What does the community think about *Pythonic* code?

Jose J Merchante
Universidad Rey Juan Carlos
Fuenlabrada, Madrid, Spain
Email: jj.merchante@gmail.com

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

Abstract

Python is a programming language that has grown a lot in the last years. In the Python community there is a culture of programming the "right way", which is called *pythonic*. That term could be hard to understand in an empirical way, therefore it is necessary to understand what the community thinks about this "right way" of doing things. In this paper, we show the results of some interviews conducted with Python developers with different experience in the language. We want to know what the Python community sees as *pythonic* code and what is the importance. The results are diverse, but all agree on the final meaning and the relevance of *pythonic* code.

1 Introduction

During the last years, the Python programming language has become one of the most loved and popular languages in the world as we can see in surveys carried out by StackOverflow[1]. The impact of this language is also reflected on GitHub: Python is the second most popular language with over 1M projects and 40% more pull request in 2017 than in 2016[2].

Python is a general purpose language that is focused on the readability, being simple and also efficient. One principle of Python's design, guided by the *The Zen of Python*[3], says that "There should be one– and preferably only one –obvious way to do it". For an advanced programmer in Python there is always a right way of accomplishing a task that may be not obvious at first for beginners. In the Python community, "the obvious way" is called the *pythonic* way.

The term *pythonic* is not very used in introductory books, or beginners' tutorials. Neither is it in the Python Style Guide (PEP8), where it is mentioned once explaining four guidelines. And it only appears a few times in all the official Python documentation. In the Python glossary it is defined as "an idea or piece of code which closely follows the most common idioms of the Python language, rather than implementing code using concepts common to other languages".

In the following piece of code we can see a more practical meaning of the *pythonic* term. Many programmers who are beginners in Python will iterate over a list as they were in an other language:

```
# Get the string items
l = ["blue", 5, "yellow", 12, "abc"]
o = []
i = 0
while i < len(l):
    if type(l[i]) == str:
        o.append(l[i])
    i += 1
print o
```

However, even if the code runs and works perfectly, there is a more *pythonic* way of doing it:

```
items = ["blue", 5, "yellow", 12, "abc"]

str_items = [item for item in items
               if isinstance(item, str)]
print str_items
```

It is more readable, simple and also improves the performance.

We have been working in identifiying *idioms* from the code and analyzing the results [3]

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

[1] https://insights.stackoverflow.com/survey/2018
[2] https://octoverse.github.com/
[3] PEP 20 – The Zen of Python

Id	Exp.	Current employment
I1	6	DevOps Eng.
I2	15	Softw. Consultant, Python Trainer
I3	4	Chief Data Scientist
I4	3	SecDevOps Backend Eng.
I5	11	Researcher
I6	>6	Director of Eng.
I7	6	Software Developer
I8	2	Software Developer
I9	>10	CTO
I10	2-3	Student
I11	3	Chief Data Scientist
I12	1	Software Developer
I13	9	Infrastructure Automation Eng.

Table 1: Developers interviewed. The first column an identifier, the second column the years of experience and the third column the current employment

Many idioms can make the code more *pythonic*, but after all, there is not an empirical way to show their importance, and thereby we had to interview developers to know how the Python community defines "*Pythonic* code", and what they think about all of these structures.

2 Research method

For the interviews, we focused on getting information on the community opinion about Python idioms. We collect information about how they learn the idioms, the evolution of their code, and how important is this kind of structures in their companies.

In order to find candidates for the interviews, we asked for participation in the PyConEs (the most important conference about Python in Spain) and contacted several developers by email. The knowledge of the interviewees about Python ranged from 1 year to 15 years. They work in different companies with a diverse background as can be seen in the table 1.

Most of the interviews were conducted with a video chat software, some of them in person and two in written form. On average, each interview lasted about 15 minutes. All the interviews were recorded, transcribed into text and translated into English for the subsequent analysis.

At the beginning of the interview, we have put the interview in context and have introduced its purpose to the interviewees. We explained the research we are doing on *Pythonic* elements and possible future research.

