From Python to Pythonic: A study of Python idioms

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

Our study is focused on developing a 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. The study of Python idioms could give us a quantitative measure about how are used, but we also need a qualitative measure to obtain information on what the python community think about the use of this structures and also improve the tool and do more research on this topic. We are now making interviews to Python programmers and we are going to show the procedure, the results of the interviews and also a brief overview of the actual project.

1 Introduction

Every programming language has its culture and usual way to code a task; that's what programmers usually call *idioms* [2]. 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* [6]). 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
```

These are some reasons that shows that the Python code could be better with the use of idioms, but after all, there isn't a empirical way to show their importance in the python code, and thereby we have to make some interviews to know what the Python community thinks about all these structures. Also, this information could be useful to improve the tool we are developing to help beginners and advanced programmers to evolve their Python code, making it more legible, readable, and show them how to write the task the *right*, *Pythonic* way.

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

2 Interviews in empirical software research

Interviews historically have been frequently used to study human behaviour. The purpose to use them in empirical studies is to obtain information about a topic that cannot be measured using only quantitative methods. However, interviews are a useful method for data collection but are also very time-consuming activities because you have to spend time planning the questions, conducting the interview and analysing the results.

The most important part of making interviews is the preparation. All the questions must be very accurate and focused in the main topic. There is usually a limited time to ask the interviewees and sometimes you just have one opportunity to do it, therefore a previous preparation is really important in order to collect the most information as possible.

The second most important part of an interview is how to conduct it. There exist a large amount of literature about how to do it [4]. Some important skills are create an atmosphere of trust, not expressing disagreement with what the interviewee is saying, letting him talk, paying attention to avoid asking questions answered and also expressing the questions clearly among others issues.

And last, but not least, is to analyse the results of the interview. Good questions and accurate interviews may show that this part is easier to do, but that does not mean that it consumes less time. Occasionally, the interview should be transcribed into text to extract the information of all the interviewees and merge to know what they think in general.

3 Phases for the *pythonic* interview

3.1 Preparation

The interview focuses on getting information about what people think about Python idioms, which are the most important, how they cooperate to propagate them, and how important are in the code.

For the preparation of the interview questions, we have found in the literature that there are different types of questions [5]: open-ended questions for the interviewees to talk freely about the topic, closed questions for specific details and hypothetical questions about a situation.

In order to complete the preparation, once the questions are ready, these should be studied and also answered by the interviewer to detect repetitions or the possible lack of information.

In the results sections, you will see the answers to some of the most relevant questions asked to the interviewees.

3.2 Interview

The interview duration lasted 11 minutes on average. All the interviews were recorded in order to pay more attention to what he was saying and avoid a possible loss of information.

At the beginning of the interview, we put in context and inform the interview purposes to the interviewees. We explain the project we are doing as well as possible future research.

Some good interview skill [3] are to encourage the interviewees to talk freely, ask relevant questions, not repeat answered questions and follow up and explore interesting topics.

There are also some important notes to take in the interview, such as python experience, for what purposes they use python, and also the surroundings in their work. This information may be a bit personal, but it could be very important in further analysis.

3.3 Analysis

For subsequent analysis it is very important to transcribe the records into text. There are some useful voice-to-text tools, but they are still not very accurate for interviews or conversation where there are two or more people talking. Therefore, the consequent transcript is somehow slow and should be done by listening multiple times to make an accurate representation of what we were saying.

4 Results

In this section we show some of the most relevant questions answered by the interviewees. We can differentiate the questions in three main groups. General questions related with the language in general, specific questions focusing on some details on how programmers learn python and also relevant social questions to know how the idioms propagate in the community.

We have done four interviews to python programmers with different levels of python, from 1 to 8 years. One of them came from a different language programming (Java), and could be considered important due to it is lack of knowledge on python and how is he learning the idioms. They all come from companies developing open software.

4.1 General questions

To get started, we asked a question to know how much they know about python idioms and also to know if we have the same idea about the definition of pythonic. One of them found the questions difficult to answer, he knows a lot about the topic, and summarizing a definition in a sentence could be difficulty in a first a

approach. They all know the concept, they related the meaning as write python in less lines, more readable, improve in some cases the performance in the code and the use of those structures make python a powerful language. Some examples they gave were mainly list comprehensions and decorators.

Other questions have been asked about whether the pythonic code means that an advanced programmer has written it. All of them agreed that the more appearances of idioms you see in the code and varied, the more pythonista that person should be, because he should understand the language very well. Some of them also pointed out that some idioms might degenerate the python level if aren't used well, such as multi-line list comprehensions.

