

#What are Variables?

Variables are one of the cornerstones of programming and development. At the most basic level, variables are containers. On their own, a variable is just an empty placeholder waiting to store a value. Variables have a *type*, *name*, and *value*. Once data is stored inside a variable, it can be used and manipulated in your scripts.

Declaring Variables

Variables must be declared before they're used. When coding for the Arduino, variable declarations must include the name and type. Setting the initial value is optional. Here are two examples, both of which are **correct**:

```
int ledPin1; // this declaration includes the name (ledPin1)
int ledPin2 = 12; // this declaration also includes the init.
```

Variable Types

The above examples specify the variable **type**. Understanding variable types is an important aspect of programming, especially when coding for Arduino because you must correctly specify the type when declaring the variable. This isn't the case in other languages such as Javascript, so if you're coming from a background with "weakly typed" variables you'll need to make sure that you're properly specifying the variable types when working with Arduino.

Types:

Here are some of the most common variable types you'll encounter. There are more types, but they're rarer. You'll encounter these types in most Arduino code:

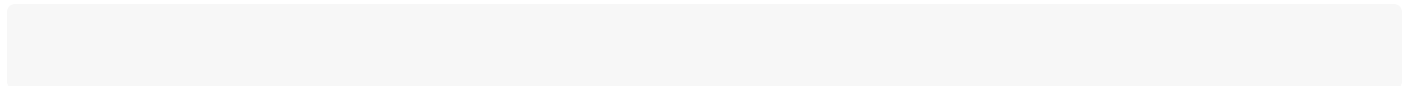
Note: The initial value is optional, whereas the type and name are required.

1. **Integers (int):** Integers are primarily what you'll encounter for number storage. These are only integers with no decimal values that are between -32,768 and 32,768. The above examples are type **int**. These are declared as **int varName = 10;**
2. **Characters (char):** Characters are either single or multiple characters (strings). These are declared like this: **char exChar = 'J';** for single characters and **char exStr = "Jonathan";** for strings. Strings use double quotes whereas single characters use single quotes.
3. **Floating Points (float):** Floating points are numbers that have a decimal point. These variables are common in Arduino projects. These are declared like this: **float zVar = 2.0;** When doing any math operations on floating points, make sure to include the decimal such as 2.0 instead of 2. If you don't include the decimal, it will be interpreted as an integer instead of a float.
4. **Long (long):** These are long values that don't include decimals. These are the values between -2,147,483,648 and 2,147,483,647.
5. **Booleans (bool):** These are Boolean values (true or false).
6. **Constants (const):** This isn't a variable type- it's a qualifier that changes the behavior of a variable so that it can be read/accessed throughout the code but is unable to be modified. It remains *constant* throughout. Here's an example: **const int ledPin = 13;** This would make it so that the variable ledPin value can be accessed in the program but never changed.