The questions order varied depending on the interviewees's answers. We divided the questions in three main categories: (1) those related to the meaning and concept of the term *Pythonic*, (2) how developers became aware of the *Pythonic* idioms and how they approached them, and (3) those on the impact and spread of *Pythonic* in the Python community, including how it affects the professional environment. Some of these questions are in the appendix A, and the transcripts in the following link: interviews

3 Interview results

Some common questions asked to the interviewees are in the Appendix A. For a better understanding, the answers to the questions has been divided in the three main categories shown in the previous section.

a) Meaning and concept of Python idioms

The 13 interviewees know what *Pythonic* means. Most of them defined the term as code that is readable and easy to understand with the use of constructions that are provided by the language or its standard library. Some of them cited that *Pythonic* code is a code that adheres to the principles of the *Zen of Python*, and therefore it is code that is *beautiful*, *simple* and *readable* among others principles.

The most frequent example were *List Comprehensions*, a one-line statement to create a list in Python. Other examples mentioned were the rules from the Python Guide Style, the use of Python keywords and the way of using some Python statements like loops.

Depending on the experience with Python, interviewees explained *Pythonic* in a slightly different way. Those who are more experienced mentioned built-ins, efficiency, and structure more frequently, while the less experienced described it as a way of obtaining better styled code requiring a lower number of lines of code for solving a problem.

Regarding other languages, interviewees commented that idioms can also be found in languages like *Golang*. However, even though idioms exist in other programming languages, there is not always one typical way to code a task as there is supposed to be in Python, but there could be some exceptions like *Smalltalk* [1].

> *"In Python maybe there is a greater tendency to value that. In other languages, it is a quality that is not pursued by the community. But [in] Python from its origin, perhaps by chance or by its creation, I do not know, it is highly valued. It is a language that has many possibilities and more advanced functionality that makes many idioms. It is something that comes from the language, and in Python there is more [of it] than in other languages." [I2]*

Some of them pointed that in *Pythonic* code is not the same as idiomatic code; therefore, idioms can improve the code in the right hands, but for beginners it could make the code harder to understand.

All of them told us that *Pythonic* code is desirable. In their opinion, the more *Pythonic* is the code, the fewer errors the programmer will make.

b) How developers get to know Pythonic code

Most of the interviewees answered that they learned about *Pythonic* code from reading code in repositories of other projects. They also got to know *Pythonic* code from books, conferences and from colleagues at work. They pointed out that on *StackOverflow* you can find the best way to program a specific task, but it is not the best source to learn to program in Python.

Most of them think that understanding and reading idioms is easier than implementing them, nevertheless, their code has become more *pythonic* over the years.

When asked if they can differentiate the level of Python (such as beginner Python programmer vs advanced Pytohn programmer) by just reading the code, they point out that the knowledge of the usage of the idioms is a good signal and also the way of writing the code, for example from the use of *snake_case* instead of *camelCase*.

Among Python developers, those who use more *Pythonic idioms* are seen as having more expertise in the Python language:

> *"Pythonic and Python idioms are different things. [However,] Pythonic can be used to measure a developers skills, idioms can be used to (at least) measure a developers knowledge." [I5]*

c) Impact and propagation of Pythonic in the Python community and at work

In open source companies it is almost mandatory to develop *Pythonic code*. In companies where the code is not open oource, it is not mandatory, but is necessary to be at least well documented. In open source companies it is usual to make peer reviews in order to improve the code and in the process *Pythonic idioms* are frequently introduced. But once merged into the repository, code tends not to be changed until a bug appears.

When we asked if *Pythonic code* is positively viewed in a work interview, most of the interviewees agreed that writing *pythonic* way is usually associated with high expertise in the language, but also, it is important to show good programming skills instead of writing *Pythonic* code.

When they discover a new idiom, a majority of the interviewees recognised that they usually do not adapt their old code, but they incorporate into "their toolbox" and then when they modify something in the code, leave it better with their new knowledge.

