

## What is *Pythonic* code for you?

1. It is a code that adheres on the one hand to the principles of *The Zen of Python*, which can be seen when doing “import this”. There are some of them that are a bit abstruse but that you can get to understand. *Pythonic* code is in general flat code, little convoluted, legible and within those principles even that follows a kind of *idioms* or of patterns that are rewarded by the community as the most readable or the most used. They are usually *idioms* that do not prioritize the deficiency but the readability.
2. It is a code that makes use of the expressiveness of the language so that in a more compact and clear way to transmit to the one who is reading the code what is the objective of that line or of that piece.
3. The code that you can read without having to ask yourself what is doing that.
4. It is that code that follows some rules that are based practically on the pep8. The pep8 is developed by experienced programmers who want the code to be self-documenting and beautiful (In the examples question he defines also structures).
5. *Pythonic* code is just an elegant and readable code that makes use of constructions/tools/things that are provided either by the language or by its standard library (meaning that there is no reason to not use these as they do not require external libraries or "hacks"). Readability is achievable in most languages (not in Perl, though), but Python makes it very easy to achieve, while keeping the code simple and efficient.
6. It is a code that follows the most accepted standards in the community and that uses the best option within the available tools.
7. It is hard to explain. The good thing about Python is that it forces you to structure the code in a certain way. Tabulation and spaces as they already come predefined allows you to make the *pythonic* code.
8. It's like a way of writing. What would have to be written in several lines of code could be in less. In a more concise way.
9. Code that takes advantage of python-specific functionalities.
10. For me the *pythonic* code, is all that code that takes advantage of the advantages offered by the Python syntax, to make a more efficient and pleasant code for the programmer.
11. It is clean, unambiguous code and follows certain conventions that are more established in the Python community.
12. It is the code that uses a series of *idioms* that are what make Python a powerful language that in other languages you could not do.
13. Is the *idiomatic* way of write Python.

## Can you give me some examples?

1. For example, check if a key exists in a dictionary using `in`. There are several methods to extract the list of keys or whatever you want but it is normal to use this because above all it will be worth for several different types of data.
2. What is more common in Python are the *list comprehensions*: it is something typical of Python that is not seen in other languages and not only by space but because it is easier to see what is being done.
3. `pep8`, tests. The *list comprehension* we also use them a lot.
4. It shows when a person has had a lot of experience in other languages, for example in spacing, and the use of `pep8`. But it also goes further, as the *list comprehension*, the structures, the searches in C is with an array, in Python with `"in"` it becomes more *pythonic*. Using the built-ins well ...
5. It's hard to give an example of *pythonic* code, but I can give an example of common Python *idioms*. By the way, that's not the most readable way to achieve the same result:

```
\n'.join((
    '{}: {}'.format(i, example)
    for i, example
    in enumerate(data_dict.get('examples', []))
))
```

Note that *pythonic* is different than simply using *idioms*. For instance, while `a_dict.get(key, default_value)` could be considered as a *pythonic idiom*, `a_dict.get(key)` could similarly be considered as an *idiom*, but is not really *pythonic*. (there's no added value compared to `a_dict[key]`).

EAFP is an *idiom* and is usually *pythonic* (to some extent, because one smart `"if"` can replace many `try/except` and be more elegant and readable).

6. For example, a *list comprehension* instead of a loop in only one line.
7. For example the *list comprehensions*, any type of iterators or generators or the `yield`s, when you see them you start to see how people use them. For example, the use of dictionaries perhaps, in Python, is like day to day. And it depends on how people use the dictionaries and if you use more or less objects you can understand the level a bit.
8. For example the use of a *list comprehension* in one line instead of two `"for"` in 4 lines.
9. The most obvious are the *list comprehension*.

10.

