an articulated strategy, and performance close to optimal so as not to tempt people to make the code messy with unprincipled optimizations. Clean code does one thing well.

**Grady Booch, author of Object-Oriented Analysis and Design with Applications:**

Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer’s intent but rather is full of crisp abstractions and straightforward lines of control.

**“Big” Dave Thomas, founder of OTI, the godfather of the Eclipse strategy**

Clean code can be read and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways of doing one thing. It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone.

## Smells of bad code.

Here are some smells that if you see in a code, you can tell that it’s a bad code.

* **Redundant comments**, which comments which explain something obvious. You are writing code twice and forcing developers to read the code twice. For example, this is a redundant comment:



The comment above is unnecessary and redundant.



* **Position makers comments**  
  Some programmers use comments like the one below to mark a position in the source code or to gather a group of function together. They are unnecessary and should not be used; they make the source code destructed while it needs to be one piece of code that can be read like a poem.



* **Commented-out code**commented out code decreases the readability of the code. Other programmers who work on the same program will have to read it even if it does not do anything in the program. It is ok to comment code temporarily to test and debug the program, but do not leave commented code in the source file.

# Conventions and style guide

This is a fork of PEP 8 -- Style Guide for Python Code. If anything is not mentioned here please refer to the pep document: <https://www.python.org/dev/peps/pep-0008/>

## Indentation

* Use 4 spaces per indentation level.
* 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.

In your IDE, you should change the indentation method to 4 spaces.

When invoking the Python 2 command line interpreter with the -t option, it issues warnings about code that illegally mixes tabs and spaces. When using -tt these warnings become errors. These options are highly recommended!

## Maximum line length

Limit all lines to a maximum of 79 characters.

## Comments

Comments should be used only when necessary. Try to make the code self-explanatory instead of writing a comment when possible.

### Do not use a comment when you can use a function or a variable.

Consider this code:



Would not be better if we replace the comment with a function:



### Inline comments

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.

Inline comments are unnecessary and in fact, distracting if they state the obvious. Do not do this:



### Block Comments

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 #.

## Naming styles

There are different styles for writing names for variables, functions, and variables. You should use these each time you name something.

* Camel case: “myCar” (use it for functions names)
* Snake case: “my\_car” (use it for variables names)
* Pascal case: “MyCar” (use it for classes names)

If a variable is going to store a boolean it should be started with “is\_”. For example: “is\_adult”.   
Note: Follow the naming conventions of the files you work on when you work on old files. Please keep the naming consistent.

## 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.

## Strings quote

A string should be contained with double quotes (“). A substring (a string inside a string) should be contained in a single quote (‘),

Matthew

* Indentations
* Maximum Line Length (79 Characters per line)
* Blank Lines (Separate Functions and Classes)
* Avoid Whitespaces in Expressions and Statements
* Source File Encoding
* Importing libraries and/or modules
* Comments (Understand what you did when writing)
* Naming Conventions (a set of rules for choosing the character sequence that should be used for identifiers which denote variables, types, functions, and other entities in source code and documentation.)

*James*

* Indentations (4 spaces per indentation level)

The red box is the indentation.



* Blank Lines (Separation)



* Source File Encoding
* 79 Characters per line max
* Importing libraries and/or modules



* Avoid Whitespaces in Expressions and Statements
* Comments (use #) (Understand what you did when writing)



* Naming Conventions

# Signatures

James J Wheeler

Ryan Gray