Five Rules for Writing Good Code :: http://yanpritzker.com/2009/09/29/five-rules-for-writing-good-code/

1. Write for an audience

Code tends to outlive jobs. When we create code, we are not writing for ourselves, but for an audience of peers and progeny that will look upon it and have to maintain it. Do we want them to do so with awe and respect, or with fear and disgust? Writing maintainable code not only makes our life easier, but the lives of those around us, and garners admiration, praise, and rewards – if not always financial, at the very least karmic.

2. Establish a clean framework for future changes

If we start writing new code, and the pattern we establish is that we've copied and pasted a line across ten functions, what will happen when someone else comes along to add an eleventh function? Let's face it, even the neatest programmers can get lazy. If something has been copy-pasted ten times, it will get copy-pasted for the 11th time, and a year later, when we find out we have to change the logic or content of that line, we are now changing 50 lines where we could have had one. Untangling code is always a lot harder than writing it, and we could have prevented the spread of the copy-paste disease by evaluating our code initially for repetitive statements. Always establish a framework for others to follow by writing clean code that reduces repetition, and encourages maintainability by isolating each piece of logic and content to one spot only.

3. Be brief, self-descriptive, and avoid inline comments

Five line functions with descriptive names are easy to understand. They require no comments. Thirty line functions take quite a bit of brainpower to digest, and usually have smelly comments scattered all throughout trying to explain bits of the function. Hundred line functions are an assault on all that is holy and stink to high heaven. If we see lots of comments interspersed in a method, it is a good sign that the code cannot be easily understood. Break it down into lots of little functions with descriptive names, and all of a sudden our code reads like a very clear and concise recipe, and we find that the comments become redundant. Most comments that live inside functions are parasitic organisms, treat them with suspicion.

4. Follow language standards and community conventions

If we break conventions, the next person to read our code will wonder why we did so. When our code raises questions about its style, the reader may start wondering if there was some specific reason that we coded it that way. Worse, they may perpetuate our unusual style through imitation and copy-paste tactics. This makes code harder to understand and wastes other people's time. I've heard an argument from more than one person that they will code with the style they are comfortable with rather than follow language convention. This is especially true of people migrating from one language to another without investing their time in learning new habits and techniques. Remember that most of the time, we are writing code that someone else will be maintaining. Make their job easier by conforming to widely acknowledged standards, so they don't have to spend any extra mental energy reading through it. It may take we a larger effort initially, but it will pay off with dividends when it comes to maintenance.

5. Really learn the language and the framework

Lots of bad code is written because of language or framework ignorance. If we don't know the framework we use, we might reinvent the wheel or write obtuse code because we're not taking advantage of the conventions and helpers already created for us. No need to be a walking encyclopedia, but remember to occasionally open up that encyclopedia and read through it so that you know at least what's out there. I am sometimes surprised by new things I find in a framework I've used for quite a while, that makes my life a whole lot easier. Don't neglect the docs, and don't neglect to keep up with blogs that discuss new techniques.

Extra. Why we do some code and How ? Hacernos esta pregunta sire para evaluarnos y motivarnos.
Si estamos aprendiendo y programando en código abierto, como con Openframeworks, somos parte de un desarrollo colectivo, que gracias a las metas comunes cada día se está mejorando y poniendo en practica aquello que estamos co–creando.
Cuando compartimos nuestros avances hacemos que este entorno siga creciendo, si mantenemos la línea y la mejoramos estaremos compartiendo nuestro avances, motivando al grupo y a nosotros mismos.