BAD:

```
i = 0
```

```
while i < len(data):
```

```
    print("Now i is {}".format(i))
```

```
    print("The value is {}".format(data[i]))
```

```
    i += 1
```

```
print()
```

GOOD:

```
for idx, value in enumerate(data):
```

```
    print("{} --> {}".format(idx+1, value))
```

11. Ways to write code that is more explicit, for example the *list comprehension*.

12. The *list comprehensions* for example, when doing it in a single line, you can do the whole loop that generates a whole list. The decorator of functions that is something that is not available in other languages and allows you to make operations on functions that are already done.

13. For example sometimes the use of *list comprehensions* instead of "fors" and also some modules from standard Python library.

## **Is in your opinion the concept of *pythonic code* different from the need of proper coding style in other program languages (e.g., Java)?**

1. In Golang it is true that there are a number of *idioms* but it is not so strict with them. On the one hand the syntax is strict (there are linters that leave you the code in the right way) but it is not as relevant as in Python and reading a couple of documents you see everything important in Golang.

2. The concept is the same, many people look for the aesthetics of the code (nice and clean that applies to every language, even for a neophyte it would not be hard to see that something is better or worse) and it is something that applies to all languages.

In Python maybe there is a greater tendency to value that, in other languages, it is a quality that is not persecuted by the community. But 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 functionalities that makes many *idioms*. It is something that comes from the own language and that in Python there is more than in other languages. In PERL there may be a hypercomplex way of doing something that makes everything very simple but it is so difficult to understand. However, in python, the most *pythonic way* is the clearest, the best or even the most optimal one in computational sense, and that makes it a little better to use.

3. We work a lot in C and JS. We have acquired some rules to make the code more *pythonic* that we have taken to other languages, such as the fact of being explicit, ordered ... It is something that goes with the community.

4. x

5. I think yes, it's different, but it's very difficult to explain why. The thing is that you \*can\* actually write "proper code" in Python without being *pythonic*, similarly to what you can do in other languages. *pythonic* is more like "using the right tool at the right place", and by "right tool", I mean "everything that's provided by the language and its standard library". Similarly, while there are many *idioms* in Python, using them does not mean that you're writing *pythonic code*. Sometimes, *idioms* make the code less readable, or more complicated (for instance, using `reduce` can be seen as *idiomatic*, but in general results in a code that is not easy to understand, and thus is not *pythonic* to my eyes).

6. In Go, it is true that there is a lot of interest in the subject and they make an effort. In C for example there are ways to write also, how you write things, how structure the file.

7. X

8. X

9. It is important to adapt to the conventions of each language. If you are Python programmer, expect *pythonic* code, if programs in Java *pythonic* patterns decreases readability.

10. Yes, PERL

11. In other languages, according to my experience, which is not much, there are certain conventions when you approach the style but not the way to solve a problem. For example in Perl there are many ways to implement a task while in Python there is one that is doing well.

12. X

13. The *idioms* are in other languages and use the term. But not in the same way. For example in Go the use of *idiomatic* terms is something important, but it includes also some structures that are best practices. In other languages the *idioms* doesn't focus the performance.

## Do you think *pythonic* code is desirable?

1. It is desirable, there can always be a very good programmer and that simply by rebellion says “this code is for me and I will do as I want”, but in general there is a conception that you make the code more readable for the rest of the world, the more you adhere to these principles, the easier it will be to read and understand. Someone when looking at the *pythonic* code finds the structure easier and it costs less time to understand the logic of the program. If you do not follow that logic or you are revealing them, you are showing that you have not understand the essence of the language.
2. Readability is one of the first objectives. I am an advocate for those who believe that the code is read many more times than it is written. The more readable it is (as compared to prose or literature) the less difficult for you to understand will be. It helps you detect errors and even generates fewer errors by having a more readable code and in general all this leads to a better quality
3. Yes, because you have to think about the future of the project. Document the project keeping it descriptive.
4. X
5. I think that *pythonic* is, by definition, a desirable property of a code, and probably one of the most important one (at least for me), largely before efficiency (if you're looking for an efficient code, you probably won't write it in Python).
6. X
7. X
8. On the one hand it shows that someone knows how to write but on the other hand if you do not know this way of writing it can be quite complex. But it reduces the programming time.
9. It depends, I give more priority to that the code is readable instead of being *pythonic*, using a *idiom* can improve or worsen the readability of the code. For example with list comprehensions there are times that it helps readability and others that can be counterproductive. It depends on what you are doing. I am not very strict. I choose one thing or another depending on the context.
10. It is desirable to improve the level of python, but a developer is not more advanced if he uses *idioms*.
11. Because a large part of the complexity in the development of projects comes from understanding what other people are doing in their code or maintaining code from a while ago. Therefore, being legible and easy to maintain is fundamental.
12. X
13. The comprehensions are optimized, therefore the use of map is less *idiomatic* than list comprehension because is less optimal.