Focused on which phase of learning python you have to learn the idioms there are different opinions. Some of them think that you should learn them when you know all the syntax of python; you are programming the task as you were in another language, and as time passed, you should improve the knowledge in python and hence become a expert in python. On the other hand, they think that the idioms should be acquired while learning python, that is, while you are learning how to make a list, they have to learn also how to do it in a pythonic way. This debate is also in the Python books, some of them teach the idioms while you are learning the language and also most of them do not mention nor list comprehensions.

There is another question to get details about how complex the idiomatics expressions are in terms of reading and implementation. In this question, the answer is basically the same from all of them, it depends on your Python level. For beginners Python programmers, understanding an idiom for the first time can be difficult, but then it is easier. Implementing the idioms is a bit more complex in some situations such as decorators, but in others like methods from the package "collections" or "list comprehensions" could be easier.

4.2 Specific questions

For the specific questions they tend to response similar answers. They used to use python idioms as much as possible because they think that could improve the readability. Their code has evolved over the time, when they started programming they didn't know so much about pythonic programming, but now they are using more idioms in their code.

Sometimes, but not frequently, they use to search in stackoverflow specially, what is the pythonic way to write a task, they argue that sometimes help them to learn something new and also improves the readability of the code writing it in less lines and with a better syntax.

They also say that they use a IDE and install some plugins to help them to write more pythonic in terms of syntax and if there were some plugin to identify anti-idioms and the possible python alternative, they will use it everyday.

4.3 Social questions

In the social questions we try to understand how the idioms get propagated over the community, how do they learn them and also the advantages of using these structures in a social aspect.

To the questions how do they learn the idioms, they affirm that most of them are learnt reading python code of other people, that is the main way the idioms get propagated. Some of them affirm that they have seen some conferences of python with people talking about what is the pythonic way of coding some tasks, or how to turn python code to pythonic code. They admit that haven't read many python books because most of them doesn't show the pythonic way to code a task and just explain the python syntax.

They think that code in a pythonic way in interviews or uploading pythonic code to GitHub increase the opportunity to get hired in a company in contrast to write the code like you where programming in another language.

Among them they admit that use to show how to write more pythonic and some tricks for improving their code, they review the code before they push to GitHub and try to improve it to be more readable.

For the Java programmer, he says that python idioms are very easy to understand, he used to write python as he was programming in Java, but with the time, he is learning how to write the pythonic way because their cowerkers offer him help to improve his code to be more readable for python programmers.

5 Conclusions

We have obtain information about the interviews but we have also to obtain more in order to find patterns and possible differences about what do they think about python code. This is a work in progress, the idea is to do about 20 interviews and show a better results. In general, in these interviews, we can extract that idioms are necessary for a better code and for improving the readability. They used to cooperate between them for obtaining a better code and also learn a lot from that technique.

6 Work in progress

We are conducting some interviews and also developing the tool to make the code more pythonic

and therefore make an evolution of it. The benefits of the pythonic code can be extracted from the interviews and we are now looking for new participants in order to obtain more accurate results.

The results of the interviews are contributing to develop a tool to help python programmer to improve their code with python idioms.

The application is running over Django, a high level Python framework that encourages rapid development and clean, pragmatic design[2].

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.

Our work in progress is to identify the level of a user and give a mark about his Python knowledge. Our first approach is to classify the idioms in three different levels depending on the difficulty to be learnt.

For example, analysing a previous version of the tool that extracts Pythonic idioms, we got the mark that is in figure 1. That is a good mark, but is also tricky, because in this repository there are test for each idiom in Python.

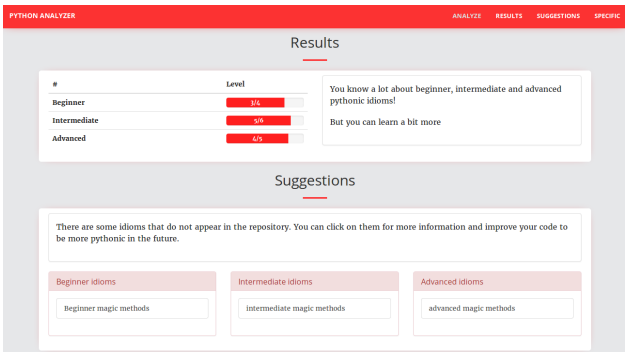

Figure 1: Results of 'Pythonic analyser' repository

We are working on more precise metrics that can be used to assign a level of Python mastery to developers and some path to improve his level.

7 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 [1]). 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.
- Cluster python programmer depending on their code, like scientific, scripters and software programmers.

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

8 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).

References

[1] W. H. Brown, R. C. Malveau, H. W. McCormick, and T. J. Mowbray. *AntiPatterns: refactoring software, architectures, and projects in crisis*. John Wiley & Sons, Inc., 1998.

[2] J. Coplien. Advanced c++ programming styles and idioms. In *Technology of Object-Oriented Languages and Systems, 1997. TOOLS 25, Proceedings*, pages 352–352. IEEE, 1997.

[3] S. E. Hove and B. Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10 pp.–23, Sept 2005.

[4] S. Kvale. *Interviews: an introduction to qualitative research interviewing*. Sage Publications, 1996.

[5] S. Merriam. *Qualitative Research: A Guide to Design and Implementation.* Higher and adult education series. John Wiley & Sons, 2009.

[6] G. Van Rossum et al. Python programming language. In *USENIX Annual Technical Conference*, volume 41, 2007.