Example: ledPin Variable

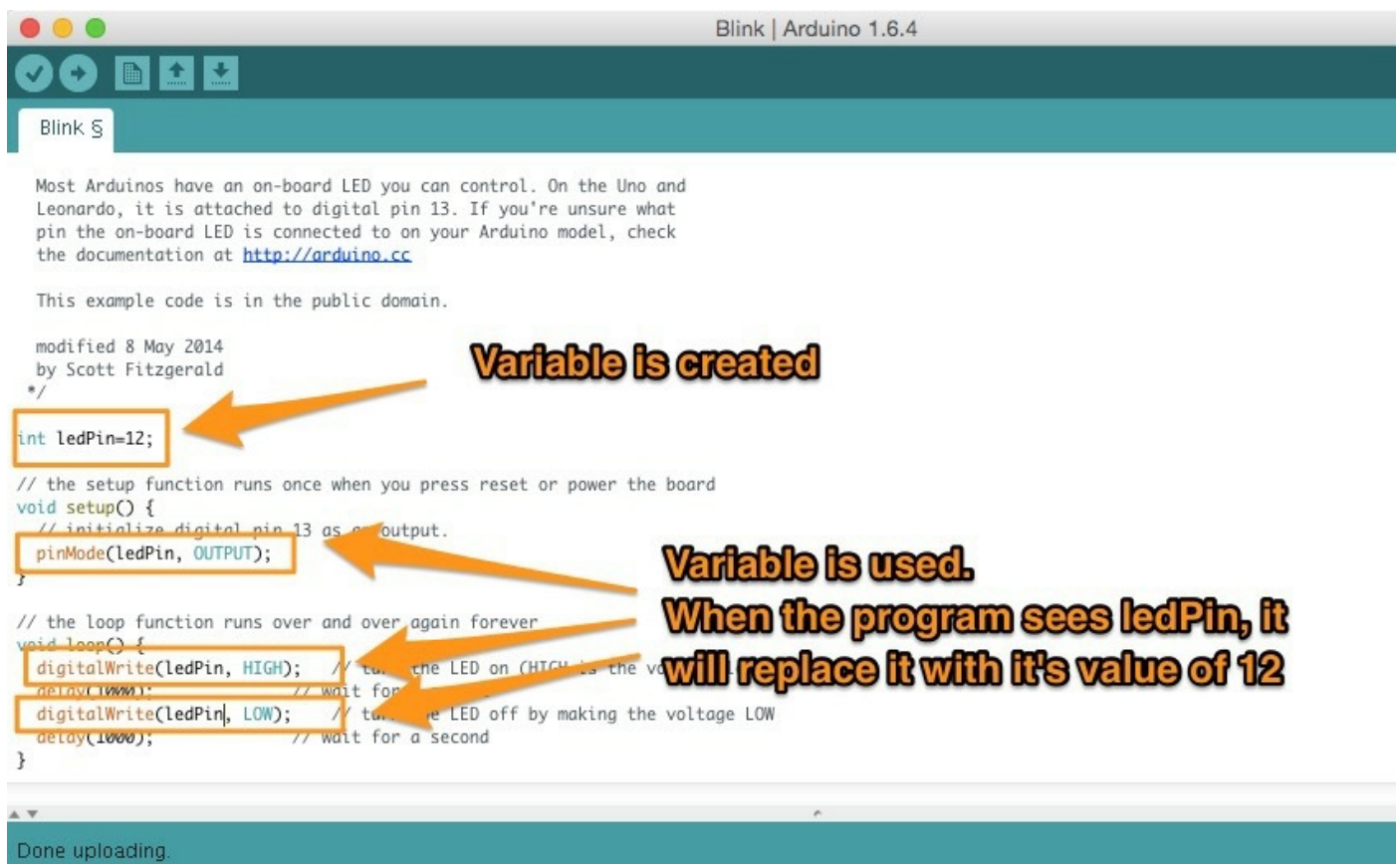
Now that you have an overview of variables, let's take a look at an example. In the "Blink LED" project the LED is in pin 13 in the Arduino, and the Arduino code corresponds to this with a **pinMode** value of 13. However, you've likely noticed that when you wanted to change the LED to pin 12 in the Arduino, you've also needed to change it in several spots in the code (pinMode, digitalWrite). This isn't too difficult when you don't have many lines of code, however it becomes overwhelming when working in larger projects. Additionally, when you find yourself needing to do the same task repeatedly while programming, you should ask yourself if there is a more effective method. In this particular situation, using variables is the optimal method!

You're going to create a variable that stores the value of the intended ledPin:



The variable in this example has an initial value of 12. Note that if you were to change the pin that the LED is in you'll need to change this variable.

Take a look at this updated version of the Blink code that now uses variables:



Notice that our variable `ledPin` is now being passed as a parameter for **`pinMode`** and **`digitalWrite`**. Since variables are containers, the variable named "`ledPin`" is currently storing an the number 12, an integer. Now when you want to change the pin that the LED is in on the Arduino you only need to update the Arduino code in one place: the variable declaration at the top of the code.

Variable Scope

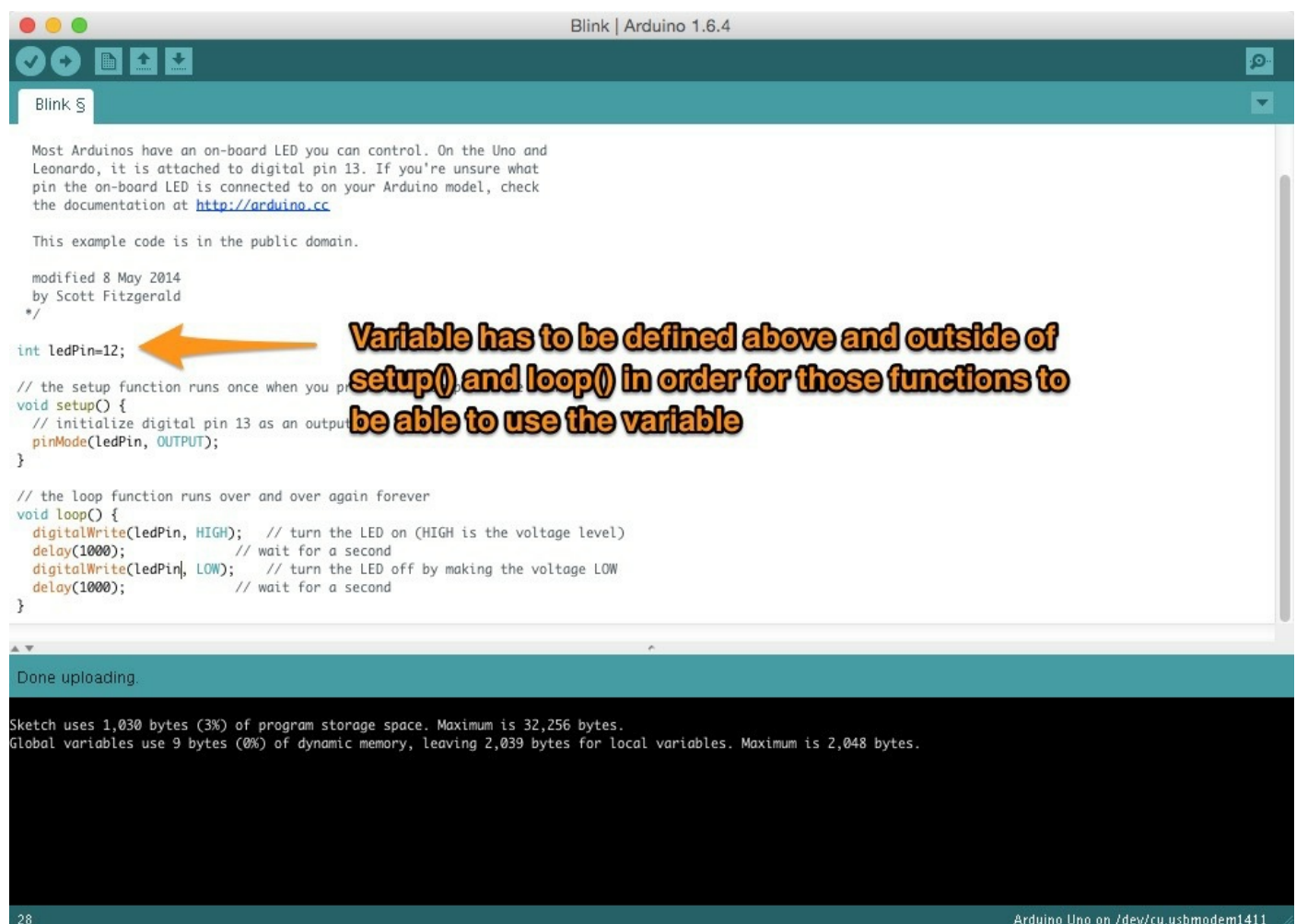
The Arduino programming language is based on the Processing language, which itself is based on C. Variables in C have a property called **scope**, and therefore variables in Arduino also have this property. The two scopes that variables have are *global* and *local*:

Global variables can be seen by every function in your Arduino code, whereas local variables are only seen by the function that they're declared in. Here's a non-programming parallel:

Your full given birth name is on your birth certificate and is seen/accessed by anyone in any of your circles, as long as you reveal it. This is your "global name." A nickname that is known to only your inner circle of friends and only applied to you when in their company is your "local name." There is no record of it anywhere else other than that particular local circle, so if someone called you by this particular nickname in an official capacity, it wouldn't be recognized.

Remember that **void setup()** and **void loop()** are both functions. This means that any variables declared within those functions will be local to it. Any global variables must be **declared outside** of these functions.

Here's an example:



```
Blink | Arduino 1.6.4

Blink

Most Arduinos have an on-board LED you can control. On the Uno and Leonardo, it is attached to digital pin 13. If you're unsure what pin the on-board LED is connected to on your Arduino model, check the documentation at http://arduino.cc

This example code is in the public domain.

modified 8 May 2014
by Scott Fitzgerald
*/

int ledPin=12;

// the setup function runs once when you power up
void setup() {
  // initialize digital pin 13 as an output
  pinMode(ledPin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(ledPin, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(ledPin, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}

Done uploading.

Sketch uses 1,030 bytes (3%) of program storage space. Maximum is 32,256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2,039 bytes for local variables. Maximum is 2,048 bytes.

28 Arduino Uno on /dev/cu.usbmodem1411
```

Notice that **int ledPin = 12;** is declared before the setup function begins. This means that the ledPin is a global variable that can be

seen/accessed by *all* functions in the program. A quick way to tell if a variable is global is to check to see that it is declared *outside* of a function, including both **void setup()** and **void loop()**.

Note that the `ledPin` variable is accessed in the **void loop()** later in the example. If the `ledPin` variable was declared *within* the **void setup()** function, it wouldn't have been accessible by the methods in the loop. Likewise if it had been declared within **void loop()**, it wouldn't be accessible during **void setup()** or anything else.

Test it!

Go ahead and move the variable declaration into the **void setup()** function and test your code. You'll get an undefined error.

Activity: Practicing with Variables

Variables make your code much more flexible. The purpose of this quick activity is to practice integrating variables into a familiar project before you dive into a completely new concept.

Supplies:

- The same setup and supplies from the *Breadboard Blink* activity.
- If you haven't already, go ahead and rewire this circuit and make sure that your sketch is uploaded to the board.

Steps:

1. Create a new sketch and name it "Blink-Variables" or something similar.
2. Copy the code from your previous sketch and paste it into the new

sketch. You're going to be altering it but there's no need to completely rewrite it.

3. Using the examples in this lesson, create a "ledPin" variable.
 - Make sure to use the correct datatype for this!
 - Also make sure to make it **global**!
 4. Replace any of the "hard coded" pin number entries with your new variable.
 - If you have multiple LEDs, you'll need multiple, uniquely named variables!
 5. Test your code and debug any errors.
-

Resources

[Arduino Reference](#) : This is the Arduino Reference page for variables. There is lots of useful content as well as more examples.