### **Which characteristics do you think are the most important?**

1. Some are aesthetic like the methods used. The variables have snake\_case format. The use of docstring to document the methods and even the indentation. If you see a code with many levels of indentation, surely there is another easier way to do it. With seeing the code in general you can see if it smells. The *non-pythonic* code you see a very strange way of doing tasks, and very long variables or camelCase, it is not mandatory to be snake\_case, but it is usually done by people who start in Python.

2. Above all readability. Do things that are ingenious, the tasks that are done every day are much clearer, easier to understand. You read it and it seems that you are understanding the program, even though it is not like that either because it is complex and maybe you work at many levels of abstraction but when you see the *pythonic code* you understand what is happening. I do not think there is a *pythonic code* that is not good.

3. Clarity and conventions.

4. X

5. Because writing elegant and readable code is important. (The definition again) *pythonic code* is just an elegant and readable code that makes use of constructions/tools/things that are provided either by the language or by its standard library (meaning that there is no reason to not use these as they do not require external libraries or "hacks"). Readability is achievable in most languages (not in Perl, though), but Python makes it very easy to achieve, while keeping the code simple and efficient.

6. When you program with Python code, you are on a different level of abstraction and the most important is that the code is more legible in that way.

7. X

8. X

9. (Couldn't ask this question, didn't consider *pythonic code* important)

10. Yes, firstly for oneself as it helps to have a more effective purification and many do a considerable efficiency improvement.

Obviously if you work in a group it helps to share code in a much more pleasant way for all the developers of the team.

## Where have you learned how to write *pythonic* code?

1. The first of them with O'reilly Learning in Python that already makes mention. But reading about *pep8* and "Python cookbook" are already *ultra-pythonic* recipes.
2. I am learning them in a staggered way, and I still have to learn. The sources are the blogs of concrete people, some book (Python Cookbook) and especially reading community code. Source code more influence on me: django REST framework (tom christy) readable and complex. And also from the Python libraries a lot, you can get many useful functions.
3. From PyCon Barcelona and also from StackOverflow.
4. I have read many books and there are times when you find yourself stuck and StackOverflow show you multiple points of view of people and you always learn.  
In interviews. I have done many and at first I was very new and they caught me, and I was curious about the problems they had, because they asked a lot about this and that's when I read more about this.
5. Documentation, code, discussions, ... it's hard to be exhaustive. I think the most valuable source of inspiration was the official documentation (including the one of the standard library).
6. Reading code and contributing, if something is broken or something is not working well, you learn new things. In the end you end up learning new code in free software platforms.
7. I have seen videos of Raymond Hettinger and the truth is that his talks are very interesting and forces you to think more in *pythonic* than in another language and well from there I got a little more *pythonic*.
8. X
9. In blogs especially. In books I use many but for *idioms* it's not where I got the information from.
10. Conferences, Fluent Python book and open source projects.
11. In meetups and I have read about it. To find how to do things in stackoverflow, but many times it is not the most *pythonic* way
12. X
13. Conducting activities in the community of Python Barcelona. You can see the code from a lot of people and you can see a lot of *idioms* that I didn't know. Fluent Python, but it doesn't show many *idioms*. Reading open source projects, like flask, bottle. There are some that are not very *pythonic* like ansible, that tool doesn't focus a lot in being *pythonic* but is very popular. Sometimes is acquire some standards that if someone reads your code can understand it.

