Jenny’s Rules of Readable Code *(Page:1/2)*

1. Why this matters?

With a large project, its easy to get out of control, with unreadable code. After each short deadline, each ‘ehh it works’, after every single mistake. The code begins to become a mess.

2. Rules

2.1. No unnecessary comments

Its becoming somewhat of a meme within programming spaces to create unnecessary comments that take up more space then the code itself even does. An example of a bad comment is “x +=2 // this adds x with 2, this allows us to get what x + 2 is,” Is a bad comment as it has no purpose, any reasonable programmer will be able to understand what such is doing. So, there’s no point in adding further context. An example of a good comment is “int bar = foo.example(); // opens a window and plays a small sound effect,” This is a good comment due to it giving further context then is visible in the first place.

2.2. If using classes the names must mean something

In OOP programming, the classes are intended to mean something to the programmer. If you have a class named ‘tool’ or ‘utilitys’, it doesn’t give any further context, making it unreasonable and often uncontrollable.

2.3. Functions must not be bigger then a page if printed (~60 lines max)

This is simply a thing of organisation and re-usability. If a function has too much logic, odds are it will need to be used again. So, it makes more scene to spilt the function into smaller parts.

2.4. Never assume safe input

Assuming a safe input will always cause exploits. This happens as its likely its not the first time the function is used, assuming that any checks are being used at all. So, an example of a good start to a function is “void test(int foo, float bar){ if(foo == 0){ThrowError(“null error”);} if(bar == 0) {ThrowError(“null error”);} },” This example shows a function able to take broken inputs. Every function should follow this rule, even if the input has been checked beforehand. Just in case this function is reused, it would be beneficial to make sure every function can independently take broken inputs.

3. Conclusion

Overall, if these rules are followed it will create code that can by any reasonable programmer be understood and added too.