From Python to Pythonic: Searching for Python idioms in GitHub

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

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

Abstract

This paper presents our work in progress. Our study is focused on creating a web tool that can help beginners and advanced programmers to make their Python code more legible and readable with the use of Pythonic idioms, that is, using typical ways to accomplish some tasks.

1 Introduction

Every programming language has its culture and usual way to code a task; that's what programmers usually call *idioms* [**?**]. For an advanced programmer in a given language there is always a *better* way of accomplishing a task that is more suitable in that language (e.g., it improves its readability) instead of writing the implementation it replaces in the same way as in another language.

Python is a programming language that in the last years has grown a lot. For this language there are many tools that check the code against very common style conventions (such as the ones specified in the PEP-8[1]), but there is to our knowledge no tool that identifies what idioms a program contains, or that helps improving your Python code making it more idiomatic (commonly referred to as more *Pythonic* [**?**]). Even though no such tool exist, the Python community is concerned a lot about these issues and many books, articles, talks and references on how to make your Python code more Pythonic can be found.

Many Python books and web pages explain the language without including these idioms, and focus on explaining the language as it would be another programming language, but with Python syntax. As an example, the following is correct Python:

```
colors = ["blue", "red", "yellow"]

for i in range(len(colors)):
    print colors[i]
```

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

```
colors = ["blue", "red", "yellow"]

for color in colors:
    print color
```

For these reasons, we think it would be a good idea to analyze Python idioms and create a web application that can help beginners and advanced programmers to make their Python code more legible, readable, and write the task the *right*, *Pythonic* way.

2 Methodology

For the implementation of this tool we have searched for the most important Python idioms in books and talks at PythonCon, the most important conference on Python. Like this, we have collected Python idioms of various difficulty levels. This list contains idioms such as:

- Comprehensions
- Magic methods
- Lambda functions

Proceedings of the Seminar Series on Advanced Techniques and Tools for Software Evolution SATToSE 2017 (sattose.org).
07-09 June 2017, Madrid, Spain.

[1]PEP 8 – Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/.

- Decorators
- Collections structures
- Class methods
- Closures
- ...

In order to identify the idioms in Python source code, we look for tokens in Python code. For some idioms, this is straightforward (e.g., for the *with* keyword), while for others it is more complex (e.g., for list comprehensions, in particular those that contain Boolean logic). Then, we scan Python code in a project with a lexer called `Pygments`[2] for these idioms.

For each idiom found, we also retrieve and store meta-information from the versioning repository, such as its author (we therefore use `git blame`). That allows to obtain the Pythonic contribution of each collaborator to a GitHub project.

In order to improve our tool and to perform a first study of the use of Pythonic elements in real Python code, we mined all projects with Python as main language from GitHub. We used GHTorrent [**?**] as data source for knowing the main programming language of a repository[3], of which 700,000 (out of over 15 million) were in Python. Out of these, we downloaded a sample of 70,000 projects.

Given the amount of data to be downloaded and analyzed, the analysis of the repositories took five weeks. Projects were downloaded, and then non-Python files in them were removed in order to store space. All in all, at the end of the process we had a total of 500 GB of Python files. These were analyzed with our tool. The output was redirected to a database and analyzed with Pandas [**?**].

3 Some preliminary results

In this section, we present some preliminary results.

Figure **??** shows the number of idioms per repository. This curve shows a powerlaw distribution with a long tail: many of the repository have few idioms, although more than 50% of the repositories have more than 40 idioms – the mean is 304.52. Noteworthy is the fact that around 12% of the repositories do not contain any of the Python idioms in their code.

Figure **??** displays the most used idioms by number of repositories, counting an idiom only once per repository. As can be observed, decorators and list comprehensions are two of the most common Python idioms, although the most frequent ones are the

[2]http://pygments.org/

[3]We do only consider those repositories that are non-forks.

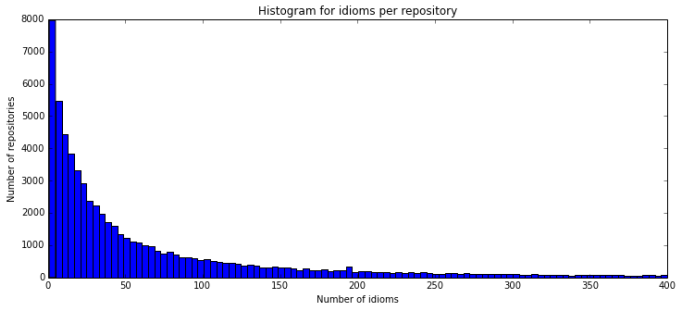

Figure 1: Number of idioms per repository

use of named arguments in function calls and *docstrings* (attached comments in a specific format that serves as documentation for classes and methods). The `if __name__ == "__main__"` construct is also at least one time in a repository for over 80% of the cases.