## Could you differentiate from a beginner programmer and advanced programmer from its code? How?

1. (Answered in "Which characteristics do you think are the most important?")

Some are aesthetic like the methods used. The variables have snake\_case format. The use of docstring to document the methods and even the indentation. If you see a code with many levels of indentation, surely there is another easier way to do it. With seeing the code in general you can see if it smells. The *non-pythonic* code you see a very strange way of doing tasks, and very long variables or camelCase, it is not mandatory to be snake\_case, but it is usually done by people who start in Python.

2. Without any doubt, it can be seen even if it comes from other languages, for example how he/she organizes the code (functions and classes). Use of *getter* and *setters* instead of *properties*, the type of loops (variables to count instead of simple fors), *enumerate*, *collections*, *functools*. They are part of the Python library but they only start to be used when you are a little more initiated. The own *context managers* that can be useful and expressive and that can be complicated for one initiated.

3. I could not tell you, someone newer can be more *pythonic*.

4. It's easy, first it does not make use of many of the built-ins or structures, like range or xrange in Python 2. He / she does things that in other languages looks good, but not in Python. For example in Python there are generators that novice programmers in Python would not know how to use

5. Yes, there's even a strong difference between "advanced programmer in language X" that writes code in Python and "advanced programmer in Python" that writes code in Python. (ie. you can write perfect code in Java and translate it to Python, it won't be Python code to me)

6. Yes, the style, when you read good code and with pleasure. When it is simpler more it is from advanced Python programmer.

7. X

8. X

9. I would look at the Python's own functionalities, in the data structures and in the way to organize the code.

10.

junior - simple code but with errors "holes"

mid - complicated code

senior - simple code, *idioms*, *pythonic*

11. Yes, due to the lack of *idioms* when programming and resorting to doing tasks that already have Python done in itself

13. The *idioms* he uses, how, and why. But it's not the only variable. For example, if he knows to program in different ways. I mean, for example, I have seen data scientist that program as data scientist, rarely you see them use classes. If you use Python for data scientist, web programming, or other area, you can see different ways to code a task.

## When you program, do you rely on *pythonic idioms*, structures and methods, or you don't think about it?

1. I search in the Standard Python Library, I look for documentation, or some tutorial example that I can take advantage of or even in some books. There I will look for the essence.

If I'm going to look in stackoverflow I'm going to look for recipes of something that I have not been able to find there and the answer they give you does not have to be *pythonic*.

2. The first time I write the code, it always comes out dirty. Peter Norvic, program simple and clean since the first time, I see it and mine is never like that the first time. After iterations I get it, sometimes I write *idioms* to the first I code, but until I don't refactor my code it is not satisfactory. But the basics is the natural way (looping, defining classes, using functions and decorators).

3. I usually look at the *pep8* if it is written, but in StackOverflow they always comment on the best way to do it.

4. X

5. I don't think about it until I read my code and I see horrible (or not so horrible) things. At that point, I say "Ok, there must be a more elegant way to do it".

6. No, once you learn it you use it, it has value like that, but not looking for it.

7. (Similar answer in where did he find the *idioms*).

8. X

9. Frequently

10. I look for the way to code a task in github repositories, or simply in the Python documentation.

11. I write the code without thinking, I use them but I do not think about it.

12. I try to find a way to do one thing in Python more readable and also to try to learn the code of others.

The *idioms* under my point of view are not always readable and you do not know what you are reading. For example the list comprehension that seems at first glance very clear is not all that clear when you arrive and you see it in a single line it is sometimes complicated to see the variable and see what you are changing but nevertheless you have the lambda functions etc, but when you see it for the first time it scares. And now it costs me much less and by understanding what you're doing, it's like reading music.

