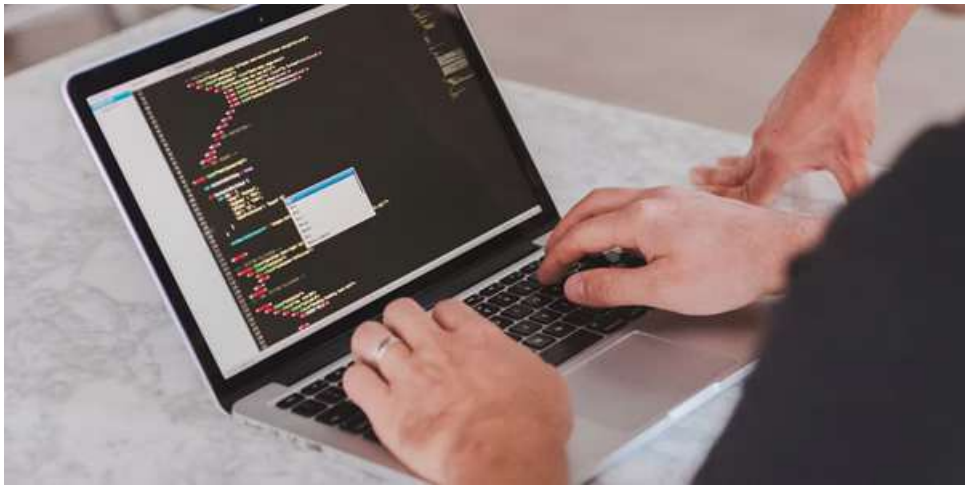
10 Basic Programming Principles Every Programmer Must Know

https://www.makeuseof.com/tag/basic-programming-principles/?utm_medium=newsletter&utm_campaign=MUO-202311180700&utm_source=MUO-NL

Your code should be clear and easy to maintain. Here are several other programming principles to help you clean up your act.



Key Takeaways

- Keep It Simple, Stupid (KISS): Write code that is as simple as possible. Avoid being overly clever and use clear variable names.
- Write DRY Code: Avoid duplicating code; make use of functions, loops, and algorithms instead. DRY code is easier to maintain and debug.
- Open/Closed: Write code that is open to extension but closed to modification. Prevent direct modification and encourage extension for more stable and maintainable code.

It's easy to write code yet challenging to write good code. Embracing basic programming principles is a surefire way to write high-quality code that is efficient, readable, reliable, secure, and maintainable, regardless of the size of a software project.

Bad code comes in many forms: messy, massive if-else chains, unreliable programs, variables that don't make sense, etc. How do you write effective code? With discipline and purpose. Here are the core programming principles that will make you a better coder.

1. Keep It Simple, Stupid (KISS)

It sounds a little harsh, but it's one of the most important principles to adopt when you're writing computer code. What does KISS mean?

It means you should be writing code as simple as possible. One of the rules of basic programming is to never get caught up in trying to be overly clever or showing off with a thick block of advanced code. If you can write a script in one line, write it in one line.

Here's a simple function:

```
function addNumbers(num1, num2) {
    return num1 + num2;
}
```

Pretty simple. It's easy to read, and you know exactly what is going on.

One programming principle in this spirit is to use clear variable names. Take advantage of coding libraries and use existing tools. Make it easy to come back after six months and get right back to work. Keeping things simple will save you so much needless suffering down the line.

2. Write DRY Code

The Don't Repeat Yourself (DRY) computer programming principle means, plainly, not repeating code. It's a common coding mistake. When writing code, avoid duplication of data or logic. If you've ever copied and pasted code within your program, it's not DRY code.

Take a look at this script:

```
function addNumberSequence(number) {
    number = number + 1;
    number = number + 2;
    number = number + 3;
    number = number + 4;
    number = number + 5;
    return number;
}
```

Instead of duplicating lines, try to find an algorithm that uses a loop instead.

DRY code is easy to maintain. It's easier to debug one loop that handles 50 repetitions than 50 blocks of code that handle one repetition each.

3. Open/Closed

This principle of programming means that you should aim to make your code open to extension but closed to modification. It ensures that you create code that doesn't need to be modified even when requirements change. This is an important principle when releasing a library or framework that others will use.

For example, suppose you're maintaining a GUI framework. You could release a version for coders to modify and integrate your released code directly. What happens when you release a major update four months later, though?

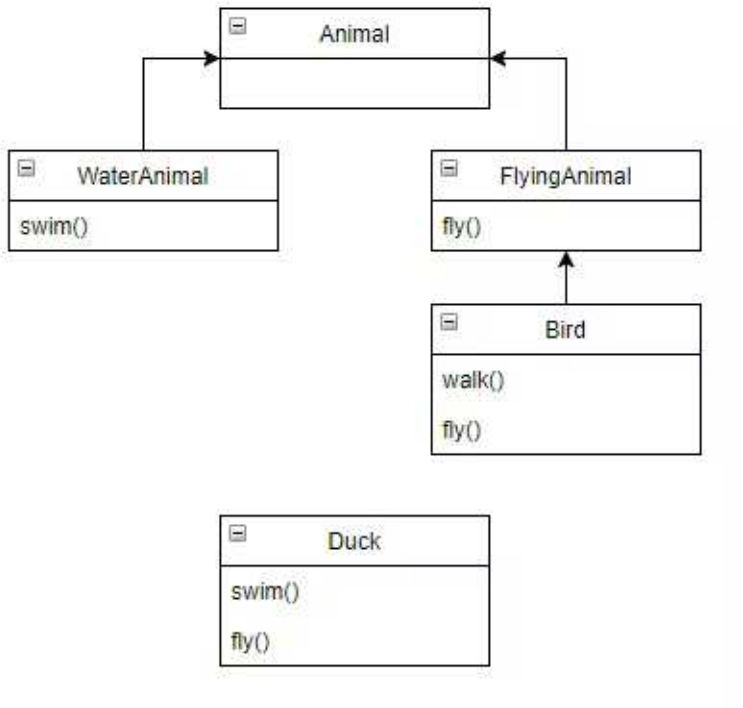
Their code will break. This will likely make your cohorts very unhappy. They won't want to use your library for much longer, no matter how helpful it may have been in its heyday.

