E. E. S. T. Nº 5 – TIGRE

ELECTRÓNICA

SISTEMAS DE CONTROL

Arduino

TRABAJO PRÁCTICO Nº 6

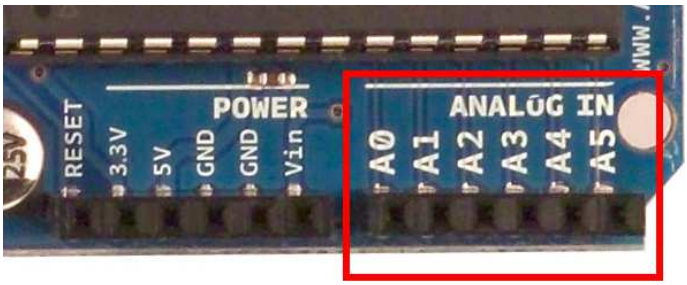
TEMA: Entradas analógicas y salidas PWM.

Operadores aritméticos y lógicos.

ALUMNO:	**Curso:**
FECHA DE ENTREGA:	
FECHA DE APROBACIÓN:	
CALIFICACIÓN:	FIRMA:

Pines de entrada analógica

El Arduino **UNO** posee **seis** entradas **analógicas** (A0..A5) identificadas en la placa con la leyenda **ANALOG IN** que están ubicadas al lado de los pines marcados como POWER.



El uso principal de estos pines es para la lectura de sensores **analógicos**.

El microcontrolador Atmega que usa Arduino lleva incluido un conversor **analógico-digital** (A/D) de 6 canales. Tiene una resolución de **10 bits**, retornando enteros desde 0 a 1023.

Esto quiere de decir que si aplicamos una tensión entre 0 y 5V (sin superar los 40mA de corriente) el conversor lo **transformará** en un número entero entre 0 (que representaría **0V**) y 1023 (que representaría los **5V**). Cualquier valor intermedio de tensión se traduce en un valor entero **intermedio**.

Estos pines tienen **también** toda la funcionalidad de los pines de **entrada-salida** de propósito general (GPIO) (al igual que los pines 0 - 13).

Consecuentemente, si un usuario **necesita** más pines de propósito general de entrada-salida, y no se está usando ningún pin analógico, estos pines **pueden usarse** como GPIO.

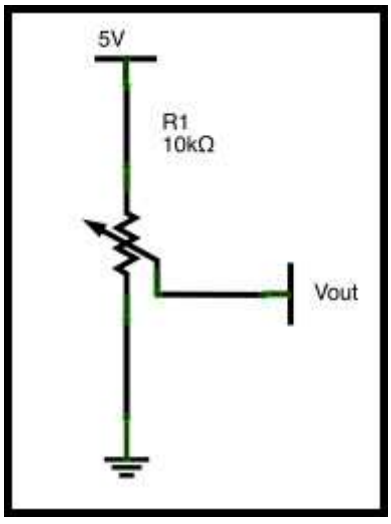
Los pines de Arduino correspondientes a los pines **analógicos** van desde el **14 al 19** y pueden usarse de manera **idéntica** que los digitales, así que por ejemplo, vemos un código como para configurar uno de los pines analógicos como **digital** y establecerlo a HIGH:

```
pinMode(14, OUTPUT);
digitalWrite(14, HIGH);
```

Mientras que si lo utilizamos para su uso natural, como pin **analógico** de **entrada**, **no** es necesario configurarlo.

