Refactoring

Refactoring is when you take existing code that gets the job done, and you improve the quality, readability, or idiomaticness of your code without any noticable improvement for the user. According to Martin Fowler, who more or less invented the term it means:

> Refactoring (noun): A change made to the internal structure of software to make it easier to understand and cheaper to modify with-out changing its observable behavior.

It's a term that is commonly misused. I often hear other programmers say that they are going to refactor something to fix bugs or improve performance. This is not what we are talking about.

Refactoring is a favor to your future self reading code from ages ago or a collaborator that you want to give the benefit of readable, modular, easily changed code. I like to think of it as user experience for the next person dealing with your code. You want to make things easy and painless down the road. After you get used to doing it every now and again, you will become skilled at finding 'code smells' and you will know how to fix them. Explaining to your boss why you just spent a day coding and deploying software but there isn't any observable change in the output is more of a challenge.

When you join a team and you have to learn the ropes of a new code base, I find that the best way to introduce yourself to the new code is to step through and refactor away the complex stuff. You will know what does what, and everyone else will have cleaner code to work with.

Some pointers for starting to refactor

First look for smells:

- Squint test
- Excessive comments
- Long parameter lists
- You start reading a method or a class and by the time you are at the end, you forget the beginning
- Long method
- A class that knows way too much
- Methods that are not reusable
- Inflexible -- expensive to update or expand upon
- Local variables inside methods

For all of these symptoms, there are cures. We will discuss a couple today.

We are going to take a horribly designed class, do a little of refactoring and a little enhancing once it's nice and modular, i.e. easy to test and work with. After that, you can go through the extra example and get some reps in yourself.

Practical Intro

Keep in mind that since I am not your normal instructor I don't know everything that you have gone over already, so *please* tell me if I show you something that you haven't seen yet!

If you get a little behind, don't worry, look at the other branches of this repo to see my solutions for each step in the lesson.

Help is out there

Though I personally prefer against them, IDEs such as Ruby Mine and PyCharm from JetBrains offer built in advice such as pointing out bits that should be refactored.

For Ruby, there is a gem called Rubocop which studies your code and points out uncleanliness. It's based on the widely followed Ruby Style Guide. Unlike Python, Ruby lacks a canonical set of rules regarding coding style, and some companies have their own style guides. It's a bit strict, but Rubocop provides food for thought when refactoring your code. Until your nose becomes adept at spotting code smells, you might have some luck with the Reek gem.

Some great books:

- POODR by Sandi Metz
- Refactoring -- Ruby Edition
- Ruby Science by Thoughtbot