On the Effect of Python Style Guide's Line Length Limit: Does Code Become Less Readable?

Kevin Oliva Muñoz
Universidad Rey Juan Carlos
Fuenlabrada, Madrid, Spain
kivenoliva@gmail.com

Gregorio Robles
Universidad Rey Juan Carlos
Fuenlabrada, Madrid, Spain
grex@gsyc.urjc.es

Work in Progress

Abstract

Programming languages usually have a style guide that indicates, among other things, how to format the code. Although most of these guidelines are widely accepted conventions, some rules are heavily debated. This is the case for Python's maximum line length limit rule, that limits lines of code to a maximum of 79 characters. According to some opinions, following this rule often results in code that is more difficult to understand, because developers may shorten the variable names to fit the line length limitation up to a point to making these variable names unintelligible. Our goal is to detect and quantify how many lines have suffered this malpractice. Therefore, we download several git repositories of projects written in Python, and scan the code for variable name changes in lines that before the change were over 79 characters long, but not after the change. Our results show that the number of occurrences that we have found following this method is very low (0.01%). We offer a manual analysis of those lines which provides further insight.

1 Introduction/Motivation

On top of the syntactic rules of programming languages, developer communities often agree on using a set of common rules for writing code, that ranges from variable names to how spaces are to be used. Style guides are something that is generalized in different programming languages. Sometimes more than one style guide exists for a programming language; it is even frequent for free software projects have their own (e.g., Linux [2]). However, there are some style guides that are more popular than the others. For example, for PHP we have PSR-2 [1], and in JavaScript one of the most used ones is Crockford's[1]. Something very common is to consult the guides that Google has created, e.g., for C++ [3] or for Java[2].

The style guide in use in the Python Standard Library has been widely adopted by many other Python projects, and can be found as a Python Enhancement Proposal (that is why it is known as PEP8[3]). PEP8 was authored by Guido van Rossum, Barry Warsaw and Nick Coghlan, and had to follow the usual acceptance procedure as all other PEPs for the Python language.

This guide is made up of several conventions, in which we will not go into depth, but that includes code layout, comments, naming conventions, and programming recommendations, among others. All these recommendations, some of them just conventions to have a similar format, have been widely accepted by the Python community. Nonetheless, there is one rule that has always provoked a great debate: the maximum line length. The rationale for this rule is as follows:

> "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.

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

[1] http://crockford.com/javascript/style1.html
[2] https://google.github.io/styleguide/javaguide.html
[3] `https://www.python.org/dev/peps/pep-0008/`

> The default wrapping in most tools disrupts the visual structure of the code, making it more difficult to understand. The limits are chosen to avoid wrapping in editors with the window width set to 80, even if the tool places a marker glyph in the final column when wrapping lines. Some web based tools may not offer dynamic line wrapping at all."

Many voices argue that in order to comply with the rule, developers shorten the variable names[4]. To try to illustrate this situation, let us consider a small example. Let's imagine that the developer has written the following line of code (note that we divide the line in two in order to make it fit in one column, but it should be seen as a single line):

```
'subtitles': self.extract_subtitles(video_id,
                                    video_subtitles_id)
```

If we are analyzing code and read this line, in principle, it can be understood well without the necessity of having more context or documentation, even without any knowledge of the Python language. We can easily assume that what we are getting in the variable "subtitles" are the subtitles of a video, which we extract by calling the function "extract-subtitles" and passing as parameters the id of the video and the id of the subtitles. There is no need of comments or documentation in order to understand it.

The line of our example itself is 69 characters long. However, Python is a language where *indentation* has semantic meaning[5]. And in its context, including several indentation levels[6], this line exceeds 79 characters.

The developer will then have to refactor the code to comply with the maximum line limit rule. There are many possibilities for doing this, but one possibility (probably the easiest, or the fastest one) is to shorten the name of variables and functions/methods. The resulting line, which is under 80 characters (including indentation), could look like this:

```
'subtitles': self.e_s(v_id, v_s_id)
```

The result is a line that is PEP8 compliant, but harder to understand. For the sake of following the style guide, we land in a situation that is worse than before, as readability of code is a major goal of the style guide. This is stated in themotivation of PEP8:

> "One of Guido's key insights is that code is read much more often than it is written.
>
> The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code. As PEP 20 says, "Readability counts"."

[4]See Raymond Hettinger's "Beyond PEP8" talk at PyCon 2015 in Montreal, Canada: https://www.youtube.com/watch?v=wf-BqAjZb8M

[5]This term *indentation* means moving a block of text to the right by inserting spaces or tabs, in order to separate it from the left margin and better distinguish it from the adjacent text. It is used to improve the readability of the source code by programmers.