13. X

## Do you code become more *pythonic* with more experience and years? Why?

1. Yes, I use them more but it is not a gradual use, when you read a lot like in the *pep8* it gives you an overdose of conventions and you will remember a large part and then after using linting tools like *flake8* that reminds you and that is the moment when you start to apply it more consciously because otherwise it will give you the error.

All the Python elements that can be recognized by a linter can help you learn more.

2. X

3. Generally speaking, yes, but you acquire bad habits that are not so *pythonic* that you drag them.

4. In scripting it was all very simple, you did not use classes and you did not use any of that. The I began to understand all about *pythonic* meaning.

5. Yes, of course. I read a lot of things about Python, I read a lot of code, I read a lot of documentation, etc. With an increasing number of "good examples", it felt more and more natural to write *pythonic code*.

6. Yes, there has been an evolution in the language, now in Python 3 it is all more *pythonic*, we started in PHP, later in Python 2 and in Python 3 it is code has become much more *pythonic*.

7. Yes, yes of course, for example I came from java and other language that we studied in the grade. Then the data types, and the access to the data was obligatory and here is not that then I tried to replicate those things a bit, I got them and I set them and already after the time I did not. For example with the generators and with the "yield" it does not force you to do the typical "loop for and make a list" but here you can return item to item in a simpler way, then yes, much more.

8. X

9. At the beginning it was much stricter and over time I have noticed that many times the readability worsens and I prefer not to do it.

10. Yes, because I know the syntax better, and I am able to take advantage of the full potential of the language.

11. X

12. Yes, it came from Java. In Java I had to define everything, in Python you have it, in Java you have to generate a super strict data model when in Python you may not need to create a class when in fact you just have to create a dictionary. That is not even an *idiom*, but is a data structure that is there and in Java is not.

**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?**

1. It is desirable but not mandatory. In the code reviews we usually look at those details. Reading code is a very good way to learn and not just from peers for example the “request” library code is a good way to learn how to make a good API and also very *pythonic*.
2. We do peer review and it is one of the things that we usually comment on. In our company programming is our core. It has to be organized. Practices are transferred between programmers of the company.
3. It is required, it is recommended to use certain structures, and reading the code of others you end up acquiring some techniques.
4. It is not, it is more a personal norm. I am trying to maintain and improve the code a lot and I have seen that many structures are not at all *idiomatic* and are redundant. Many terms and concepts are repeated that could be made more *pythonic* with *built-ins*.

Some of my coworkers learn, but we work in different areas, but usually some of them learn. For example some structures are not entirely efficient but are more understandable. For example, reading very large files can be complex and in Python you can do it in four lines.

5.  
A personal choice.

It's sad, but there's no one else in the office that uses Python as its main language. But when it comes to teaching, I often suggest modifications to the student in order to make their code more elegant, or more *pythonic* (without using the term).

I had several opportunity to work on open source software written in Python, and I learnt a lot by looking at how other people wrote Python code. This was very interesting.

6. We do code-reviews and we do not call it *pythonic*, but we say we program python, not assembler, we are going to use what the language provides. Doing an "if" inside a "for", maybe it's a comprehensive list. It is cleaner. Yes we discussed it, not with that term but yes
7. Currently, we only have 2 developers. I check what they do and I try to make their code use more Python structures instead of the typical of other languages and use more generators when they can be used. But I am the one who recommends the rest.
8. (7.)
9. I try to put a lot of emphasis on people writing readable code. Between them there is an evolution and it is seen that they improve their code by looking at it among themselves.
10. Almost mandatory, and I used to learn from my coworkers.
11. It is quite personal and there is no rule about it. We usually compare our code and see what we can improve between us.

12. A partner checks the code before uploading it in a strict manner. I try to ask my coworkers which is the best way.

13. No, it's not, we decide the style and the strategies that we want to follow up. Could be any discussion, but its not normal.

