Arduino Programming :

Arrays

Objectives

- Understand what an array is
- Understand how data can be stored in arrays
- Understand the practical uses of arrays

Arrays

An *array* is a series of *objects* of the same *size* and *type.*

- An *object* is usually a variable
- *Size* refers to number of bits used: boolean, int, float, etc.
- *Type* usually refers to numeric or character (numbers vs. letters)

Variables typically hold a single value (although the value can change):

```
example:    val = 32;           // val equal to 32
            val++;              // val now equals 33
```

An array also has a variable name, but has two brackets following it:

int val[x]   the letter x refers to the *index* of the array

Arrays – using an index

Arrays can hold multiple elements or values.

The *index* defines a specific element in the array:

```
val[0] = 3;

val[1] = 46;

Val[2] = 17
```

Arrays: initializing the variable

Arrays can be initialized with preset values:

int val[] = {3, 46, 17};

```
val[0] = 3;    // The first indexed value equals 3

val[1] = 46;   // The second indexed value equals 46

val[2] = 17   // The third indexed value equals 17
```

Arrays initializing the variable

Arrays can also be defined with a maximum “size” before setup() :

```
int val [50];                    // allocate 50 values ( 0 – 49)
```

This is really setting aside 50 memory locations to store all of the possible indexed values of variable val.

Using Arrays

How does this work??

Let’s say we want to store three analog values from a single potentiometer at different times.

We could store them as:

```
analog1 = analogRead(potVal);          // Read the voltage
analog2 = analogRead(potVal);          // Read the voltage
analog3 = analogRead(potVal);          // Read the voltage
```

However, a more efficient way is to create an array.

Arrays

```
int x;
int analog[3];
potPin = 0;   // Potentiometer connected to analog input 0

for (x = 0; x < 3; x++)
{
   analog[x] = analogRead(potPin);
   delay(1000);                          // read 3 values spaced
}                                        // one second apart
```

This becomes more efficient when we work with lots of values.

Arrays – another example

What if we wanted to print these values now?

```
for (x = 0; x < 3; x++)
{
   Serial.println (analog[x]);
   delay(200);
}
```

This becomes more efficient when we work with lots of values.