Hardware

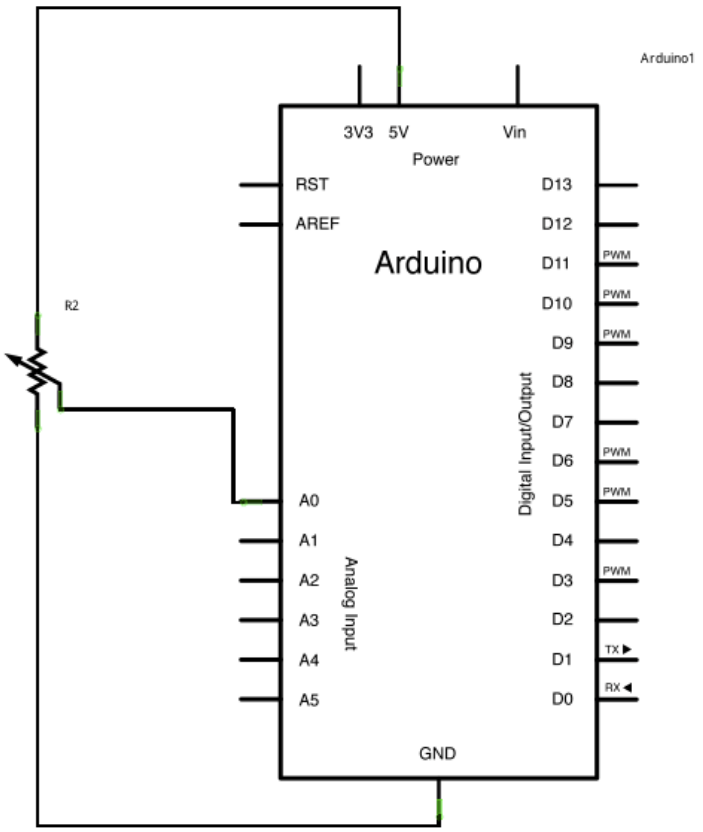
Una de los métodos más sencillos para probar las entradas analógicas en la utilización de un **potenciómetro**, que es una resistencia **variable**.

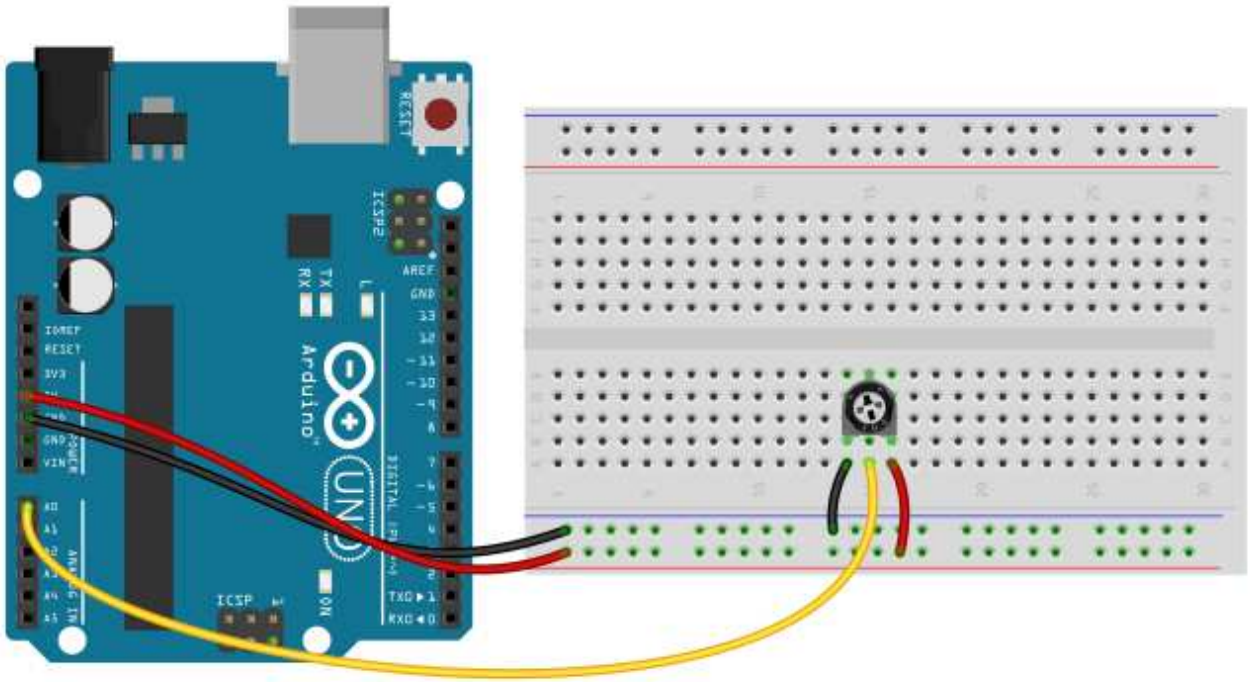


La configuración de esta resistencia es la de un **divisor** de tensión. Veamos cómo se conecta para que esto sea posible.

