**Python Style Guide**

This style guide is to be used as a reference for all work being done under the Amazon Team related to the AlarmBuddy project. The attached guides were heavily inspired by style guidelines provided by Google under the CC-By 3.0 License, which encourages the sharing of these and the original source documents. The original style guides provided by Google can be found at <https://google.github.io/styleguide/>. The python style guide PEP 8 provided at <https://www.python.org/dev/peps/pep-0008/> also contributed towards the creation of this style guide. Copyright for this document has been place in public domain, reference Copyright in link.

**Introduction**

This is the documentation for Amazon Team’s style guide for the AlarmBuddy project.

The goal of this style guide is to give a documented structure for organizing and formatting code for Amazon Team.

**Readability**

A key feature to not is that while following this style guide is important, the most important thing for readable code is consistency. Consistency will allow a reader to pick up structures and patterns to allow more readable code. Consistency is the most important inside a module/function.

Yet, consistency is not always important and some places, this style guide might not fit. PEP 8 provides some reasons to ignore a styleguide.

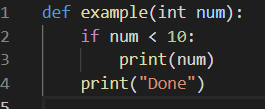
1. When applying the guideline would make the code less readable, even for someone who is used to reading code that follows this PEP.
2. To be consistent with surrounding code that also breaks it (maybe for historic reasons) -- although this is also an opportunity to clean up someone else's mess (in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

-Provided by PEP Style Guide

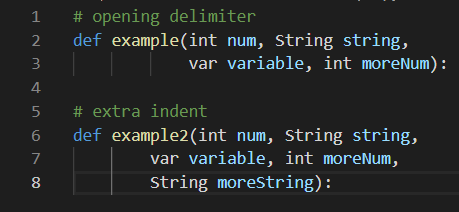
As a general rule, comment is the section can be confusing and consult group members if questions arise.

**Layout of Code**

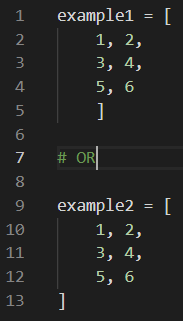
Indentation level should be represented by 4 spaces. (ie 1 indent = 4 spaces) This rule is key as python does not use curly brackets for defining code blocks.



Long, continuous lines should be wrapped based on the opening delimiter or with an extra indentation for clarity and separation.

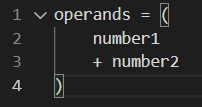


Variables must be wrapped as well. If a variable’s contents must be wrapped, the first of its contents must be a line under the opening brace/bracket/parenthesis and indented once. The closing of brackets, braces or parenthesis for wrapped declares for variable must either line up under the first whitespace character of the former line or can be in line with the variable name.



**Line Break with Operands**

Operands should be a the start of a line break, not the end.



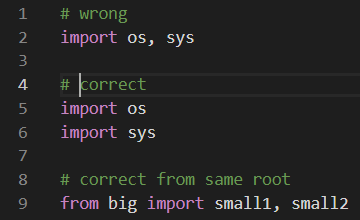
**Maximum Line Length**

The general rules for AlarmBuddy’s style guides states that all lines should be kept under 120 characters long. For python, this will be limited to 79 characters.

This refers to each individual line literally, not the whole statements. If a statement exceeds 79 characters, it should be wrapped so each wrapped line only contains 79 characters. This will allow for statements above 79 characters and methods with multiple variables.

**Imports**

Separate imports should be on separate lines unless both imports are from the same root folder.



Imports should always be at the top of the file. Only module comments and docstrings should precede imports.

Import order should be Standard library, Related third party imports and then local or library specific imports. Each group should be separated by a empty line.

Absolute imports are recommended but explicit relative imports are accepted.

**Strings**

A single quotes and double quotes are the same in python.

**Whitespace**

Whitespaces should not exist

* Immediately in braces, brackets, or parentheses
* Between a trailing comma and close parentheses
* Immediately before a comma or semicolon
* Immediately before the start of an argument or function call’s open parentheses
* Immediately before an open bracket of indexing or slicing
* More than one space around an assignment or operator
* After lines with no purpose

Add white spaces around binary operators (==, <, >, etc)

**Naming Styles**

Single lowercase: a

Single uppercase: A

lowercase

lowercase\_with\_underscores

UPPERCASE

UPPERCASE\_WITH\_UNDERSCORES

CamelCase

mixedCase (Only first word is completely lowercase)

Capitalized\_Words\_With\_Underscores

Avoid lowercase L (l), uppercase o (O) and uppercase I (I).

Make sure identifiers are compatible with ASCII

**Naming Rules**

Package and Module: short, all lowercase names. Lowercase\_with\_underscores can improve readability

Class Names: CamelCase

Type Variable Names: CamelCase

Exception Names: ClassNames with “Error” at the end

Function and Variable Names: mixedCase

Global Variable Names: lowercase\_with\_underscore

Method Names: lowercase\_with\_underscore

Constants: UPPERCASE\_WITH\_UNDERSCORES

With python ever changing, due to backwards compatibility, these rules cannot always be followed. In general, try to follow these naming rules as closely as possible but if needed, adjustments can be made.

**Function and Variable Annotations**

Refer to PEP 8 Page:

Function: <https://www.python.org/dev/peps/pep-0008/#function-annotations>

Variable: <https://www.python.org/dev/peps/pep-0008/#variable-annotations>

**Miscellaneous Rules and Recommendations**

Code should not disadvantage other python libraries and implementations.

Comparisons to None should be is or is not.

Use specific exceptions for except clause in a try except method.

Limit try clause to bare amount of code as possible

Be consistent with return statements in a function. If returning a certain value based on result, return None in an area where no value is returned.

For True or False if statements, do not set equal to True or False

License:

Copyright © 2021 Students of University of St. Thomas CISC 480-D01, Spring 2021

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE  LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

End of license.

The license that is being used for AlarmBuddy is the MIT license. The licensing sub team decided that MIT was the best fit for AlarmBuddy and covered all foreseeable issues relating the development of the project. The MIT license is a permissive free software license that puts very limited restrictions on reuse giving high third-party license compatibility.

<https://opensource.org/licenses/MIT>