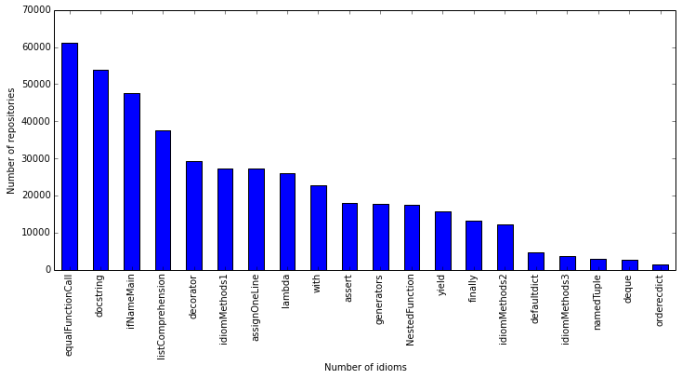

Figure 2: Ranking of most frequent Python idioms. Idioms are counted only once per repository.

Some of the most used and known Python idioms are *magic* methods (methods that are invoked when using a certain syntax, starting and finishing with __, also known as *dunder* from *double under*). These methods provide characteristics that, if implemented, are *transversal* to classes and thus provide a common, defined way for some functionality. An example of this are the __repr__, which returns a printable representation of an object, and __str__, which returns a string containing a nicely printable representation of an object, methods. Figure **??** gives the number of occurrences of each of the most frequent magic methods. This time, the total number of appearances, i.e., a magic method that appears N times in a repository will count N for that repository, is given. The most frequent one, `__init__` (the initializer method, used in Python in a constructor-ish nature), is omitted in the figure as it appears many more times than the rest.

As can be seen from Figure **??**, __repr__ and __str__ are the two most frequent magic methods. The next ones are: __call__ that implements a function call operator, __iter__ to create iterators, and __unicode__ which

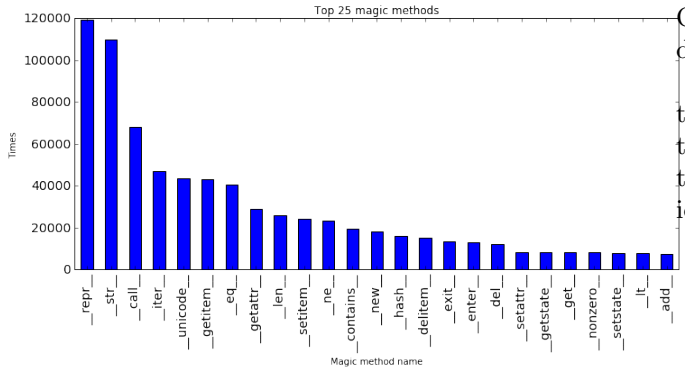

Figure 3: Most common magic methods (__init__ not included).

in Python 2 returns characters as __str__ really returns bytes[4].

4 Work in progress

From all the repositories analyzed, we have extracted some information about the importance of Pythonic idioms. Right now, we are developing the web application that will allow to improve the knowledge in Pythonic idioms to anyone.

The application is running over Django, a high level Python framework that encourages rapid development and clean, pragmatic design[5].

When a user enters his username of GitHub in the web application, we select his profile repositories in a first approach and let him introduce others that he has contributed to. When the repositories are cloned, the tool filters the Python files looking at the extension or the first line of code. In the figure **??** we can see the main page of the actual work.

Figure 4: Main web page of the application

Our work in progress is to identify the level of a user and give a mark about his Python knowledge.

[4]In Python 3, the __str__ magic method returns characters as in Python's 2 __unicode__, and a new __bytes__() magic method exists.

[5]https://www.djangoproject.com/

Our first approach is to classify the idioms in three different levels depending on the difficulty to be learnt.

For example, analyzing a previous version of the tool that extracts Pythonic idioms, we got the mark that is in figure **??**. That is a good mark, but is also tricky, because in this repository there are test for each idiom in Python.

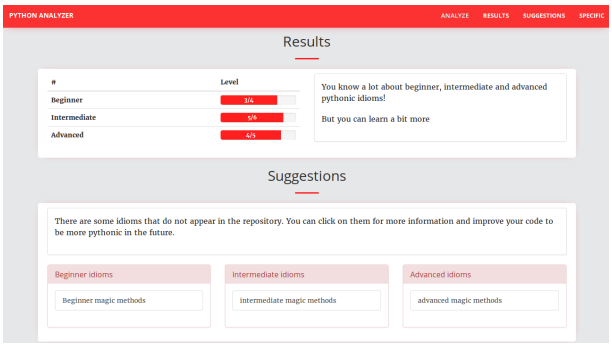

Figure 5: Results of 'Pythonic analyzer' repository

We are working on better metrics that adjust better to the reality and by this way, assign a developer a level of mastery in the Python language and some path to improve his level.

5 Future work

This paper shows work-in-progress in our quest for finding how idioms are used in Python. In the near future, we would like to do the following:

- Extend and assess the list of Python idioms. We would like to have a list of idioms as complete as possible. These should be evaluated by Python developers.
- There are idioms that are conceptually more difficult than others. We would like, again with the help of Python developers, see if we can classify the idioms by their complexity.
- If idioms are *good* practices, we have noticed as well the existence of *anti-idioms* (similar to the patterns and anti-patterns idea [**?**]). We would like to identify them and see how often they are used.
- We would like to filter 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

new Python idioms propagate, and how do developers learn them.

6 Acknowledgements

The work of Gregorio Robles has been funded in part by the Region of Madrid under project "eMadrid - Investigación y Desarrollo de tecnologías 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).