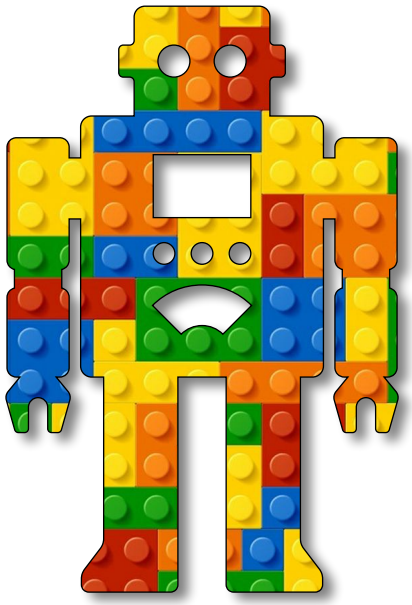
Intelligent Machines Module A Unit #6 boolean



Content

Module A Unit #6 Boolean

Sketch A6.1	analogWrite()
Sketch A6.2	minimum value
Sketch A6.3	halfway house
Sketch A6.4	short delay
Sketch A6.5	increments
Sketch A6.6	negative fade
Sketch A6.7	boolean operator
Sketch A6.8	brightness
Sketch A6.9	fadeValue
Sketch A6.10	without delay
Sketch A6.11	change the state
Sketch A6.12	a constant interval
Sketch A6.13	unsigned long
Sketch A6.14	the previous number
Sketch A6.15	checking the difference
Sketch A6.16	toggle the state

Introduction to boolean

Boolean is a form of logic which can be described as gates; the most common are AND gates and OR gates. There are more of them, and they form the basics of computing. They are also used widely in higher-level programming. Two symbols are used: && for AND and || for OR (called pipes). Trying to explain it makes it seem quite confusing, and yet it is literally the simplest logic. The best way to get a grasp is simply to use it, and you will see how simple yet powerful it is.

The if() statement can be similar to the while() loop: if something is true or a condition is met, then do something; for instance:

```
if(x < 100 && y > 50)
```

...means if x is less than 100 and y is greater than 50, then do something...

Sketch A6.1 analogWrite()

Going back to built-in LEDs, we can fade the LED using the `analogWrite()` function. The pins all support `PWR` (I won't go into the details here just yet), but it means we can assign a value to the pin, so if we give pin `13` a value of `255`, that means fully `on` (`100%`). If we assign `0` (`0%`), then it is totally `off`. Any numbers in between those two values will give a brightness between the two. The following sketch will demonstrate this.

First, full brightness has a value of `255`. Notice we put nothing in `void setup()`. The `//` means this comments out anything we write afterwards; the computer ignores it.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 255);
}
```

🗒️ Notes

Nice bright LED.

🛠️ Code Explanation

`analogWrite(13, 255);`	This writes a value (255 in this instance) to pin 13. Although it is a digital pin it is treating it as an analog input.

Sketch A6.2 minimum value

Now give it the minimum value of `0`. This should be completely `off`.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 0);
}
```

🗒️ Notes

Nicely done!

🛠️ Code Explanation

`analogWrite(13, 0);`	This writes a value (0 in this instance) to pin 13. Although it is a digital pin it is treating it as an analog input.

Sketch A6.3 halfway house

This is somewhere in between, approximately 125. It is a bit hard to see, so we will do an incremental fade-in in the following sketches.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 125);
}
```

Notes

Nice but dim.

Sketch A6.4 short delay

The delay is small, and it is the amount of time (in milliseconds) that it shines at that intensity of brightness.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 125);
  delay(5);
}
```

Notes

Nothing to see here.

Sketch A6.5 increments

Instead of a fixed value, we want to increment it by a value of 1 every 50 milliseconds.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  for (int i = 0; i < 256; i++)
  {
    analogWrite(13, i);
    delay(5);
  }
}
```

Notes

You should see it pulsing, return to 0, then increase to 255.

Code Explanation

`analogWrite(13, i);`	This increments to brightness from 0 through to 255 incrementing the value of i by 1 each iteration of the loop

Sketch A6.6 negative fade

Now to fade back down again. Notice the `i--` at the end of the decreasing fade. What you should have now is the LED getting rapidly brighter and then quickly fading and then brighter repeatedly.

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  for (int i = 0; i < 256; i++)
  {
    analogWrite(13, i);
    delay(5);
  }
  for (int i = 255; i > 0; i--)
  {
    analogWrite(13, i);
    delay(5);
  }
}
```

Notes

A nicely pulsing LED, up and down the scale.

Sketch A6.7 boolean operator

❗ New sketch.

Another way to do this is to use a Boolean operator with an `if()` statement. So start with a basic sketch, putting the LED on maximum brightness (255).

Arduino sketch

```
void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, 255);
}
```

Notes

Start with it on.

Sketch A6.8 brightness

Now adding a variable for brightness.

