Variables

- A variable is a way of naming and storing a numerical value for later use by the program.
- As their namesake suggests, variables are numbers that can be continually changed as opposed to constants whose value never changed.
- A variable needs to be declared and optionally assigned to the value needing to be stored.

The following code declares a variable called inputVariable and then assigns it the value obtained on analog pin 2:

```
int inputVariable = 0;                  // declares a variable and
assigns value of 0

inputVariable = analogRead(2);   // set variable to value of analog pin
2
```

'inputVariable' is the variable itself. The first line declares that it will contain an int, short for integer. The second line sets the variable to the value at analog pin 2. This makes the value of pin 2 accessible elsewhere in the code.

Variable declaration

- All variables have to be declared before they can be used.
- Declaring a variable means defining its value type, as int, long, float, etc., setting a specified name, and optionally assigning an initial value.
- This only needs to be done once in a program but the value can be changed at any time using arithmetic and various assignments.

The following example declares that inputVariable is an int, or integer type, and that its initial value equals zero. This is called a simple assignment.

```
int inputVariable = 0;
```

A variable can be declared in a number of locations throughout the program and where this definition takes place determines what parts of the program can use the variable.

Variable scope

- A variable can be declared at the beginning of the program before void setup(), locally inside of functions, and sometimes within statement blocks such as for loops. Where the

variable is declared determines the variable scope, or the ability of certain parts of a program to make use of the variable.

- A **global variable** is one that can be seen and used by every function and statement in a program. This variable is declared at the beginning of the program, before the setup() function.

- A **local variable** is one that is defined inside a function or as part of a for loop. It is only visible and can only be used inside the function in which it was declared. It is therefore possible to have two or more variables of the same name in different parts of the same program that contain different values. Ensuring that only one function has access to its variables simplifies the program and reduces the potential for programming errors.

The following examples shows how to declare a few different types of variables and demonstrate each variables visibility:

```
int value;                  // ‘ value ‘ is visible to any function

void setup()

{

      // no setup needed

}

void loop()

{

      for (int i=0; i<20)   // ‘ i ‘ is only visible

      {                 // inside the for-loop

         i++;

}

float f;                        // ‘ f’ is only visible

}                       // inside loop
```

if

- if statements test whether a certain condition has been reached, such as an analog value being above a certain number, and executes any statements inside the brackets if the statements is true. If false the program skips over the statement.

The format for an if test is:

```
if (someVariable == value)
{
      doSomething;
}
```

The above example compares someVariable to another value, which can be either a variable or constant. If the comparison, or condition in parentheses is true, the statements inside the brackets are run. If not, the program skips over them and continues on after the brackets.

Note: Beware of accidentally using '=', as in if(x=10), while technically valid, defines the variable x to the value of 10 and is as a result always true. Instead use '==', as in if(x==10), which only tests whether x happens to equal the value 10 or not. Think of '=' as "equals" opposed to '==' being "Is equals to".

if … else

if … else allows for 'either-or' decisions to be made.

For example, if you wanted to test a digital input, and do one thing if the input went HIGH or instead do another thing if the input was LOW, you would write that this way:

```
if (inputPin == HIGH)
{
      doThingA;
}
else
{
      doThingB;
}
```

Else can also precede another if test, so that multiple, mutually exclusive tests can be run at the same time. It is even possible to have an unlimited number of these else branches. Remember though, only one set of statements will be run depending on the condition tests:

Else if block may be used with or without a terminating else block and vice versa. An unlimited number of such else if branches are allowed.

```
if  (inputPin < 500)
```

```
{
      doThingA;
}
else if (inputPin >= 1000)
{
      doThingB;
}
else
{
      doThingB;
}
```

Note: An if statement simply tests whether the condition inside the parentheses is true or false. This statement can be any valid C statement as in the first example, if (inputPin == HIGH). In this example, the if statement only checks to see if indeed the specified input is at logic level high, or +5V.

digitalRead(pin)

- Reads the value from a specified digital pin with the result either HIGH or LOW.
- The pin can be specified as either a variable or constant (0-13).

```
value = digitalRead(Pin);   // sets 'value' equal to the input pin
```