La imagen de la derecha nos muestra que debemos colocar uno de los extremos del potenciómetro a **masa**, el otro extremo a **5V** y la patilla **central,** que será nuestra tensión de salida regulable, la conectaremos a la **entrada analógica** del Arduino y que variará de 0v a 5v dependiendo de la **posición** de nuestro potenciómetro.

Las siguientes imágenes nos muestran como conectarlo a la placa Arduino.





Software

Vamos a hacer uso de otro ejemplo del entorno de desarrollo. Se trata del sketch ***AnalogInput***. Para cargarlo debe ir a ***Archivo – Ejemplos – 03.Analog – AnalogInput***.

Como ya hemos dicho en otras oportunidades al principio del mismo nos informa sobre el objetivo del sketch: ***"Muestra la entrada analógica mediante la lectura de un sensor conectado al pin A0 y enciende y apaga un diodo emisor de luz (LED) conectado al pin digital 13. La cantidad de tiempo que el LED estará encendido y apagado depende del valor obtenido por analogRead ()."***

```
/*
  Analog Input
Demonstrates analog input by reading an analog sensor on analog pin 0 and
turning on and off a light emitting diode(LED)  connected to digital pin 13.
The amount of time the LED will be on and off depends on
the value obtained by analogRead().

The circuit:
* Potentiometer attached to analog input 0
* center pin of the potentiometer to the analog pin
* one side pin (either one) to ground
* the other side pin to +5V
* LED anode (long leg) attached to digital output 13
* LED cathode (short leg) attached to ground

* Note: because most Arduinos have a built-in LED attached
to pin 13 on the board, the LED is optional.


Created by David Cuartielles
modified 30 Aug 2011
By Tom Igoe

This example code is in the public domain.

http://arduino.cc/en/Tutorial/AnalogInput

*/
```

En la **explicación** del circuito especifica que se conecta el pin **central** del potenciómetro a la entrada analógica (**A0**) y los extremos del mismo, uno a 5V y el otro a tierra.

Como la mayoría de las placas Arduino tienen un LED incorporado conectado al pin 13, **no es** necesario conectarlo en forma externa.

A continuación se definen las **variables**, aunque técnicamente los pines **deberían** ser constantes, va a funcionar igual.

```
int sensorPin = A0;    // select the input pin for the potentiometer
int ledPin = 13;      // select the pin for the LED
int sensorValue = 0;  // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}
```

La variable **`sensorPin`** identifica al pin de entrada **analógica** (**A0**) en la que está conectado el potenciómetro.

La variable **`ledPin`** identifica al pin **digital** del LED (**pin 13**).

La variable **`sensorValue`** es la que va a **guardar** el **valor** leído desde el **sensor**. Esta es la única que va a **variar** su contenido durante la ejecución del sketch.

Por último vemos que en **`void setup()`** se configura el pin donde está conectado el LED como **salida**. Como hemos dicho la entrada **analógica no es necesario** configurarla ya que siempre es de entrada.

El programa principal nos muestra todo el proceso que se va a llevar a cabo.

```
void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
  delay(sensorValue);
  // turn the ledPin off:
  digitalWrite(ledPin, LOW);
  // stop the program for for <sensorValue> milliseconds:
  delay(sensorValue);
}
```

La primera instrucción es la **lectura** del pin analógico y la actualización del contenido de la variable **`sensorValue`**. Para ello se hace uso de la función **`analogRead`**.

`analogRead()`

Descripción

Lee el valor de tensión en el pin analógico especificado. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de 5 voltios / 1024 unidades, es decir, 0.0049 voltios o dicho de otra manera 4.9 mV por unidad. El rango de entrada puede ser cambiado usando la función **`analogReference().`**

El conversor tarda aproximadamente 100 microsegundos (0.0001 segundos) en leer una entrada analógica por lo que se puede llevar una tasa de lectura **máxima** aproximada de 10.000 lecturas por segundo.

Sintaxis

`analogRead(pin);`

Parámetros

pin: indica el número del pin de la entrada analógica que deseamos leer (0 a 5 en la mayoría de las placas, de 0 a 7 en las Mini y Nano y de 0 a 15 en la Mega).

