**3 Powerful Reasons You Should Plan Your Python Code Before You Write It**

Planning out your code matters a lot more than you think.



Photo by [UX Indonesia](https://unsplash.com/@uxindo?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral" \t "_blank)

If you’re an avid Python user, you may be familiar with the famous poem, “The Zen of Python.” It was written by Tim Peters (one of the original developers of the language), and can be read in its entirety by typing the command import this into the Python interpreter.

I won’t inundate you with the whole poem, but if you’re curious you can [check out my previous article on the topic](https://towardsdatascience.com/4-simple-tips-from-pythons-inventor-to-improve-your-code-5429297505a9" \t "_blank).

For the purposes of this article, I want to draw your attention to the following two lines from the poem:

*If the implementation is hard to explain, it’s a bad idea.  
If the implementation is easy to explain, it may be a good idea.*

The words that stand out to most people are “implementation,” “hard,” “easy,” “bad,” and “good.” But I would argue that the most significant and impactful word in this couplet is something else entirely: *explain*.

The ability to explain your code ***before you write it*** is an incredibly underrated skill, especially overlooked during the early stages of learning programming. Back when I was an undergraduate, there were large numbers of students who programmed well but couldn’t write a proper sentence to save their lives — and the worst part of it all was that they felt this to be completely acceptable.

Making a habit of mapping out your code will only benefit your career in the long run, regardless of if you want to remain a programmer or move into management-style roles.

With that in mind, let’s take a look at three powerful reasons you should learn to explain your code, and explain it well.

**1. As a data scientist, you’ll gain a leg up in the workplace**

If you learn computer science through a traditional academic program, by the third or fourth class, you’re generally required to submit **design documents** along with your projects.

This is a detailed document which usually doesn’t have any code, but instead consists of a high-level description of the code you will eventually write. More specifically, a good design document has one or more of the following:

* An overview of the eventual system’s overall goals
* Multiple maps and charts detailing how different parts of the program will fit together
* Fairly detailed pseudocode which describes the data structures and algorithms you will use, but stops short of showing actual code.

Students are required to do this for good reason: software engineers in the wild pretty much always write a design document before they begin coding.

Unfortunately, this is an aspect of programming you may be unaware of as a data scientist. Since [data scientists tend to use programming as a tool](https://towardsdatascience.com/the-three-building-blocks-of-data-science-2923dc8c2d78) (rather than the end-all be-all of their work), most either learned it informally or in classes that weren’t heavily focused on traditional software engineering.

If you, a data scientist, can learn to do plan your code well, you can gain a meaningful leg up in your career. For multiple reasons (as we’ll see below), effective planning improves eventual code quality.

Being a competitive candidate for employment comes down to one simple question: can you offer an in-demand skill set that is hard to come by?

Software engineers can design systems well, but most don’t have the expertise to be data scientists. Data scientists know their statistics, but often have limited programming abilities.

Master both. Become desirable.

**2. You’ll avoid extremely annoying bugs down the line**

Earlier this year, I was a TA for an introductory data science class at my university. It was designed for students who had a bit of programming experience, but still hadn’t worked with complex data structures or larger programs.

At the end of the class, students had to complete a final project of their own choosing, which consisted primarily of a programmatic analysis of a data set that interested them. As part of their initial proposal, they needed to submit a rough breakdown of the time they expected to spend on each part of the project (gathering data, coding, writing final report, etc.).

Seeing the results was admittedly a bit amusing. Many of the students hadn’t written large programs before, and naively stated that they expected to spend only 6–8 hours writing *all* the code. I warned them that this was a serious underestimate.

How did I know this? Well, I (and many colleagues) have worked on many a project where it took upwards of 8 hours just to fix a *single* bug.

Although syntactical bugs like this are often unavoidable (e.g. a missing character somewhere that causes a miscalculation but no error), there are many cases in which they can be avoided simply by planning a little bit better. These are bugs caused by logical errors.

Put simply, algorithms are often complex, and if you try and write them into code directly, it’s likely you’ll unknowingly mess up somewhere along the way. And then you’ll go through hell trying to figure out what went wrong when your final program doesn’t work.

Alternatively, if you first map out your algorithm in human language, you can ensure each part is correct *before* writing any actual code. Then, getting your program to work becomes a simple matter of translation.

This process may seem like extra work, but in the long run, it’s just making your life easier. Trust me.

**3. You’ll develop more maintainable and extensible code**

This is a big one. Many folks new to programming often fail to realize a very important fact: your code needs to make sense to other people.

And not just your coworkers either. If it was just them you had to satisfy, you could potentially get away with needlessly complex code, since you could personally explain it to them. I’m not saying this is good or recommended, but it is possible.

However, it’s unlikely you’ll stay at one job forever — but while you can leave, the program must stay. There will come a day when someone pulls out code that you wrote ages ago, so lost to your memory that even you wouldn’t recognize it.

When that day comes, one of two things will happen:

1. They’ll spend hours upon hours parsing an unintelligible mess of variables, keywords, and functions.
2. They’ll read the accompanying design document and quickly comprehend what the code is meant to do as a whole and how its various pieces work together.

By putting in the effort to write an effective design document during the early stages of a project, you immediately set the stage for others to easily maintain and extend your code going forward. In one fell swoop, you’ve benefited not only yourself, but other programmers and the company as a whole.

Need I say more?

**Final Thoughts + Recap**

If you have a habit of straight-up coding without any planning, it’s time to make a change. Here’s a quick reminder of the reasons for doing so:

1. **Career Points**. If you’re a data scientist, mastering the skill of properly planning out your code will give you a serious leg up among your colleagues.
2. **Peace of Mind**. By designing and checking for accuracy in the early stages — when your data structures and algorithms are still in human language — you’ll reduce the chance of running into annoying bugs later on.
3. **Benefits All Around**. Good planning = good code. This in turn translates to happy coworkers, happy future workers, and a happy company. Write a design doc. Spread happiness.

Best of luck on your programming endeavors!