Mitch Negus
9/25/2017

Treating Code as an Essay

Matsumoto, Y.

Pythons Dictionary Implementation: Being All Things to All People

Kuchling, A.

Summary

This week's articles focused on writing elegant and readable code. While the first article by Matsumoto provided a more in-depth discussion of how this goal can be achieved for a practicing programmer, the second described a tool which Python uses to facilitate elegant coding. Matsumoto mentions that code will almost certainly need to be debugged eventually, and even if not, it will to be updated to remain compatible with newer technological advances. If a developer must eventually return to the code, then it also must be written in a way that they do not need to spend a prohibitively long time attempting to decipher the code. In this sense, Matsumoto argues that is far more important to write code that can be understood by humans, than it is code that is optimized for a computer. The computer will manage. Along these lines, another point Matsumoto makes is that code ought to be as simple as possible. This way code can be both visually scanned by a developer attempting to understand it, but also to minimize redundancy which will require extensive effort to be properly maintained. one way that this simplicity can be achieved is through providing programmers flexibility in how the write programs. Python dictionaries, as described by Kuchling, offer this flexibility. They can take in various datatypes and they can be iterated over or called by key, among other functionality.

Exploration

While I agree with Matsumoto's assessment that code should generally be kept as simple as possible, I think there are times where this rule must be stretched. For instance, in large code projects, the principles of abstraction suggest that code ought to be separated into manageable chunks. While this does reduce redundancy, it limits code readability to some degree. Like a paper with many references, it can be tedious to trace through each function call or class instance (as compared to a script where everything is presented at once). That being said, I would say the benefits of abstraction for maintaining code consistency are worth the loss of simplicity. What is lost in clarity is more than accounted for by reducing redundant code, or enabling rigorous unit testing.

Notes

——Code as Essay——
- code will most likely need to be rewritten (debugged, updated, etc.); people must be able to read the code or else these functions will not be realized (even the best computer can't read code that hasn't been updated properly) - computers can deal with complexity; people can't
- concise is important in writing (don't waste words); similarly, brevity is important in code (be to the point)
    -allows scanning of code by eye to find utility
    -redundancy reduces cost of maintaining consistency (do it once, fix it once)
- code should be kept as simple as possible (I agree with this statement, though I think there comes a limit where simplicity and functionality converge)
    $\rightarrow$ classes are significantly more complex than functions or simple scripts, but are drastically more powerful; know when to use both
    $\rightarrow$ avoiding redundancy can lead to complicated software structures; simplicity should be balanced against abstraction

——Python Dictionaries——
- explains dictionary usage (multiple types; assignments/lookups, etc.)
- interesting to see how dictionaries are used as the underlying framework on which classes are built