The importance of consistency in your Python code  
Python代码前后一致的重要性  
======  
This is part of a special series about the Zen of Python focusing on the  
12th, 13th, and 14th principles: the roles of ambiguity and obviousness.  
这是有关Python Zen的一个特别系列的一部分，该系列着重于第12、13和14条原则：歧义和明显性的作用。  
这部分是有关 Python Zen 聚焦在第 12,13,14 规范：角色的二义和明确  
![Two animated computers waving one missing an arm][1]  
！[两台动画计算机挥舞着一只手臂失踪] [1]  
The principle of least surprise is [a guideline][2] when designing user interfaces. It says that when the user performs an action, the program should do whatever would surprise the user the least. This is for the same reason kids love reading the same book over and over again: there is nothing more comforting to people than the ability to predict and have those predictions come true.  
  
A critical insight in the development of the [ABC language][3], Python's inspiration, was that programming languages are user interfaces and need to be designed with the same tools that UI designers use. Thankfully, since then, more languages have adopted the concepts of affordance and ergonomics from UI design, even if they apply them less strictly.  
  
This brings us to the next three principles in the [Zen of Python][4].  
  
### In the face of ambiguity, refuse the temptation to guess.  
  
What should the result of **\*\*1 + "1"\*\*** be? Both **\*\*"11"\*\*** and **\*\*2\*\*** would be valid guesses. This expression is \_ambiguous\_: there is no single thing it can do that would not be a surprise to at least some people.  
  
Some languages choose to guess. In JavaScript, the result is **\*\*"11"\*\***. In Perl, the result is **\*\*2\*\***. In C, naturally, the result is the empty string. In the face of ambiguity, JavaScript, Perl, and C all guess.  
  
In Python, this raises a **\*\*TypeError\*\***: an error that is not silent. It is atypical to catch **\*\*TypeError\*\***: it will usually terminate the program or at least the current task (for example, in most web frameworks, it will terminate the handling of the current request).  
  
Python refuses to guess what **\*\*1 + "1"\*\*** means. The programmer is forced to write code with clear intention: either **\*\*1 + int("1")\*\***, which would be **\*\*2\*\***; or **\*\*str(1) + "1"\*\***, which would be **\*\*"11"\*\***; or **\*\*"1"[1:]\*\***, which would be an empty string. By refusing to guess, Python makes programs more predictable.  
  
### There should be one—and preferably only one—obvious way to do it.  
  
Prediction also goes the other way. Given a task, can you predict the code that will be written to achieve it? It is impossible, of course, to predict perfectly. Programming, after all, is a creative task.  
  
However, there is no reason to intentionally provide multiple, redundant ways to achieve the same thing. There is a sense in which some solutions are "better" or "more Pythonic."  
  
Part of the appreciation for the Pythonic aesthetic is that it is OK to have healthy debates about which solution is better. It is even OK to disagree and keep programming. It is even OK to agree to disagree for the sake of harmony. But beneath it all, there has to be a feeling that, eventually, the right solution will come to light. There must be the hope that eventually we can live in true harmony by agreeing on the best way to achieve a goal.  
  
### Although that way may not be obvious at first (unless you're Dutch).  
  
This is an important caveat: It is often \_not\_ obvious, at first, what is the best way to achieve a task. Ideas are evolving. \_Python\_ is evolving. The best way to read a file block-by-block is, probably, to wait until Python 3.8 and use the [walrus operator][5].  
  
This common task, reading a file block-by-block, did not have a "single best way to do it" for almost \_30 years\_ of Python's existence.  
  
When I started using Python in 1998 with Python 1.5.2, there was no single best way to read a file line-by-line. For many years, the best way to know if a dictionary had a key was to use **\*\*.haskey\*\***, until the **\*\*in\*\*** operator became the best way.  
  
It is only by appreciating that sometimes, finding the one (and only one) way of achieving a goal can take 30 years of trying out alternatives that Python can keep aiming to find those ways. This view of history, where 30 years is an acceptable time for something to take, often feels foreign to people in the United States, when the country has existed for just over 200 years.  
  
The Dutch, whether it's Python creator [Guido van Rossum][6] or famous computer scientist [Edsger W. Dijkstra][7], have a different worldview according to this part of the Zen of Python. A certain European appreciation for time is essential to appreciate it.