Devuelve

Esta función **devuelve** un valor **entero** entre 0 y 1023 que representa en nivel de tensión en la entrada analógica.

Las siguientes instrucciones del sketch analizado ya les debe resultar familiares.

Luego de la lectura el valor del potenciómetro se **enciende** el LED.

Utilizando el valor digital convertido se hace una demora mediante la función **`delay`**: cuanto más **alto** sea el valor de **tensión** leído **mayor** será el **tiempo** en que el LED permanecerá encendido.

Luego de **apagar** el LED, se **vuelve** a usar **`delay`** para hacer otra **pausa**.

En síntesis, mediante el potenciómetro se va a **variar** la frecuencia de parpadeo.

Actividades

EJ-0601) Conecte en el protoboard el **potenciómetro** de acuerdo al modelo de hardware mostrado y compruebe el funcionamiento del sketch ***AnalogInput***.

EJ-0602) Conecte **tres LEDS** (verde, amarillo y rojo) y un **potenciómetro** al protoboard. Escriba un programa que muestre encendido:

- Solamente el LED **verde** si en la entrada analógica hay valores de tensión **bajos**.
- Solamente el LED **amarillo** con niveles tensión **intermedios**.
- Solamente el LED **rojo** cuando los valores de tensión sean **altos**.

Pista: seguramente deberá utilizar la instrucción **`if`** junto con algún operador **booleano**.

Operadores booleanos o lógicos

Se pueden usar dentro de operaciones condicionales o en una sentencia if.

&& (AND lógico): Resultado verdadero sólo si **ambos** operadores son Verdaderos

Ejemplo 1

```
if (digitalRead(2) == HIGH  && digitalRead(3) == HIGH) { // lee dos pulsadores
  // ...
}
```

Es Verdadero sólo si **ambas** entradas estás activadas, es decir, en estado HIGH.

Ejemplo 2

```
if (a >= 10 && a <= 20){}   // Verdadero sólo si el valor de a está entre 10 y 20
```

|| (OR lógico): Resultado verdadero si **alguno** de los dos operadores es Verdadero.

Ejemplo 3

```
if (digitalRead(2) == HIGH  || digitalRead(3) == HIGH) { // lee dos pulsadores
  // ...
```

Es Verdadero sólo **alguna** de las entradas estás activadas, es decir, en estado HIGH.

Ejemplo 4

```
if (x > 0 || y > 0) {
  // ...
}
```

Es Verdadero si alguno de los valores de x ó y es mayor que 0.

! (NOT): Resultado verdadero si el operador es Falso, por ejemplo:

```
if (!x) {
  // ...
}
```

Es Verdadero si x es Falso (p. e. si x es igual a 0). Para este lenguaje de programación el 0 puede interpretarse como falso y un valor diferente a 0 como verdadero.

Salidas PWM

En ocasiones es necesario algo más que una señal de 0 o 1 en nuestros proyectos, por ejemplo para modificar la **velocidad** de giro de un motor, para variar la **intensidad** con la que luce un diodo, para **transmitir** los grados de giro de un servomotor, etc.

