Style Guide Questions

1. **What is the purpose of a programming language style guide like PEP 08?**

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself. Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

1. **Provide some examples of when it would be ok to be inconsistent with the PEP 08 style guide.**

* When applying the guideline would make the code less readable, even for someone who is used to reading code that follows this PEP.
* To be consistent with surrounding code that also breaks it (maybe for historic reasons) -- although this is also an opportunity to clean up someone else's mess (in true XP style).
* Because the code in question predates the introduction of the guideline and there is no other reason to be modifying that code.
* When the code needs to remain compatible with older versions of Python that don't support the feature recommended by the style guide.

1. **Summarize proper indentation style. Explain some examples of good and bad indentation style.**

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.

**Example of a good indentation:**

1. # Aligned with opening delimiter.
2. foo = long\_function\_name(var\_one, var\_two,
3. var\_three, var\_four)
4. # More indentation included to distinguish this from the rest.
5. def long\_function\_name(
6. var\_one, var\_two, var\_three,
7. var\_four):
8. print(var\_one)
9. # Hanging indents should add a level.
10. foo = long\_function\_name(
11. var\_one, var\_two,
12. var\_three, var\_four)

**Example of a bad indention:**

1. # Arguments on first line forbidden when not using vertical alignment.
2. foo = long\_function\_name(var\_one, var\_two,
3. var\_three, var\_four)
4. # Further indentation required as indentation is not distinguishable.
5. def long\_function\_name(
6. var\_one, var\_two, var\_three,
7. var\_four):
8. print(var\_one)
9. The 4-space rule is optional for continuation lines.
10. **What should be used for whitespace, tabs or 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.
* Python 2 code indented with a mixture of tabs and spaces should be converted to using spaces exclusively.
* 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!

1. **Why does PEP 08 recommend limiting line length to a maximum of 79 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.

1. **What does PEP 08 have to say about ‘single string’ quotes and “double string” quotes? How would you quote a string with an apostrophe like don’t?**

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.

1. **Summarize proper use of “whitespace”. Explain some examples of good and bad whitespace style.**

Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker. Some editors don't preserve it and many projects (like CPython itself) have pre-commit hooks that reject it

* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -=etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).
* If operators with different priorities are used, consider adding whitespace around the operators with the lowest priority(ies). Use your own judgment; however, never use more than one space, and always have the same amount of whitespace on both sides of a binary operator.

1. **What is the recommended use of a “block comment”. Provide an example of a block comment and explain how a block comment is different from a “inline comment”**

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

1. **Summarize proper use of “inline comments”. Explain some examples of good and bad inline comment style.**
2. **Which naming styles have you seen used on example code provided in class so far?**
3. b (single lowercase letter)
4. B (single uppercase letter)
5. lowercase
6. lower\_case\_with\_underscores
7. UPPERCASE
8. UPPER\_CASE\_WITH\_UNDERSCORES
9. CapitalizedWords (or CapWords, or CamelCase -- so named because of the bumpy look of its letters [[4]](https://www.python.org/dev/peps/pep-0008/#id11)). This is also sometimes known as StudlyCaps.
10. mixedCase (differs from CapitalizedWords by initial lowercase character!)
11. Capitalized\_Words\_With\_Underscores (ugly!)
12. **What is the recommended naming style for types and variables? Provide an example of a variable definition using this style.**

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. Examples:

1. from typing import TypeVar
2. VT\_co = TypeVar('VT\_co', covariant=True)
3. KT\_contra = TypeVar('KT\_contra', contravariant=True
4. **What is the recommended naming style for function names? Provide an example of a function definition using this style.**

Function names should be lowercase, with words separated by underscores as necessary to improve readability.

mixedCase is allowed only in contexts where that's already the prevailing style (e.g. threading.py), to retain backwards compatibility.

1. **What is the recommended naming style for constants? Provide an example of a constant definition using this style.**

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include MAX\_OVERFLOW and TOTAL