4 Conclusion

This paper shows a better understanding of what *Pythonic* means for developers: why is it important, what are the benefits, how people adapt their code... On the other hand we have seen how programmers learn to program more *Pythonic*, what are the causes that motivate the use of idioms.

Finally we have seen how the idioms are being spread, what are the differences between companies that use and do not use open source code and also how important is the *Pythonic* code in interviews.

To sum up, the *Python* language has a huge community that focusses on the readability of the code and also the performance. The use of some structures in Python (idioms) is desirable because it improves both factors. It is necessary to teach Python programmers the idioms once they have learned the basics and therefore improve their code for both a personal and professional field.

5 Future work

This paper shows work-in-progress in our quest for identifying the importance of the use of idioms in Python. In the near future, we would like to do the following:

- We have seen idioms are *good* practices. We have noticed as well the existence of *anti-idioms* (similar to the patterns and anti-patterns idea [2]). We would like to identify them and see how often they are used.
- We would like to rank projects by their importance, first by omitting pet or student projects (for instance those that have a lifetime of less than 6 months) and second by giving a weight to projects by using data from the Python Package Index (PIP).
- We would like to study how Python idioms get propagated. This has two perspectives: how do Python idioms propagate, and how do developers learn them.
- Cluster Python programmers depending on their code, like scientific, scripters and software programmers.
- Use the *pythonic* code as a quality metric for software projects.

References

[1] Kent Beck. *Smalltalk: Best Practice Patterns.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[2] William H Brown, Raphael C Malveau, Hays W McCormick, and Thomas J Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[3] Jose J Merchante and Gregorio Robles. From python to pythonic: Preliminary results from searching for python idioms in github. *SATToSE*, 2017.

6 Appendix

A Interview questions

Group a: Questions related to the meaning and concept of Pythonic.

- Are you familiar with the term "*Pythonic* code"? Can you give me some examples?
- Is *Pythonic* code important? Why?
- Is the concept of "*Pythonic* code" different from the need of proper coding style in other program languages (e.g., Java)? What are the main differences?
- Could there be a dictionary of *Pythonic* code snippets or is writing *Pythonic* code more a style than a collection of concrete practices?
- Is the list of *Pythonic* idioms "open-ended" or could it be catalogued?
- When you program, do you rely on *Pythonic* idioms, structures and methods, or you do not think about it?
- Does the use/adoption of idioms facilitate programming tasks? And also to debug?
- When you discover a new idiom, do you adapt your old code? When? (Like immediately, next time, I maintain that file, or next item , I add functionality to the file)

Group b: Questions focusing on how programmers get to know Pythonic idioms.

- When was the first time that you read/heard about *Pythonic* idioms?
- Did you code become more *Pythonic* with more experience and years? why?
- When you began to learn Python, did you understand *Pythonic* idioms? If no, how and when did you understand them?
- Was it difficult understand or to learn some idioms (e.g., decorators)?
- Where have you learned how to write *Pythonic* code? At conferences? From StackOverflow? From coworkers or mentors?
- From books? (If at a conference or from books, ask which one)

Group c: Questions on the impact and propagation of Pythonic in the Python community and at work.

- Could you differentiate a beginner programmer from an advanced programmer from its code? How? Is the level of adoption of *Pythonic* idioms a good way to measure that?
- Do you think that be proficient on *Pythonic* code in your repositories or in a job interview could help to get hired?
- Do you think that be proficient on *Pythonic* code is desirable in any open source or industrial environment?
- In your work, is it mandatory to write *Pythonic* code or it represents a mere personal choice?
- Do you compare your code with your coworkers to recommend/teach a more *Pythonic* way to do some tasks? Do you read and learn Python idioms by reading the coworker's code?
- Do you push your code in any open software platform such as GitHub?
- What IDE do you usually use? Do you have any plugin to make your code more *Pythonic*? (pylint, pyflakes, etc.)
- How often will you use a tool that can make your code more *Pythonic*?
- What is more important: idioms or coding conventions established in a project? (Follow-up) In your experience, do they typically differ?