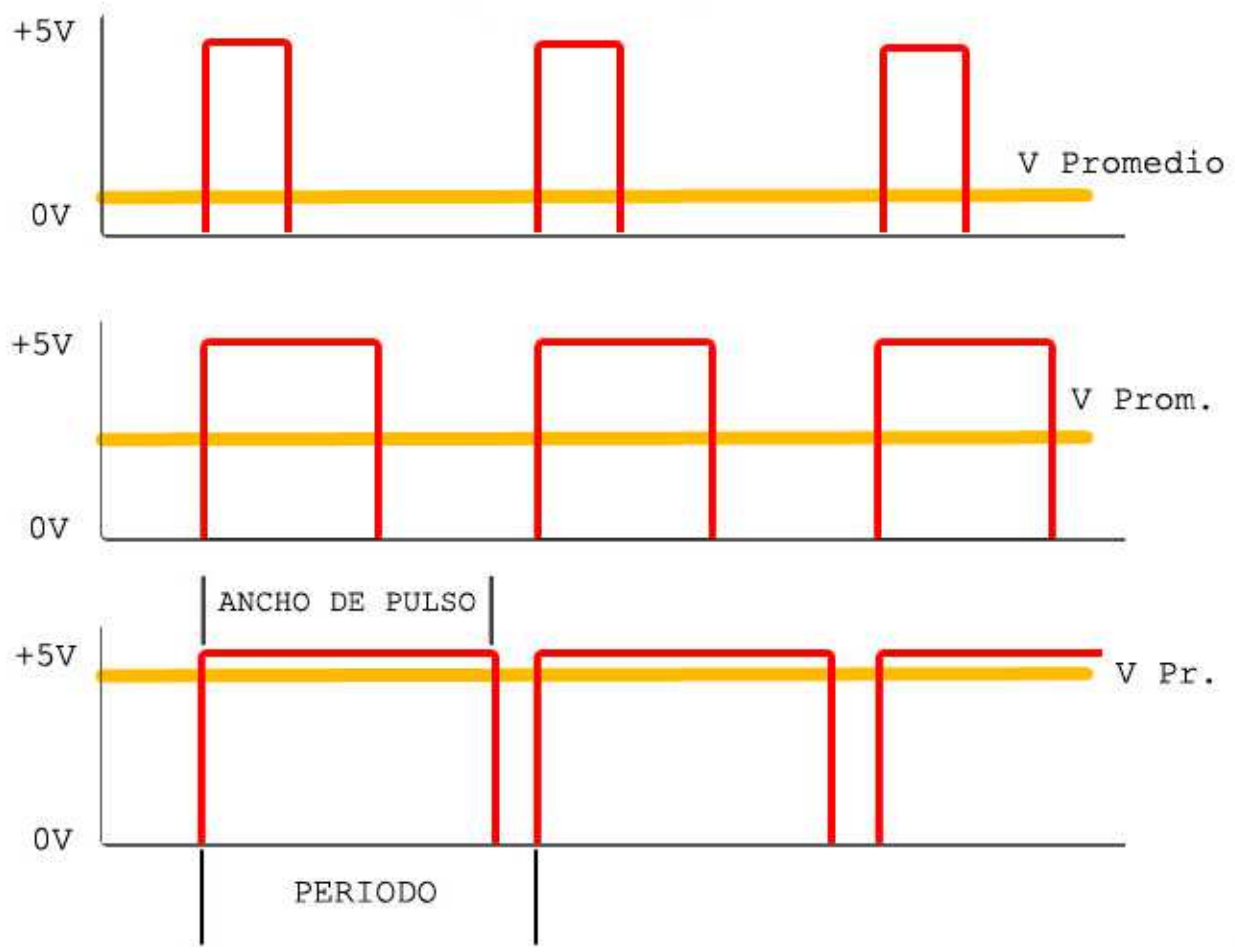
Para todo esto, y mucho más, nos servirá el **PWM**, que **emula** una señal analógica a partir de una señal digital.

La sigla **PWM** viene de Pulse Width Modultation, o **Modulación de Ancho de Pulso**.

Lo que hace este tipo de señal es emitir, en lugar de una señal continua en nuestra salida, una **serie** de **pulsos** que es posible variar en su duración pero siempre con una frecuencia **constante** de aproximadamente 490 Hz.

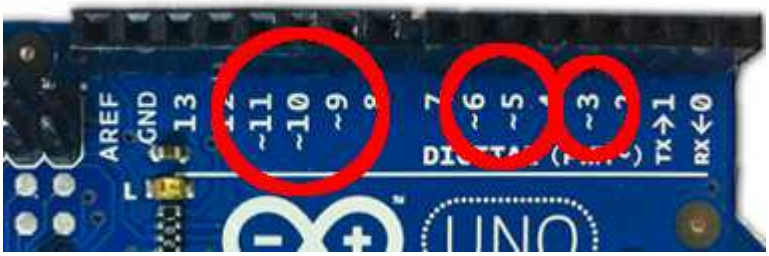
De esa manera la tensión **promedio** resultante, es directamente **proporcional** a la **duración** de los pulsos, es decir, cuanto más **juntos** estén esos pulsos de +5v, **mayor** será la tensión promedio de nuestra salida, y cuanto más **distantes** sean estos, **menor** será dicha tensión.

