

From Python to Pythonic - Preliminary Research

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

Bogdan Vasilescu
Computer Science Department
University of California, Davis
Email: vasilescu@gmail.com

I. INTRODUCTION

Every programming language has its idioms and culture.

Every programming language has as well been designed to optimize certain tasks [3], [2]. This goes beyond a style guide (such as PEP-8 in Python¹), code conventions [1], or other complexity measures and tools commonly used in software development (as it is common for programming languages). There are plenty of tools that offer feedback to the developers about their programs, providing complexity metrics, warnings or even hint to the presence of *bad smells*. PyFlakes² and PyLint³ are an example of such tools in the Python ecosystem.

The mastery of a programming language can take years. However, there is a lack of knowledge of many of the idioms and advanced characteristics that a programming language includes. So even with years of experience, a developer may not be aware of them [5].

In our research we want to study the precense of advanced idiomatic elements in the Python language. Experienced Python programmers use these elements, called *Pythonic*, in order to have more readable, more concise and more optimized code. By identifying idiomatic Python we will be able to assess the skills in Python of programmers. We also plan to use it to identify learning paths, the presence or absence of certain idioms, etc.

II. IDIOMS

Kapser et al. provide a good definition of an idiom [4], although their research focus is on software cloning. There is an ample culture and public debate in the Python community about writing idiomatic code, referred as *Pythonic* code. However, there is nowhere an exact definition of what this means⁴:

“*Pythonic* means something like *idiomatic Python*, but now we’ll need to describe what that actually means.

Over time, as the Python language evolved and the community grew, a lot of ideas arose about how to use Python the right way. The Python language actively encourages a large number of idioms to accomplish a number of tasks (“the one way to do it”). In turn, new idioms that evolved in the Python

community have influenced the evolution of the language to support them better.”

Many beginners in Python with some experience in other languages could solve the problem of finding all countries in the `mylist` list that contain the letter “a” with following code:

```
countriesWithA = []
i = 0
while i < mylist_length:
    if "a" in mylist[i]:
        countriesWithA.append(mylist[i])
    i += 1
```

Although completely correct, it would not be considered *good Python*. We could improve it by using a built-in function such as `range`:

```
countriesWithA = []
for i in range(mylist_length):
    if "a" in mylist[i]:
        countriesWithA.append(mylist[i])
```

But this is not *Pythonic* either. The following is better, as it uses the `for` loop in a more *Pythonic* way:

```
countriesWithA = []
for country in mylist:
    if "a" in country:
        countriesWithA.append(country)
```

However, there is a Python idiom, list comprehensions, that specifically can be used to solve the problem of creating a list from another list (in general, by filtering it):

```
countriesWithA = [country for country in
                  mylist if "a" in country]
```

We have identified a set of advanced features of the Python language that are specific to the language or that are “embedded” into its culture. There are many developers that program in Python as they would do in any other language (i.e., without using *Pythonic* elements). Even, we have found 500+ pages introductory books that teach Python without including any of the elements considered *Pythonic*! Plenty of other books and web sites focus on these advanced elements of a programming language [6].

¹<https://www.python.org/dev/peps/pep-0008/>

²<https://pypi.python.org/pypi/pyflakes>

³<http://pylint.org/>

⁴The excerpt has been taken from <http://blog.startifact.com/posts/older/what-is-pythonic.html>.

An incomplete list of advanced features that we target is as follows:

- Comprehensions
- Magic methods
- Keyword arguments
- Structures defined in `collections`
- Lambda functions
- Context managers
- Decorators
- Properties
- Static methods and classes
- Class methods
- ...

Our research goal is driven by the fact that there is a lack of methodologies and tools that show the acquired level/skills in a given programming language and that potentially help developers to gain knowledge of new characteristics. This is mostly obtained by collaborating with peers and reading others' source code. Therefore, we present a framework proposal to evaluate the skills of a programmer in a given programming language, in this case Python, and want to study as well how these abilities affect the project in which these developers collaborate.

III. GOALS AND RQS

The RQs we want to address are following:

- 1) What is the extent of Pythonic skills present among Python developers in GitHub?
 - a) The more social (number of forks, number of projects involved) a developer, the more Pythonista⁵?
 - b) The more experienced a developer (number of commits, number of LOC, number of comments, etc.) a developer in Python, the more Pythonic code?
 - c) Can we find other characteristics that co-relate with Pythonic code (gender, age, geographic location, etc.)?
 - d) Are projects/repos clusters for Pythonistas?
 - e) Is the use of Pythonic elements constant over time? Is today's code more Pythonic than older Python code?
- 2) What are the advanced elements mostly used? Which are the less used ones?
- 3) How do Python developers learn advanced skills?
 - a) Can we identify the spread of Pythonic elements from experienced developers to other developers by means of collaboration (and/or forking)?
 - b) Can we identify a common learning roadmap?
- 4) Are Pythonista projects larger (in commits, in LOC)? Do they attract more developers? Do they have more forks?

⁵A Pythonista is a developer who codes using idiomatic Python.

IV. METHODOLOGY

Our current research methodology is based on mining GitHub, identifying those where the main language is Python (and not being a fork). These projects will be downloaded and analyzed by means of identifying advanced Python usage. To identify Python idioms we use a lexer⁶. We have therefore transformed the idioms in our list to tokens or a sequence of tokens that we look for in the Python code under scrutiny. This is an approach based on heuristics, but although as by now we do not have values of precision and recall, our first experiences show that it provides high values of both.

The author and time of the code snippets identified by means of the `blame git` command will allow us to assign that snippet to an author and obtain statistics on a project and on an individual and project basis.

V. THREATS TO VALIDITY

As with all empirical studies, there are a number of threats to its validity. The following is an uncomplete list of threats that we have to consider.

- Our list of Python idioms considered as Pythonic will always be incomplete, and because of its heuristic nature prone to errors in identification.
- Although the number of Python projects hosted in GitHub is large, it is by no means all the Python code that exists. Much Python code is still using Mercurial, or may be hosted in other forges and repositories.
- We are aware that the occurrence of patterns may be influenced by the Python libraries used. Actually, some of the Python idioms refer specifically to the use of such libraries.
- Small and large projects may have completely different characteristics. This makes it hard to identify skills of individual developers.

VI. ACKNOWLEDGMENTS

The work of Gregorio Robles has been funded in part by the Region of Madrid under project "eMadrid - Investigacin y Desarrollo de tecnologas para el e-learning en la Comunidad de Madrid" (S2013/ICE-2715) and in part by the Spanish Government under project SobreVision (TIN2014-59400-R). The authors would also like the BENEVOL 2015 anonymous reviewers for their feedback and suggestions.

REFERENCES

- [1] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In *FSE*, pages 281–293. ACM, 2014.
- [2] J. Coplien. Advanced C++ programming styles and idioms. In *Tools*, page 352. IEEE, 1997.
- [3] J. O. Coplien. Idioms and patterns as architectural literature. *IEEE Software*, (1):36–42, 1997.
- [4] C. Kapsner and M. W. Godfrey. "cloning considered harmful" considered harmful. In *WCRE*, pages 19–28. IEEE, 2006.
- [5] H. P. Langtangen. *Python scripting for computational science*, volume 3. Springer, 2006.
- [6] A. Martelli, A. Ravenscroft, and D. Ascher. *Python Cookbook*. O'Reilly, 2005.

⁶<http://pygments.org/docs/lexers/>