[6]PEP8's recommendation for indentation is "Use 4 spaces per indentation level."

2 Methodology

Python files are retrieved from a git repository and analyzed line by line. Depending on the length of a line, a line is categorized into three groups:

- "Green" line: the line less than 70 characters long (including indentation spaces).
- "Orange" line: the line has between 70 and 79 characters in length (including indentation spaces).
- "Red" line: the line has 80 or more characters in length (including indentation spaces). These lines do not comply with the PEP8 style guide.

Our hypothesis is that developers shorten variable names to comply with the line length limitation in PEP8, resulting in an *orange* line in the newer version, when in a previous version the line was *red*. The rationale for identifying changes in this way is that we hypothesize that developers who shorten variable names to comply with PEP8 will not drastically change the modified lines. Hence, the resulting line will be close to the limit (at least 70 characters).

For all lines that are *orange*, we collect following information:

- **line:** Contents of the line.
- **file:** Path of the file within the git repository.
- **numLine:** Number of the line within the file.
- **lengthLine:** Length of the line.
- **pastRed:** (*boolean*) This parameter indicates if the line was at some point in the past a *red* line.
- **history:** (*list*) This parameter stores the different states that the line has had throughout its history.

We call the lines that are currently *orange* and were *red* at some point in the past, the lines of interest. We search for the cases where the two versions of the line differ in a single word. These lines are called the lines under study. We also consider the case when the lines differ in a single word and indentation has been modified.

3 Results

We have randomly downloaded 9 repositories from GitHub that contain at least 10,000 lines of Python code, in order to apply the above described methodology on them.

Summing all sizes, the total number of lines that have been analyzed is 369,807. From these, we have found:

- **Number of green lines:** 330,805 (89.5%)
- **Number of orange lines:** 20,101 (5.4%)
- **Number of red lines:** 18,901 (5.1%)

As expected, in all repositories, the vast majority of lines are green. However, none of the repositories are completely PEP8 compliant, as they all have lines that are 80 characters or more. Five out of the nine repositories have more *red* lines than *orange* lines.

We have investigated how many of those red lines correspond to comments (both *in-line* comments as well as multi-line comments). Column "Red (comments)" in Table 1 offers our results. In total, 99 out of the 20,101 *red* lines are comments, which is 5.3%.

The number of *lines of interest* that we have found is 396, which supposes 0.10% of the total lines analyzed and 1.96% of the total lines in orange area (20,107). It can be seen that the percentages that are handled at this point are very low.

Regarding the number of *lines under study*, we have found 125 lines with a change of length in a single word and 3 more lines where in addition to the length change indentation has also been changed, totaling 128 *lines under study*. This makes 0.03% of the total lines analyzed and 0.6% of the total lines in orange area (20,107).

3.1 Manual Analysis

At this point, we have found 128 lines that were *red* in the past and that are now *orange*. We have seen that the change corresponds to modifying a word. However, we do not have evidence that these lines have undergone the change that we have hypothesized. That is why we decided to manually review those lines.

From the 128 cases that we have considered as *lines under study*, we found 44 cases (34%) where the change is to shorten a variable name. Table 2 offers more details on this process. The old line and the new line were compared and evaluated for the change performed. The change was considered a *positive* when it was due to a variable name shortening. From the ten cases shown in Table 2, this happens in three lines (#2, #8 and #9). For the other seven cases, we assume that the change does not affect the readability of the code, although in some cases (such as for lines #1 and #6) we had our doubts.

4 Discussion, threats to validity and further research

We have generally obtained negative results regarding the case sought. We have only found 396 (0.10%) of lines that have been trimmed from over 80 characters to less, although only 128 (0.03%) match the condition that only one word has been modified. Out of these, we manually analyzed if a variable name has been modified, making the code more difficult to understand. Our results show that 44 out of 128 lines are affected (0.01%). In other words, out of 10,000 lines of Python code we have found that there is one single line where a variable name has been shortened to comply with the PEP8 guideline. In this sense, our results do not offer evidence that PEP8 affects what many of the criticism it has raised.

However, **our results have to be taken with care** as some threats to its validity exist. Our impression is that with our methodology we have not captured the effect, because of several reasons.