Las siguientes imágenes **ilustran** este efecto. Observe como el **ancho** que tiene cada pulso (en rojo) se relaciona con el **nivel** de la señal **promedio** (en naranja) y como el período de la señal permanece **inalterable**.



El Arduino UNO posee **seis** salidas que pueden utilizarse como **PWM**. Están rotuladas con los números 3, 5, 6, 9, 10 y 11.

Todas ellas tienen una pequeña onda **~** que las identifica.



Hardware

Para probar una de las salidas PWM vamos a **variar** la intensidad lumínica de un LED. Y como a esta altura ya sabe conectarlo a la placa Arduino, con el cuidado de limitar la corriente, solo le dejo la siguiente consigna:

Conecte un LED con la resistencia adecuada al pin 9 del Arduino

Software

Vamos a analizar el sketch ***Fade*** que se encuentra en ***Archivo – Ejemplos – 01.Basics – Fade.***

```
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup()  {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}
```

Este ejemplo **muestra** como **varía** la intensidad lumínica de un LED conectado al pin **9**.

Al principio del programa son declaradas tres variables:

- **`led`** que se **inicializa** con el valor **`9`** y representa el **pin** donde está el LED.
- **`brightness` inicializada** en `0` que representará el **brillo** lumínico del LED.
- **`fadeAmount` inicializada** en `5` que representa en cuanto se incrementará o disminuirá el **desvanecimiento** lumínico del LED.

Por último vemos que mediante **`pinMode`** se configura el pin del led como **salida** sin otra aclaración. Tenga en cuenta que solamente **6 pines** pueden usarse como salida **PWM** y uno de ellos es el pin 9.

```
// the loop routine runs over and over again forever:
void loop()  {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness == 0 || brightness == 255) {
    fadeAmount = -fadeAmount ;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

El código principal resulta muy interesante porque además de utilizar **nuevas funciones** nos muestras diferentes **trucos** para resolver situaciones que le puede resultar **útiles** para futuros proyectos.

En primer lugar tenemos la función **`analogWrite`** utilizada de esta manera:

```
// set the brightness of pin 9:
analogWrite(led, brightness);
```

Veamos en **detalle** que hace esta función.

`analogWrite()`

Descripción

Escribe un valor **analógico** (**PWM**) en un pin. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor. Después de llamar a la función **`analogWrite`**(), el pin generará una onda **cuadrada** estable con el ciclo de trabajo especificado hasta que se vuelva a llamar a la función analogWrite() (o una llamada a las funciones digitalRead() o digitalWrite() en el mismo pin). La frecuencia de la señal **PWM** será de aproximadamente **490 Hz**.

En la mayoría de las placas Arduino (ATmega168 o ATmega328), se podrá generar señales PWM en los pines 3, 5, 6, 9, 10, y 11. En la placa Arduino Mega, se puede llevar a cabo con los pines desde el 2 hasta el pin 13.

Sintaxis

```
analogWrite(pin, valor);
```

Parámetros

pin: indica el **número** del pin en el que se quiere generar la salida **PWM**.

valor: Es el ciclo de trabajo deseado. Es un valor entero comprendido entre 0 (siempre apagado) y 255 (siempre encendido).

Devuelve

Nada.

Esta función nos permitirá **emular** una señal analógica a partir de una salida digital. En otras palabras la salida se va a comportar como si en vez de haber solamente 0V o 5V **hubiera** también valores **intermedios**.

Como el segundo valor de esta instrucción puede variar de 0 a 255, los valores promedio de salida va a ir desde 0 = 0V hasta 255 = 5V Por lo tanto, si quisiéramos conseguir un valor de tensión promedio que sea intermedio, como por ejemplo 2,5V, el valor a asignar también debería ser **intermedio**, por ejemplo, 127.

La siguiente instrucción involucra el operador **aritmético +**. Los operadores aritméticos producen resultados aritméticos, es decir, valores **numéricos** y son:

Operadores Aritméticos

= (asignación)

+ (suma)

- (resta)

* (multiplicación)

/ (división)

% (resto)

¿Qué es lo que hace la instrucción…