**Do you think that be proficient on *pythonic code* in your repositories or in a job interview could help to get hired?**

1. Yes, it can be. The interviews usually have a limited time and as long as it is not a burden for you to think about which is the most *pythonic way* of doing something and leaving aside the end of the exercise is something very positive and when correcting it you can tell if it is a programmer with experience.

2. It depends on the perception of the other side and sensitivity, it is seen if the person has a certain level and wants to do things right, and that would give points for any position and it would be something positive.

3. I do not have so much experience, but I think it's important.

4. Yes, but that depends on the companies, in a department there is a need and it depends on the manager, if there are many candidates and almost all are equally good, he will choose the people who can contribute the most.

5. I hope

6. I have it in mind and I give it a lot of value.

7. It will probably depend on what they are looking for, obviously if you look at Python's top things it is clear that if it benefits you, in general I think it forces you to think differently and make your code more readable.

8. Yes, I think so, because it shows that you have a higher level.

9. It depends on the person, if you know Python but it is not your main language I will not look at that, but if it is someone who is working in Python and does not take care of those things it would be a rather negative point.

10. Yes

11. It's better than not.

12. Yes, because it gives an idea of what you know about Python, so if you are looking for someone for a specific language the more you know better.

13. X

## **When you discover a new *idiom*... do you adapt your old code? When?**

1. If it is an active project or a project that I am maintaining I would bother doing it.  
If it's a long-standing project, it's likely that I did not know. But if I had to modify it for something and I found it for sure I would change it
2. I'm not looking at all the previous lines, I incorporate it into my toolbox and then when I'm going to touch something I modify it and leave it better, or if I go through it, but it's not an obsession. The *pep8* may be one of the first *idioms*, I always try to make sure that everything is well formatted, but I am more tolerant of there being ifs or loops that are not entirely *pythonic* or things of that style
3. Yes, if I go through that code again. In other case, I would not do it.
4. X
5. It depends on "how ugly the old code look afterwards". If there is no much work and a "great" improvement to readability (or if it may affect my pride), I'll do it nearly immediately. If there's a lot of work, I'll open an issue or add a TODO note.
6. X
7. X
8. X
9. It is more a question of time, if I am adding something to a functionality that I touch, I usually do not return to do it. As I refactorize the code, if I see something that can modify, I do, but usually for lack of time I do not usually do it
10. Yes
11. Normally if it is an active project, I usually adapt the code to include the new *idiom*, if it is a project already done, I prefer not to touch it.
12. X
13. X

## **What IDE do you usually use? Do you have any plugin to make your code more *pythonic*?**

1. Vim and I usually use a style checker
2. X
3. Vim and use *pyflake* and "syntactics?"
4. Normally PyCharm, but to go fast sublime text or notepad. I use a *pep8 plugin*

5. It depends on the task. For small projects, I use Atom. For research and data analysis, I directly use Jupyter Notebook. For medium to large projects, I rely on PyCharm (especially when I've complex test suites or environment, ..)

pylint/pyflakes & co are quite commonly integrated in most IDE and in most text editors. I don't think they really help to write *pythonic* code or to suggest specific *idioms*. I find them useful to identify trailing spaces, undeclared variables (due to a typo, most of the time), or things like that.

6. X

7. Atom + Pylint and pep8

8. PyCharm, no plugins

9. Pycharm and I use tools like pep8 but it's generally for the order (imports, spacing) not so much for data structures. In itertools there are many data structures but if you do not know how to use them you would have to see to what extent you improve the code or not

10. VSCode and plugins (no mention)

11. Emacs + Pylint and pep8

12. Atom + linter

### **What is more important: *idioms* or coding conventions established in a project?**

1. The conventions. If you have chosen them wrong it is an error but it is not good to have mixed code that does not use the same conventions since it does not go anywhere. This is the reason why CamelCase is still allowed in legacy libraries, because if it were mixed, nothing would be understood.

