CSI2100-01 Spring 2020

Python Style Guide

Contents

1 Introduction

This style guide is intended to help you to adopt to basic coding conventions. Properly styled source code is more easily read and understood by humans.

In your career as a software engineer, you will work on code with other people. It is important that your code is easily understood by yourself and others. Software companies will employ similar style guidelines, and it is usually mandatory to adhere to those.

This guide is a simplified version of the *Style Guide for Python Code* by Guido van Rossum (the Python inventor) and Barry Warsaw. Their original document has become the well-known Python enhancement proposal PEP 8.

The rules that we employ with the CSI2100-01 course are a subset of PEP 8, which we consider sufficiently general for you to pick up. We have deliberately left out specifics or "nit picking" rules. We hope that the chosen selection will allow you to experience the benefit of style guidelines and that you will experience an improvement in your code quality.

To aid developers to adhere to a style guide, an automated tool is employed to analyze the code. Most coding conventions are easy to check, for example, such a tool can count blanks to determine whether code is indented in multiples of four. Nevertheless, code analysis has fundamental constraints. For example, it is impossible for a tool to determine whether the content of a docstring contradicts its accompanying source code (to do so, the tool would have to solve the Halting Problem, which is

known to be undecideable[1]).

In the CSI2100-01 course, we will employ the open-source flake8 tool with Hyeongjae Python. We configure flake8 to check only the PEP 8 subset of the CSI2100-01 course.

By uploading your code on Hyeongjae Python, a style check will be conducted and the result will be provided to you. Please fix all style issues in your code. The explanations in the following sections will help you to understand how to fix the code.

You can find the full list of CSI2100 checks (and error messages) with examples and explanations here. By clicking on an error code (e.g., E111), you will be directed to the page with an example error, its fix, and a general description of the error. Please also refer to the following section for a general description of the style guidelines.

Note 1: it is mandatory to apply the rules explained in this style guide with the lab problems that you will submit on YSCEC. Failing to adhere to the style guide will result in points deductions, as outlined in the lab specifications.

Note 2: please kindly understand that the source code provided to you on lecture slides cannot adhere 100% to this style guide, for reasons of space constraints. **However**, starting from Lab 12, the source code provided to you on YSCEC has undergone the CSI2100-01 style checks of this guide.

If you have questions regarding the style guide and how to adhere to it, please ask on our CSI2100-01 YSCEC Q&A board.

2 Guidelines and Flake8 Error Messages

2.1 Indentation

Always use four spaces per indentation level. Never use tabs for indentation, use spaces only.

Limit all lines to a maximum of 79 characters. Limiting windows to 80 characters makes it possible to have several windows side-by-side. Long lines disrupt the visual structure of the code in a 80-character wide window, which makes it more difficult to understand.

The preferred way of wrapping long lines is by using Python's implied line continuation inside parentheses, brackets and braces. If necessary, you can use a backslash to continue to the next line.

These rules are covered by flake8 errors E101 – E117, E501, and W191 (see here).

2.2 Blank Lines

Separate top-level function and class definitions with two blank lines. (Top-level means that those functions or classes are not nested inside another function or class.)

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

Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between related, small (one-liner) functions.

[1] You will study this matter in detail in the Automata course.

Use blank lines in function bodies sparingly, to indicate logical sections of code.

These rules are covered by flake8 errors E301, E302, E303, E305, and E306 (see here).

2.3 Whitespace in Expressions and Statements

2.3.1 Use spaces around binary operators

Yes:

```
i = i + 1
submitted += 1
x = x * 2 - 1
c = (a + b) * (a - b)
```

No:

```
i=i+1
submitted+=1
x=x*2-1
c=(a+b)*(a-b)
```

2.3.2 Avoid Whitespace in the Following Situations

- Immediately inside parentheses, brackets or braces.

  Yes: `spam(ham[1], {eggs: 2})`
  No: `spam( ham[ 1 ], { eggs: 2 } )`

- Immediately before a comma, semicolon, or colon:

  Yes: `if x == 4: print x, y; x, y = y, x`
  No: `if x == 4 : print x , y ; x , y = y , x`

- Immediately before an open parenthesis:

  Yes: `spam(1)`
  No: `spam (1)`

- Immediately before the open bracket that starts an indexing or slicing operation:

  Yes: `dict['key'] = list[index]`
  No: `dict ['key'] = list [index]`

- More than one space around an assignment (or other) operator to align it with another.

  Yes:

```
        x = 1
        y = 2
        long_name = 3
```

  No:

```
x         = 1
y         = 2
long_name = 3
```

- Do not use space around = with function default parameters:

  Yes:

```
def func(key1='val1',
         key2='val2'):
    return key1, key2
```

  No:

```
def func(key1 = 'val1',
         key2 = 'val2'):
    return key1, key2
```

  These rules are covered by flake8 errors E201 – E251 (see here).

2.4 Comments

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

In-line comments must be separated from the preceeding code by two blanks.

After the hash (#), one blank must follow, followed by the comment itself.

Please refer to the examples with the explanations of the flake8 error messages for errors E261 – E266 provided to you here.