```
// change the brightness for next time through the loop:
brightness = brightness + fadeAmount;
```

?

La traducción sería: *El **nuevo valor** de **`brightness`** será el que tiene actualmente (**`brightness`**) **más** el valor de **`fadeAmount`**.*

Para que esta instrucción funcione, tanto **`brightness`** como **`fadeAmount`** tienen que tener valores **iniciales** conocidos. Y lo tienen, ya que al principio **`brightness`** vale `0` y **`fadeAmount`** vale `5`. El siguiente cuadro muestra lo que pasa con esta instrucción cuando se **repite** varias veces:

Valores actuales de…		Se ejecuta la instrucción…	Nuevos valores de…	
`brightness`	**`fadeAmount`**		**`brightness`**	**`fadeAmount`**
`0`	`5`	**`brightness = brightness + fadeAmount;`**	`5`	`5`
`5`	`5`	**`brightness = brightness + fadeAmount;`**	`10`	`5`
`10`	`5`	**`brightness = brightness + fadeAmount;`**	`15`	`5`

Y así sucesivamente.

Observe que una misma instrucción produce un **incremento** de la variable **`brightness`** en 5 unidades cada vez que se ejecuta. En cambio, el valor de la variable **`fadeAmount`** no es modificado.

Solo se modifica el valor de una variable si está a la izquierda del operador =.

La siguientes instrucción que se ejecuta dentro de **`void loop()`** es:

```
// reverse the direction of the fading at the ends of the fade:
if (brightness == 0 || brightness == 255) {
  fadeAmount = -fadeAmount ;
}
```

Aquí tenemos dos cuestiones. Una de ellas es lo que encierra el paréntesis de la instrucción **`if`** y la otra lo que se va ejecutar cuando el **`if`** sea verdadero.

La interpretación de la instrucción if sería:

*si **`brightness`** vale `0` **o** **`brightness`** vale 255*.

Aquí se usa el operador lógico || (OR) que, como dijimos en páginas anteriores, arroja un resultado **verdadero** cuando **alguno** de sus operandos sea verdadero.

Como hemos explicado, durante la ejecución repetida del programa, el valor de **`brightness`** se va **incrementando**. Si no hacemos nada, ese valor crecerá hasta el **límite** de la variable **`brighteness`** que, como es de tipo **`int`**, el valor máximo que alcanzaría será 32767. Como nosotros **no queremos** que suceda esto, le ponemos un límite: 255. Porque ese es el valor **máximo** que podemos usar en la instrucción **`analogWrite`**.

Entonces cuando **`brightness`** llegue al valor 255 lo encerrado en el paréntesis del **`if`** será verdadero y se ejecutará lo que está dentro de las llaves, que es:

```
fadeAmount = -fadeAmount ;
```

¿Y qué hace esta instrucción? **Respuesta**: modifica el valor de **`fadeAmount`**.

Si `fadeAmount` vale 5 (como hasta ahora) fadeAmount pasará a valer – 5.

Con lo cual a partir de aquí la instrucción

```
brightness = brightness + fadeAmount;
```

Irá **disminuyendo** el valor de **`brightness`** de 5 en 5. ¿Entiende el truco?

Cuando **`brightness`** alcance el valor `0`, nuevamente la instrucción `if` va a ser verdadera y por lo tanto se ejecutará nuevamente la operación:

```
fadeAmount = -fadeAmount ;
```

Y ahora como **`fadeAmount` vale `-5`, el nuevo valor de `fadeAmount` pasará a ser `5`.**

Y así seguirá, aumentando y disminuyendo sucesivamente la variable **`brightness`** que, como es utilizada en **`analogWrite`**, provocará que cambie el **brillo** del LED.

Actividades

EJ-0603) Conecte el LED al Arduino de acuerdo al modelo de hardware mostrado y compruebe el funcionamiento del sketch ***Fade***.

EJ-0604) Agregue el **potenciómetro** y escriba un sketch para controlar el brillo del LED con él. Tenga en cuenta que los valores que entrega **`analogRead`** van desde 0 a 1023 y los que necesita para **`analogWrite`** van de 0 a 255. Así que no podrá usar directamente esos valores. Tendrá que hacer alguna operación **aritmética** para adaptar esos valores.