Code Building Blocks

Listed here are some common code patters you might use when doing Arduino programming. You don't need to copy them exactly – you can change variable names or combine patterns to suit your needs.

Variables

Variables are how a program remembers values. Variables must be declared, and then can be reused until the block of code they were declared in (inside of the "`{ ... }`") ends.

```
int led_on_time = 2000;
const int LED_PIN = 13;
bool use_debug_printing = true;
```

Structure of an Arduino Program

Arduino programs have a setup function that gets called once when the Arduino is powered up, and then the loop function is called over and over until the Arduino program is turned off. Remember this, take advantage of it.

Reading From and Writing to Pins

The Arduino interacts with the world through its GPIO pins (numbered on the Arduino board). Pins must be configured before they can be used (usually in `setup()`). Each function requires a pin number as the first parameter, and a value to write as the second parameter when writing. Make sure you are using a pin that allows for the operation you are trying to. Note that another name for analog output is PWM (Pulse Width Modulation).

`pinMode(MY_OUTPUT_PIN, OUTPUT);`	Use this when you want to send voltage to something, like an LED or a motor.
`pinMode(MY_INPUT_PIN, INPUT);`	Use this when you expect voltage from something, like a sensor.

	Digital	Analog
Read	`// pin_value will be HIGH or LOW` `bool pin_value = digitalRead(MY_PIN);`	`// reading can be 0-1023` `int reading = analogRead(MY_PIN);`

Write	`// we can use HIGH or LOW for`
	`// the value`
	`digitalWrite(MY_PIN, HIGH);`
		`// we can only write`
		`// values 0-255`
		`analogWrite(MY_PIN, intensity);`

Data Types and Operators

There are three main types of data we will use, and some operators that you will find necessary to use that data to make your programs interesting.

`int`	`int my_number = 5;`	A whole number. This type of variable can hold numbers from – to
`bool`	`bool my_bool = true;`	This type can only be `true` or `false`, or `HIGH` or `LOW` (0 or 1 for the computer).
`String`	`String my_string = "This is a string!";`	Strings are the CS way of talking about words or sentences.

=	Assignment	Use this to set the value of a variable	`int my_int = 5;` `// my_int is now 5`
+, -, *, /, %	Add, subtract, multiply, divide, and modulo*	Use these to do math in the usual way	`int my_int = 4 + 5;` `// my_int is now 9`

+=, -=, *=, /=, %=	Math operation with assignment	Use these to be lazy and do math and assignment in one line.	`int my_int = 5;`
			`my_int += 4;`
			`// my_int is now 9`
			`// same as`
			`// my_int = my_int + 4;`

++, --	Increment, decrement	Easy way to add or subtract 1 to/from a variable	`int i = 0;`
			`i++;`
			`// i is now 1`
			`// same as`
			`// i += 1`

==, !=	equal to, not equal to	Use these to see if two things are the same/different. BE SURE TO USE TWO '=' SYMBOLS WHEN YOU WANT COMPARISON. THIS IS A COMMON MISTAKE!	`int this_number = 1;`
			`int that_number = 2;`
			`if (this_number == that number) {`
			`  Serial.println("Same!"`
			`}`
			`if (this_number != that number) {`
			`  Serial.println("Different!"`
			`}`
			
			`// "Different" gets printed`

<, <=	Less than, less than or equal to	Use this to see if a number is less than another.	`if (7 < 8) {`
			`  Serial.println("YAY!"`
			`}`

>, >=	Greater than, greater than or equal to	Use this to see if a number is greater than another	`if (this_number > that_number) {`
			`  Serial.println("this is big!"`
			`}`

*Modulo returns the remainder of a division, like when you do long division by hand			

Branches (aka "if statements")

Sometimes you need a program that can make decisions. Sometimes this is a simple decision with one option, sometimes two, sometimes there will be many. Use comparison operators and bools to make this happen.

One Option	Two Options	Many Options

`if (/*condition*/) {`
`  /*do something*/`
`}`
	`if (/*condition*/) {`
	`  /*do something*/`
	`} else {`
	`  /*do something else*/`
	`}`
		`if (/*condition*/) {`
		`  /*do something*/`
		`} else if (/*condition*/) {`
		`  /*do something else*/`
		`} else {`
		`  /*do something else else*/`
		`}`

Loops

We can use loops for counting, or waiting for something to change. The for loop patterns are the most commonly used.

	`for` Loop	`while` Loop

Do something 5 times	`for (int i = 0; i < 5; i++) {`
	`  do_something();`
	`}`
		`int i = 0;`
		`while (i < 5) {`
		`  Serial.println(i);`
		`  i++;`
		`}`

Iterate over an array	`int my_array[3] = {1, 2, 3};`
	`for (int i = 0; i < 3; i++) {`
	`  Serial.println(my_array[i]);`
	`}`
		`int my_array[3] = {2, 5, 9};`
		`int i = 0;`
		`while (my_array[i] != 5) {`
		`  i++;`
		`}`
		`Serial.println("Found it!");`

Functions

We can use functions to make our code more expressive and make it so we don't have to write so many lines. Use the return key-word to let the function know what to return to the calling function and/or stop executing early.

```
void turn_led_on_then_off(int pin, int delay_time) {
  digitalWrite(pin, HIGH);
  delay(delay_time);
  digitalWrite(pin, LOW);
  delay(delay_time);
}
```

```
void debug_print_pin_state(int pin) {
  bool pin_state = digitalRead(pin);
  Serial.print("Pin ");
  Serial.print(pin);

  if (pin_state == HIGH) {
    Serial.println(" is HIGH");
 } else {
    Serial.println(" is LOW");
  }
}
```

```
bool number_is_even(int number) {
  return (number % 2) == 0;
}
```

Then call the function from another function like setup, loop, or one of your own!

```
void setup() {
  turn_led_on_then_off(RED_PIN, 250);
  turn_led_on_then_off(BLUE_PIN, 250);
  turn_led_on_then_off(GREEN_PIN, 250);
}
```

```
bool do_something(int intensity) {
  if (number_is_even(intensity)) {
    analogWrite(LED, intensity);
    return true;
  }
    return false;
}
```

```
void loop() {
  // do some stuff with a
  // pin that has a sensor
  // attached
  debug_print_pin_state(SENSOR_PIN);
}
```