**Clean and Modular Code**

* **PRODUCTION CODE:** software running on production servers to handle live users and data of the intended audience. Note this is different from *production quality code*, which describes code that meets expectations in reliability, efficiency, etc., for production. Ideally, all code in production meets these expectations, but this is not always the case.
* **CLEAN:** readable, simple, and concise. A characteristic of production quality code that is crucial for collaboration and maintainability in software development.
* **MODULAR:** logically broken up into functions and modules. Also an important characteristic of production quality code that makes your code more organized, efficient, and reusable.
* **MODULE:** a file. Modules allow code to be reused by encapsulating them into files that can be imported into other files.

# Refactoring Code

* **REFACTORING:** restructuring your code to improve its internal structure, without changing its external functionality. This gives you a chance to clean and modularize your program after you've got it working.
* Since it isn't easy to write your best code while you're still trying to just get it working, allocating time to do this is essential to producing high quality code. Despite the initial time and effort required, this really pays off by speeding up your development time in the long run.
* You become a much stronger programmer when you're constantly looking to improve your code. The more you refactor, the easier it will be to structure and write good code the first time.

# Writing Clean Code: Meaningful Names

#### Tip: Use meaningful names

* ***Be descriptive and imply type****- E.g. for booleans, you can prefix with is\_ or has\_ to make it clear it is a condition. You can also use part of speech to imply types, like verbs for functions and nouns for variables.*
* ***Be consistent but clearly differentiate****- E.g. age\_list and age is easier to differentiate than ages and age.*
* ***Avoid abbreviations and especially single letters****- (Exception: counters and common math variables) Choosing when these exceptions can be made can be determined based on the audience for your code. If you work with other data scientists, certain variables may be common knowledge. While if you work with full stack engineers, it might be necessary to provide more descriptive names in these cases as well.*
* ***Long names != descriptive names****- You should be descriptive, but only with relevant information. E.g. good functions names describe what they do well without including details about implementation or highly specific uses.*

Try testing how effective your names are by asking a fellow programmer to guess the purpose of a function or variable based on its name, without looking at your code. Coming up with meaningful names often requires effort to get right.

# Writing Clean Code: Nice Whitespace

#### Tip: Use whitespace properly

* *Organize your code with consistent indentation - the standard is to use 4 spaces for each indent. You can make this a default in your text editor.*
* *Separate sections with blank lines to keep your code well organized and readable.*
* *Try to limit your lines to around 79 characters, which is the guideline given in the PEP 8 style guide. In many good text editors, there is a setting to display a subtle line that indicates where the 79 character limit is.*

For more guidelines, check out the code layout section of PEP 8 in the notes below.

# Writing Clean Code: Nice Whitespace

[PEP 8 guidelines for code layout](https://www.python.org/dev/peps/pep-0008/?#code-lay-out)