A major threat to our methodology is that we may not capture the process by which a developer shortens variable names to comply with the PEP8 maximum line length limitation rule. Developers may well check the compliance of their code before committing it to the repository, a task that can be easily performed with some of the code quality assurance tools that exist for Python development such as `pycodestyle`[7] (formerly known as `pep8`), `pylint`[8] or `pyflakes`[9]. In such cases, our method will not be able to identify such cases. However, it should be noted that from the amount of *red* lines in the repositories that have been analyzed, it seems that this is not the case.

Another threat to our methodology is that we only look for those changes where only one variable name may have been changed. It could be well possible that many variable names could have been modified in order to comply with PEP8. However we have seen that the effect of this simplification is small, as we identified only 396 (0.1%) potential candidates.

The manual analysis is partially subjective and can only provide partial evidence. The analysis is done without further context (i.e., we only have observed the affected line, but not those lines before and after). As in the previous case, the effect of this subjectivity, given the amount of lines we have found, is also small.

[7] `https://github.com/pycqa/pycodestyle`
[8] https://www.pylint.org/
[9] https://github.com/PyCQA/pyflakes

Project	Green	Orange	Red	Red (comments)	Total
youtube-dl	112,898	6,332	8,507	5	127,737
SerpentAI	3,396	130	282	0	3,808
requests	29,164	1,475	3,235	79	33,874
scrapy	31,174	2,400	1,709	99	35,283
pythondotorg	10,609	380	601	3	11,590
models	56,246	2,949	1,058	683	60,253
lbry	24,832	1,700	1,885	59	28,417
keras	49,456	3,428	1,525	60	54,409
flask	13,030	1,307	99	11	14,436

Table 1: Analyzed projects and results.

#1	new	_VALID_URL = r'https?://(?:www)?alphaċom/videos/(?P< *id* >[^/]+)'
No	old	_VALID_URL = r'https?://(?:www)?alphaċom/videos/(?P< *display_id* >[^/]+)'
#2	new	return self.playlist_result(entries, video_id, **title**, description)
Yes	old	return self.playlist_result(entries, video_id, **playlist_title**, description)
#3	new	'**skip**': redirect to http://swrmediathek.de/index.htm?hinweis=swrlink',
No	old	'**_skip**': 'redirect to http://swrmediathek.de/index.htm?hinweis=swrlink',
#4	new	except (TimeoutError, HTTPException, SocketError, **ProtocolError**) as e:
No	old	except (TimeoutError, HTTPException, SocketError, **ConnectionError**) as e:
#5	new	elif status**['startup_status']**['code'] == LOADING_WALLET_CODE:
No	old	elif status**['result']['startup_status']**['code'] == LOADING_WALLET_CODE:
#6	new	'(batch, new_rows, new_cols, **filters**)' if data_format='channels_last'.
No	old	'(batch, new_rows, new_cols, **nb_filter**)' if data_format='channels_last'.
#7	new	def _download_name(self, name, timeout=**None**, download_directory=None,
No	old	def _download_name(self, name, timeout=**conf.settings.download_timeout**, download_directory=None,
#8	new	'lynda returned error: %s' % **video**['Message'], expected=True)
Yes	old	'lynda returned error: %s' % **video_json**['Message'], expected=True)
#9	new	with mock.patch.object(**attr**, '__setattr__') as mock_setattr,
Yes	old	with mock.patch.object(**SettingsAttribute**, '__setattr__') as mock_setattr,
#10	new	**log**.info("Responding with %s infos", str(len(blob_infos)))
No	old	**logging**.info("Responding with %s infos", str(len(blob_infos)))

Table 2: Analysis of 10 out of the total 128 randomly chosen lines. For each line, its new and old version are shown. The result of evaluating if the change is to shorten a variable name is given in the first column of the second row.

We see possibilities for further research in this topic, by applying a different methodology. A possibility is to identify projects that did not use PEP8 initially, but have moved then to PEP8. We hypothesize that in these projects, the amount of changes in variable names is going to be larger.

Acknowledgments

This research has been supported in part by the Region of Madrid under project "eMadrid: Investigación y Desarrollo de tecnologías educativas en la Comunidad de Madrid" (S2013/ICE-2715). We would like to thank the reviewers for their detailed comments.

References

[1] Josh Lockhart. *Modern PHP: New features and good practices*. " O'Reilly Media, Inc.", 2015.

[2] Linus Torvalds. Linux kernel coding style. *Also available as https://www. kernel. org/doc/Documentation/CodingStyle*, 2001.

[3] Benjy Weinberger, Craig Silverstein, Gregory Eitzmann, Mark Mentovai, and Tashana Landray. Google c++ style guide. *Section: Line Length. url: http://google-styleguide. googlecode. com/svn/trunk/cppguide. xml# Line_Length*, 2013.