Readability & Maintainability

Code is speech, according to the United States court system.

Good speech or writing should be clear to understand. The same is true for code: well-written code should be easily understood by anyone of equally good (or better) coding ability. This means that when writing code, you should prefer to use techniques that are relatively easy to understand, when choosing between multiple ways to solve a problem.

Well-written code is also designed in such a way that has any future code editors in mind. Does your code have any values that may need tweaked during or after testing? Make those variables that live in a list at the top of the program. Does your code have any complex operations? Multi-part processes could be written in several smaller defined functions with very clear names, to make the whole program easier to understand in several small chunks. This may also help you to reuse parts of your program, if things are getting repetitive. Do you have lots of variables? Try naming your variables in such a way that their names help you understand the variable’s role in the program. You can even try using a naming convention such as [camelCase](https://en.wikipedia.org/wiki/Camel_case) to make your variable and function names more legible.

# Example Sketch 1: A Bad Readability Example...

Below is an example sketch that will compile in the Arduino IDE. You can copy it over and try for yourself. This program will run just fine on an Arduino, but the way it is written makes it very hard to understand. This is a first example to consider why you should write READABLE code.

**int x =**

**2, y=3;**

**void setup() {**

**Serial.**

**begin(9600);**

**Serial.println(z(x,y));**

**}**

**void loop() {}**

**int z(int w, int r){**

**return w+r;**

**}**

What is this example program supposed to return over the Serial Monitor?

How does the program achieve this output?

What are some of the barriers to understanding this particular program?

# 

# Example Sketch 2: Readability Improved

Below is an example sketch that will compile in the Arduino IDE. You can copy it over and try for yourself. This program is an improved version of the previous example program.

// global variables

**int x = 2;**

**int y = 3;**

**void setup() {** //setup function runs once at startup

**Serial.begin(9600);** //start serial communication

**int sum = addition(x,y);** //using a variable for clarity

**Serial.println(sum);** // send the sum (result of addition fn)

**}**

**void loop() {** //loop runs continuously after startup

//loop function is intentionally left empty

**}**

//user defined function (used in the setup fn above)

**int addition(int a, int b){** //add the two given values

**return a+b;** //return the sum of the values passed into the fn

**}**

What is this example program supposed to return over the Serial Monitor?

How does the program achieve this output?

What changes made this code easier to understand?

# 

# Example Sketch 3: Bad Maintainability

Below is an example program that blinks the built in LED on an Arduino Uno or Nano. However, this code is not as maintainable as it could be with a few minor changes. For example, think about what changes you would need to make if you need to change which pin goes on and off.

**void setup() {** // the setup function runs once

**pinMode(13, OUTPUT);** //put pin 13 in output mode

**}**

**void loop() {** //loop runs continuously after setup

**digitalWrite(13, HIGH);** //5V output on pin 13

**delay(1000);** //wait one second

**digitalWrite(13, LOW);** //0V output on pin 13

**delay(1000);** //wait one second

**}**

# Example Sketch 4: Better Maintainability

Below is an example program that blinks the built in LED on an Arduino Uno or Nano. However, this code is more maintainable than the previous version. If the pin number needs changed, this program would only require a change to one line of code that is easily found at the beginning of the program. There are nuanced choices to support maintainability, but this is one example.

**int LED = 13;** //variable to hold LED address

**void setup() {** // the setup function runs once

**pinMode(LED, OUTPUT);** //put LED pin in output mode

**}**

**void loop() {** //loop runs continuously after setup

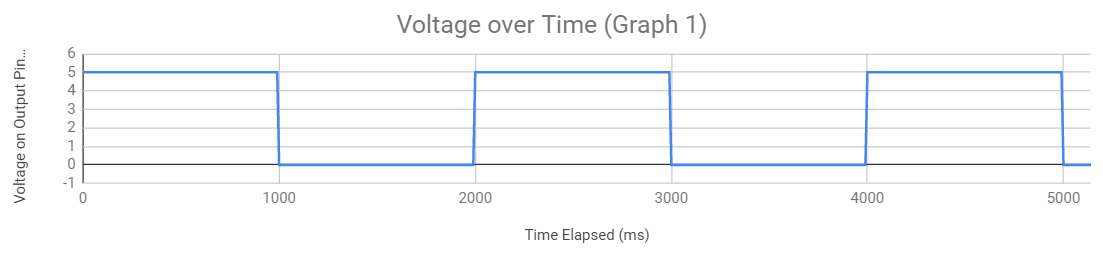
**digitalWrite(LED, HIGH);** //5V output on LED pin

**delay(1000);** //wait one second

**digitalWrite(LED, LOW);** //0V output on LED pin

**delay(1000);** //wait one second

**}**

The example sketch above would output the voltage shown in the graph below: 

Example sketches #3 and #4 are very similar, and have identical output. What makes 4 more maintainable?

Answer here

Commonly in Arduino sketches, the global values you may need to tinker with are kept at the top of the sketch, above the void setup(). This can be a good practice to adopt. Another good technique (to save on sketch memory) is to look in to ‘[const](https://www.arduino.cc/en/pmwiki.php?n=Reference/Const)’ or ‘[#define](https://www.arduino.cc/reference/en/language/structure/further-syntax/define/)’ to have these not use up precious memory space.

Also, it is often easier to change a variable in code, rather than the wiring on a breadboard or (much worse) a soldered connection. For this reason, there are some cases where planning ahead for your circuits can inform how you write your code.

# 

# Arduino’s Own Explanations

## Readability

[Arduino Style Guide](https://www.arduino.cc/en/Reference/StyleGuide) << guidance on how to write Arduino code so that it is readable

[Arduino API Style Guide](https://www.arduino.cc/en/Reference/APIStyleGuide) << guidance on how to write Arduino Libraries for readability

## Maintainability

[Software Stack Exchange: maintainable code?](https://softwareengineering.stackexchange.com/questions/141005/how-would-you-know-if-youve-written-readable-and-easily-maintainable-code) << Forum guidance on maintainable code

[7 Rules of Clean, Readable, Maintainable Code](http://shhetri.github.io/clean-code/#/) << A slideshow built by a professional

[More advice on writing maintainable code](https://www.red-gate.com/simple-talk/dotnet/net-framework/writing-maintainable-code/) << a writer from Redgate

# Create your own Sketches

*You don't know if you understand it, until you can create it from nothing...*

Using what you have seen above, in the example sketches and from Arduino’s reference materials. Create your own sketch that shows you understand how to distinguish hard-to-read and very readable code. You have plenty of creative license in this goal, but you must make a sketch that successfully shows the difference. A screenshot of your sketch and output is half of your response to these. A written explanation of the logic is also required to prove that you understand what you are doing with this work. Primary tasks are required for all students. Secondary tasks are required to get a top grade.

* Primary tasks:
  + Write a simple sketch that is styled so that it is hard to read, but functions properly. Also, rewrite the same program so that it is both functional and easy to read, so a new user could make sense of it at a glance.

Screenshot & Explanation here

* + Write an example sketch that is an example of bad maintainability. Then rewrite the sketch to achieve the same output, but with more of a design for maintainability.

Screenshot & Explanation here