Good Code Primer Group 1  
Katy Fleming, Jonathan Vanderah, Matthew Wilder

When looking at each of our values into what looks like good coding, we realized that we have many instances where we feel similarly. Several of the topics that the majority of the group felt similarly on were:

**Variables:**  
We felt like as a group that using names or variables that are appropriate for the situation/program were very important. If you use vague variable names such as letters or words that don’t relate to the situation, you could run into a problem where a person reading your code can't figure out what that code means. We also thought of some situations which contain too long of variable names, and where you have a lot of text that you have to type over and over again and that can get tedious.

We also felt like when discussing variables, you have to be careful that you are being consistent in your usage of either capitalization in your variables to represent spacing, or that you should use underscores to represent spacing. You should not use both in one program, and should stick with which option you feel most comfortable with. Choosing to use both in a single program could confuse someone else that is looking at your program.

**Commenting:**  
We felt that when commenting, you have to use the right amount, and can both have too much or too little commenting. If you have comments that are too long, then it can make your program look more complicated. Commenting should also only be used for larger programs, as with very short ones, you can get away with not having comments, because it is already very easy to understand shorter code.

For commenting in code, it is also useful to not get overly descriptive. You don’t necessarily let the person know every single step, as they should be at least able to figure out the basics. Commenting helps for more complicated bits of code. It also helps in long code if you can explain where one step leaves off and the next begins.

**Redundancies:**  
We felt that redundancies were something that was also apparent in code, and could be a problem depending on the program. What we discussed in our group was to not get too complicated with your program. To make a code less complicated, you can make loops or use functions instead of writing out code several times. Not only does this make the code look a lot less complicated, but it also helps out the person writing the code so they don’t have to write out code when it is repeating itself several times within the program. Always look at the steps within the program, and if some parts could be copied, then those parts could possibly be made as a function instead.
Spacing:
When considering good good code, spacing is a factor that might not have to do with the code itself, but is important when considering the person reading the code after you have created it, and the understandability of your code.

Too much spacing can make code look longer than it is supposed to, and hard to figure out which pieces work together. Not enough spacing can make your code look too busy, and can be difficult to see where one step ends and the next step begins. Think of a college paper that has spacing in between the paragraphs, you want to show what steps of the code go together, and so not put spaces in between those sections of code, and put spaces in between each time a step changes. These can change depending on the program and how complicated it is or how many steps it contains.

Differences:
Make sure that all of your code looks similar. Depending on the program that you are using, code can become color coded depending on the functions and parameters inside the coding program. Because each coding program has different colors that they use to identify certain parts of code, there is no one set rule for which colors to use, the objective here is to be consistent in the parts of code that you can control. Minimize the differences that you can control in your code, and be as consistent as possible.

For example, in some coding programs there are different ways that you can comment, whether by using the pound sign or by using apostrophe and quotation marks. When you use each of these symbols to comment out your text, the text changes color depending on the program that you are using to show the code in. For the program Idle, the pound sign comments are bright red and the quotation comments are green. Now, other parts of the code can show up in this same green color, specifically if you are using formatting, so if you want to make your comments more visible and less like other parts of the code, you would use comments that are commented out using the pound sign.

If you have a program that doesn’t have a certain text color, it doesn’t matter as much if your comments are a certain color, but can make a difference when you are faced with a program that needs these changes.

Conditionals:
When using conditionals in your code, you also should be aware of not being too broad when you are using ‘if’ statements. Make sure that you are using not only less than and greater than signs, but check if your program also needs to be equal to a certain number, changing it from less than to less than or equal to. Also making sure that you are using boolean statements correctly within the ‘if’ statement is essential for not only the code to be understood, but also to be properly work.

Within ‘if’ statements, you can make several exceptions by using else-if statements. Some of these programs, you need to tell how many of these else-if statements that you need for the program to work properly. Sometimes with these types of programs, you might also end your code with an else statement, which is when none of the options are true. But, an ‘else’
statement is not required for the code to run. And identifying when you need an ‘else’ statement within an ‘else-if’ function depends on the situation that you are coding. Some programs might not need something to happen if all the other statements are false. Determining this fact can change the course of your code.

If you follow all of the general principles above, observing and debugging a problem in code should become easier for the person looking at the code. Making code look ‘good’ is very dependent on the type of code that is being worked on, and the program itself. But, these general principles should make a code that was once hard to read or hard to debug become easier and more pleasant to look at and understand.