Instead, release code that prevents direct modification and encourages extension. Basic programming principles like this separate core behavior from modified behavior. The code is more stable and easier to maintain.

4. Composition Over Inheritance

If you write code using object-oriented programming, you're going to find this principle of programming to be very useful. The composition over inheritance principle states that objects with complex behaviors should contain instances of objects with individual behaviors. They should not inherit a class and add new behaviors.

Relying on inheritance causes two major issues. First, the inheritance hierarchy can get messy in a hurry. You also have less flexibility in defining special-case behaviors. Let's say you want to implement behaviors to share:



Composition programming is cleaner to write, easier to maintain, and allows for flexibility-defining behaviors. Each individual behavior is its own class. You can create complex behaviors by combining individual behaviors.

5. Single Responsibility

The single responsibility principle states that every class or module in a program should only provide one specific functionality. As Robert C. Martin puts it, "A class should have only one reason to change."

Classes and modules often start off this way. Be careful not to add too many responsibilities as classes get more complicated. Refactor and break them up into smaller classes and modules.

The consequence of overloading classes is twofold. First, it complicates debugging when you're trying to isolate a certain module for troubleshooting. Second, it becomes more difficult to create additional functionality for a specific module. Good programming principles prevent these problems before they become problems to deal with.

6. Separation of Concerns

The separation of concerns concept is an abstract version of the single responsibility principle. This idea states that you should design programs with different parts, and these parts should not have access to each other.

A well-known example of this is the model-view-controller (MVC) design. MVC separates a program into three distinct areas: the data (model), the logic (controller), and what the page displays (view). Variations of MVC are common in today's most popular web frameworks.

For example, the code that handles the database doesn't need to know how to render the data in the browser. The rendering code takes input from the user, but the logic code handles the processing. Each piece of code is completely independent.

The result is code that is easy to debug. If you ever need to rewrite the rendering code, you can do so without worrying about how the data gets saved or the logic gets processed.

7. You Aren't Going to Need It (YAGNI)

This principle means you should never code for functionality on the off chance that you may need something in the future. One of the most important principles of computer programming to learn is that you shouldn't try to solve a problem that doesn't exist.

In an effort to write DRY code, programmers may violate this principle. Often, inexperienced programmers try to write the most abstract and generic code that they can. Too much abstraction, however, causes bloated code that's impossible to maintain.

Only apply DRY programming principles when you need to; if you notice chunks of code written over and over, implement a layer of abstraction. Don't think too far ahead at the expense of your current code batch.

8. Document Your Code

With all of this talk about the principles of coding, it can be easy to forget about the human on the other side who may eventually be getting into your code themselves.

Any senior developer will stress the importance of documenting your code with proper comments. All languages offer them; you should make it a habit to write them. Leave comments to explain objects, enhance variable definitions, and make functions easier to understand.

Here's a JavaScript function with comments guiding you through the code:

```
 // This function will add 5 to the input if odd, or return the number if even
function evenOrOdd(number) {
    // Determine if the number is even
    if (number % 2 == 0) {
        return number;
    }
    // If the number is odd, this will add 5 and return
    else {
        return number + 5;
    }
}
```

Leaving comments is a little more work while you're coding. It takes time and steals your attention away from the real work at hand. You understand your code pretty well anyway, right? Who cares? It's worth remembering that nothing is disposable, even in the world of tech. What is a computer programming principle at the end of the day if the person on the other side ends up getting lost?

We recommend going the extra mile and leaving comments anywhere you worry that things will become murky or unclear, especially when collaborating with others. Don't frustrate your fellow developers by forcing them to decipher your syntax.

Try writing a program, leaving it alone for six months, and returning to modify it. You'll be glad you documented your program instead of having to pour over every function to remember how it works.

9. Refactor

It's hard to accept, but your code isn't going to be perfect the first time. Refactoring code means reviewing your code and looking for ways to optimize it, making it more efficient while keeping the results exactly the same. This is a consideration for writing cleaner and quality code.

Codebases are constantly evolving. One of the principles of programming is remembering that it's completely normal to revisit, rewrite, or even redesign entire chunks of code.

It doesn't mean you didn't succeed the first time you wrote your program; you're inevitably going to get more familiar with a project over time. Use that knowledge to adjust yourself as you make progress.

10. Clean Code at All Costs

Aside from all the fundamental programming principles, leave your ego at the door and forget about writing clever code. When we say this, we mean the kind of code that looks more like a riddle than a solution. You're not coding to impress strangers. You're in this profession to solve problems.

Don't try to pack a ton of logic into one line. Leave clear instructions in your comments and documentation. If your code is easy to read, it will also usually be easy to maintain.

Good programmers and readable code go hand-in-hand. Leave comments when necessary, adhere to style guides, and put yourself in the next guy's shoes whenever possible.

Learn the Principles of Computer Programming to Be a Good Programmer

Learning how to be a good programmer takes quite a bit of time and effort. These rules of basic programming are a roadmap to becoming a professional programmer. By following these time-honored principles, you'll set yourself up for success in your future programming career.