Arduino sketch

```
int brightness = 255;


void setup()
{
  // nothing here
}


void loop()
{
  analogWrite(13, brightness);
}
```

Notes

Still the same.

Boolean Operators

This is all to do with logic gates: AND, OR, NOT.

!	logical NOT gate, i.e. a condition where something is not true can be written != (not equal to)
&&	logical AND, i.e. a condition where two things have to be true
||	Logical OR, i.e. a condition where either one of two conditions is true

Sketch A6.9 fadeValue

We want to fade it up and down. So the initial `brightness` we will change to `0` so that when it reaches `255`, then the brightness diminishes and when it is zero, it gets brighter. We are using a boolean operator that is called pipes `||` which means that one `OR` the other has to be true.

The `fadeValue` is an integer variable that is how much it will fade by in steps of `5` in this case.

Arduino sketch

```
int brightness = 0;
int fadeValue = 5;

void setup()
{
  // nothing here
}

void loop()
{
  analogWrite(13, brightness);
  brightness = brightness + fadeValue;
  if (brightness == 0 || brightness == 255)
  {
    fadeValue = -fadeValue;
  }
  delay(50);
}
```

🗒️ Notes

Pulsing up and down. It toggles the value of `fadeValue` between `-5` and `5`, depending on whether it is starting from `0` or `255`. Just follow the logic.

🛠️ Code Explanation

`if (brightness == 0 || brightness == 255)`	If brightness is equal to 0 or 255 then change fadeValue from positive to negative or vice versa

Sketch A6.10 without delay

Sometimes it is critical that we don't use the function `delay()` simply because the whole programme stops while `delay()` is operating, and you might want other things to happen while there is a delay. There is a way of doing the blink sketch without using the `delay()` function.

❗ Starting with a very basic sketch.

Arduino sketch

```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
}
```

🗒️ Notes

Starting again.

Sketch A6.11 change the state

Instead of using HIGH and LOW for on and off, we will give these a variable name so that we can change the state of each. We will call this variable ledState and set it to LOW initially.

Arduino sketch

```
int ledState = LOW;


void setup()
{
  pinMode(13, OUTPUT);
}


void loop()
{
  digitalWrite(13, ledState);
}
```

Notes

We now have control.

Challenge

You can check if it is working by changing the ledState to HIGH and uploading it again.

Sketch A6.12 a constant interval

We want the LED to blink on/off for one-second intervals, so we have a variable called interval, and we will make it a constant (ie fixed) value of 1000. const is short for constant and means it can never be changed accidentally.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, ledState);
}
```

🗒️ Notes

Nothing to change yet.

🛠️ Code Explanation

`const int interval = 1000;`	Data types: const short for constant and cannot be changed, int short for integer, float has a decimal point

Data types

You will come across different data types, which are different ways of expressing data or information. They are very important features of coding a microcontroller.

`const`	Short for constant eg pin number
`int`	Short for integer eg 3
`float`	Has a decimal point eg 3.76
`long`	Can use much bigger numbers
`unsigned`	Can only be a positive number
`char`	A type of string (text)

Sketch A6.13 unsigned long

The Arduino starts counting milliseconds as soon as the sketch starts running with the `millis()` function. We can use that to check how much time has elapsed since a particular moment in time. So we need to get the number of milliseconds that have elapsed since a particular point in time.

This number is going to get very big very quickly, and we don't want it to become negative for any reason. So the data type we use is called `long`, and we can make sure it is always positive by defining it as `unsigned`.

Our variable will be called `currentMillis`.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  digitalWrite(13, ledState);
}
```

Notes

A lot to take in for one line of code.

Sketch A6.14 the previous number

We also need another similar variable to hold the previous number of milliseconds. We will call this `previousMillis()`. As the sketch runs, we will subtract the `currentMillis()` from the interval and compare it to the `interval` with an `if()` statement.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  digitalWrite(13, ledState);
}
```

Notes

This is a reference point: `previousMillis`.

Sketch A6.15 checking the difference

We need an `if()` statement to check the difference and reset the `previousMillis()`.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;
  }
  digitalWrite(13, ledState);
}
```

Notes

It compares the `interval` to the difference between the current and previous `millis()` values.

Sketch A6.16 toggle the state

And now to change the state depending on whether it is LOW to become HIGH or HIGH to become LOW. It toggles between them. That is why we need to keep track of the state of the LED; if it is off, then we want it on; if on, then off.

Arduino sketch

```
int ledState = LOW;
const int interval = 1000;
unsigned long previousMillis = 0;

void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval)
  {
    previousMillis = currentMillis;
    if (ledState == LOW)
    {
      ledState = HIGH;
    }
    else
    {
      ledState = LOW;
    }
  }
  digitalWrite(13, ledState);
```

```
}
```

🗒️ Notes

It should blink for one second. Basically, it checks the state every `1000` milliseconds (interval length), counts until that is true.

🌻 Challenge

Change the `interval`.

🛠️ Code Explanation

`if ... else`	Another way of using an if statement where there is more than one condition.