Chapter 1

IoT Style Guide

A programming style guide is an opinionated guide of programming conventions, style, and best practices for a team or project. Some teams call it their coding guidelines, coding standards, or coding conventions. While these each have their own meaning in programming, they generally refer to the same thing.

In IoT, we will use the below conventions:

- Start all code with the standard comment block

```
/*
 * Project:
 * Description:
 * Author:
 * Date:
 */
```

- Provide meaningful names for variables, constants and functions. Variables and constants should be nouns and functions should be verbs.
  - The one exception to variables being meaningful nouns is that integer variables i, j, and k may be used for indexing arrays or FOR loops.
  - Constants should be in all capitals.

```
    const int LEDPIN = 13;
```

- Variables and functions should be in camelCase[1], with the first letter lowercase: float elapseTime;

```
    float elapseTime;
```

- Structs, ENUMs, and classes should also be in camelCase, but with the first letter capitalized.

```
    enum TrafficState{
      GREEN,
      YELLOW,
      RED,
      REDYELLOW;
    }
```

- Code should be organized as follows:

  1. #include <>
  2. #include ""
  3. #define
  4. Data Types (e.g., structures and ENUMs)
  5. Globals
  6. void setup()
  7. void loop()
  8. user defined functions

  NOTE: const variable should be used instead of #define.

- Avoid global variables where they are unnecessary.

  - Global Variables - The variables declared outside any function are called global variables. They are not limited to any function. Any function can access and modify global variables. Global variables are automatically initialized to 0 at the time of declaration. Global variables are generally written in the header-section of the code before void setup().

[1] Camel case is the practice of writing phrases without spaces or punctuation, indicating the separation of words with a single capitalized letter.

  - Local Variables - Local variables are declared inside of a function and only exist during that function call and are not accessible by other functions. Subsequent function calls reset the variable to its initial condition, which is 0 unless otherwise declared. Note: variables defined in void setup() or void loop() are local variables not accessible by other functions, and in the case of void loop() reinitialize during each iteration. Variables uses in void setup() and void loop() should therefore be declared as global variables.
  - Static Variables - A static variable is a local variable that is able to retain its value between different function calls. A static variable is only initialized once. If it is not initialized, then it is automatically initialized to 0.

- Opening braces should be on the same line as the conditional or function.

```
if(buttonState) {
  digitalWrite(LEDPIN, HIGH);
}
else {
  digitalWrite(LEDPIN, LOW);
}
```

- Code blocks should be indented.

```
for(i=0; i < 13; i++) {
  value = i * 3;
  if(num%2 == 0) {
    Serial.printf("The number %i is even\n",value);
    digitalWrite(LEDPIN, HIGH);
  }
  else {
    Serial.printf("The number %i is odd",value);
    digitalWrite(LEDPIN, LOW);
  }
}
```

- Use braces; avoid loops and conditionals without them.

```
// Proper use of braces:
if(buttonState) {
  digitalWrite(LEDPIN, HIGH);
}
```

```
5
6  // Avoid loops and conditions without braces. Do not do:
7  if(buttonState)
8    digitalWrite(LEDPIN, HIGH);
9  //or
10 if(buttonState) digitalWrite(LEDPIN, HIGH);
```

- Use parentheses for clarity, especially with bit operations.

```
1 pixelColor = (red << 16) | (green << 8) | (blue);
2 //or
3 if((currenTime - lastTime) < timerSetting) {
4   Serial.printf("Time is up \n");
5 }
```

- When "printing" text and values to the Serial Monitor, OLED, or UART devices, use .printf() instead of .print() or .println()

```
1 Serial.printf("We use .printf() to display value = %0.2f
    \n",value);
2 Serial.println("We do not use .print() or .println()");
```

- Avoid code duplication. (i.e., appropriately use functions)
  - One of the main reasons to use a function is reusability. Once a function is defined, it can be used over and over and over again. You can invoke the same function many times in your program, which saves you work. Imagine what programming would be like if you had to teach the computer about sines every time you needed to find the sine of an angle! You'd never get your program finished!
  - Another aspect of reusability is that a single function can be used in several different (and separate) programs. When you need to write a new program, you can go back to your old programs, find the functions you need, and reuse those functions in your new program. You can also reuse functions that somebody else has written for you, such as the sine and cosine functions.
- Place a descriptive comment the line before all functions. Use comments in code as needed to clarify code logic.

```
// Get x-axis acceleration from MPU6050
float getAccX() {
  byte accelHigh, accelLow;
  int16_t accel;

  // Set the "pointer" to the 0x3B memory location
  // of the MPU and wait for data
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x3B); // starting with register 0x3B
  Wire.endTransmission(false); // keep active.

  // Request and then read 2 bytes
  Wire.requestFrom(MPU_ADDR, 2, true);
  accelHigh = Wire.read(); // x accel MSB
  accelLow = Wire.read(); // x accel LSB

  accel = accelHigh << 8 | accelLow;
  return accel;
}
```