2. The conventions.

3. We are 2, we define the conventions so they are usually related

4. X

5. Both are important because they could ease readability. I think coding conventions are more important, because they are usually defined for specific purposes (meaning there's an added value), while *idioms* are just constructions, and do not necessarily lead to "better code".

Most of the time they typically differ, coding conventions are based on PEP8 (or some variations), or to specific ways to achieve compatibility with Py2/3.

10. The *idioms* are more important

11. The conventions of the project are more important. They can be similar but in each case it depends.

## **Do you think that be proficient on *pythonic* code is desirable in any open source or industrial environment?**

2. It is much better than non-Python alternatives. It is more readable, exposes all bugs. One of the objectives is that it works well and is maintainable. I do not believe that the contributions increase. In a company I was in, my boss said he did not want "odd python", because that could put a barrier to collaborate. Maybe he was referring to metaclasses or something hard to understand. For example a "with" instead of an "if" is *pythonic* code, but metaclasses maybe not. But it would help.

5. Yes, of course, because that's a kind of expertise/skill.

6. It is much more readable, if you always do in the same way is more predictable to read, to write, you need to think less.

10. Yes

## **Did you understand when you began to learn Python most of the *idioms*?**

3. No, you spend some time to understand them and then you learn when to use them and when not.

4. It cost me a lot at first. In scripting I used list comprehensions, later in my study period I did not use them, and when I started working I started to use them a lot, I learned many structures that I did not know, and I had a hard time understanding them but once you understand them it is very easy to know when to use them .

For example implementing decorators is complex, I have implemented some but I do not usually implement them but I use them when necessary.

6. If you have not seen it in another language, yes, but not complicated either. Nothing in Python is complex. Even though it is true that the last addition of the concurrency is complex (how to use events or async).

8. Yes, at first yes, but later I learned and improved.

9. It depends, the comprehension lists are quite intuitive, the decorators are more complicated. The lambda functions I try to avoid them, I prefer to use an external function.

10. I did not understand them, but I was interested in the documentation and a world opened up for me. It took me a while to take advantage of it.

11. For example the decorators cost me a bit at the beginning, the list comprehension not so much. Understanding it is easier than implementing it.

12. The language is simple, but it is complicated to adapt when you come from a strict language (Java)

13. For example the *list comprehension* are easy to understand and to write, but for me was hard to see why were important.

The *decorators* for example are different, are *idioms*, but are structures focused on metaprogramming.

The magic methods were also hard to understand what was the real necessity. Because you can write a method that can do the same, but with the time I understood that make an interface more homogeneous and the programmer that use that interfaces doesn't have to know the specific method. It is hard to classify all the *idioms* for readability, performance and standards.

### **Is a more advanced Python programmer who uses Python *idioms*?**

5. I'll say the converse: someone who uses *idioms* is probably a more advanced programmer (but I know advanced programmers that will write Python code as they write C code, for instance).

*Pythonic* and *pythonic idioms* are different things. *Pythonic* can be used to measure a developer's skills, *idioms* can be used to (at least) measure a developer's knowledge.

6. No, it's just one aspect but there are others. You can write in code that is unstructured with *idioms* but difficult to maintain, the *idioms* are just one of the aspects

7. Yes, of course, for example, a problem that we programmers may have is that we try to replicate what we learn in a language in another language, for example, if I use java and program in java, then I try to do the same in Python that I do in java , for example get and set and that kind of methods to access the properties of an object, because in python, for example, that does not make sense because there is no data protection, there is visibility at all times and you can access everything, and implementing that in Python does not make sense, there are other ways to do it. For that reason I think that it is important and we must adapt to how it is programmed in a language and not try to replicate other languages and use the *idioms* of the language.

10. It is desirable to improve the level of Python, but a developer is not more advanced if he uses *idioms*.

11. In Python yes

12. I suppose so, because in the end the more *idioms* you know the more time you have been programming in that language. The experience gives you that, know how to do certain tasks in a simple way and know how to understand them. But they do not always make them legible.