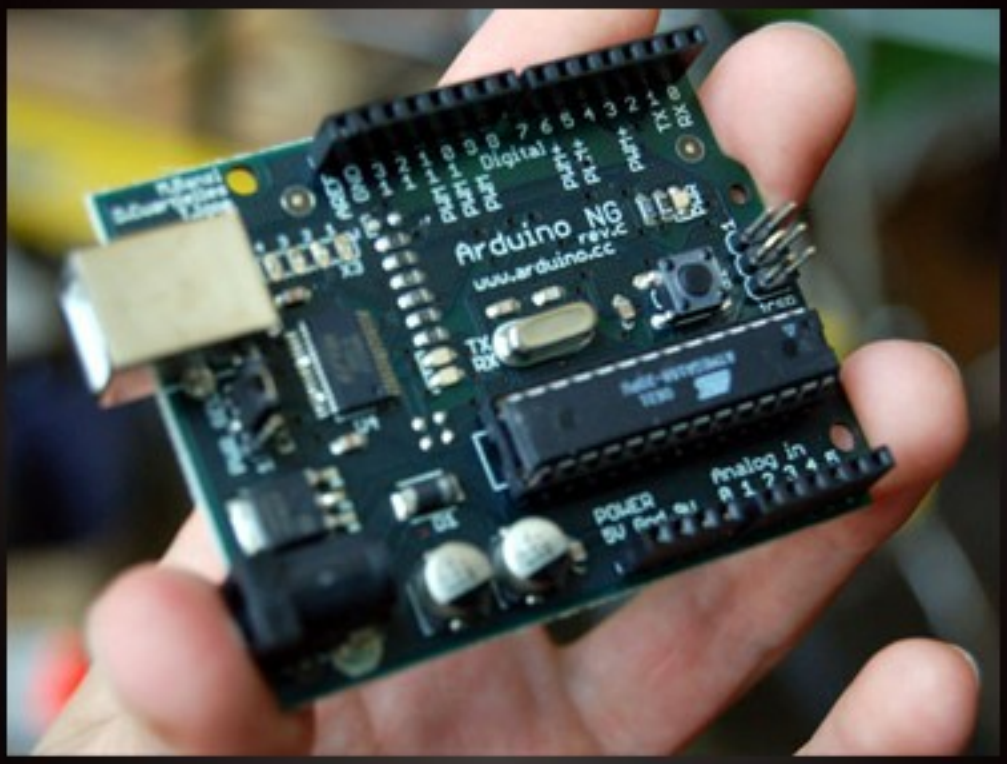
# Hack x Crack

# Arduino




Diseño hecho por: Hacker Fashion

www.hackxcrack.es

# Cuaderno creado por BlindOwl y Tarmo

# Índice

# 1- Presentación de la plataforma Arduino:

Muchas veces hemos escuchado la palabra “Arduino”, pero...¿Que es Arduino?, ¿Como surgió?, ¿Para que se utiliza?

Arduino es una plataforma de desarrollo de computación basada en una placa electrónica y un entorno de programación encargado de crear el software para dicha placa, compilarlo y grabarlo en el microcontrolador de esta.
Tanto la placa como el entorno de programación son Open Source, de esta forma tenemos acceso al código del IDEArduino y a los esquemas del circuito de la placa que podremos montar nosotros mismos si nos lo planteamos y disponemos de los componentes y herramientas necesarias.

Arduino nace de la necesidad de un dispositivo capaz de interactuar entre el mundo físico que nos rodea y el mundo virtual de los ordenadores, creando así una herramienta con mucho potencial para el desarrollo de proyectos electrónicos controlados, máquinas electrónicas y todo tipo de aparatos que nos podamos imaginar.

# 2- Instalación del entorno de programación IdeArduino en Linux y Windows.

## ·*Instalando y configurando IDEArduino en Linux:*

En la mayoría de las distribuciones Linux podemos instalar el IDEArduino ejecutando desde la consola el simple comando : **apt-get install arduino**
Es la forma mas práctica y fácil de instalarlo pero tiene el inconveniente que la versión de los repositorios esta algo anticuada y algunas placas no las reconoce adecuadamente, por lo tanto explicare como instalarlo desde el paquete oficial, así podremos tener IDEArduino totalmente actualizado.

Antes de nada debemos tener instalado java en nuestro sistema para posteriormente poder correr el IDEArduino. Si no sabes como instalarlo sigue este tutorial http://www.java.com/es/download/help/linux_install.xml

Si no estas seguro de si lo tienes instalado o no, abre una terminal y teclea: java -version y si lo tienes instalado aparecerá la versión que tienes.

Lo segundo que haremos será descargarnos el programa en su última versión de la página oficial http://arduino.cc/es/Main/Software, actualmente se encuentra en la versión 1.0.1.

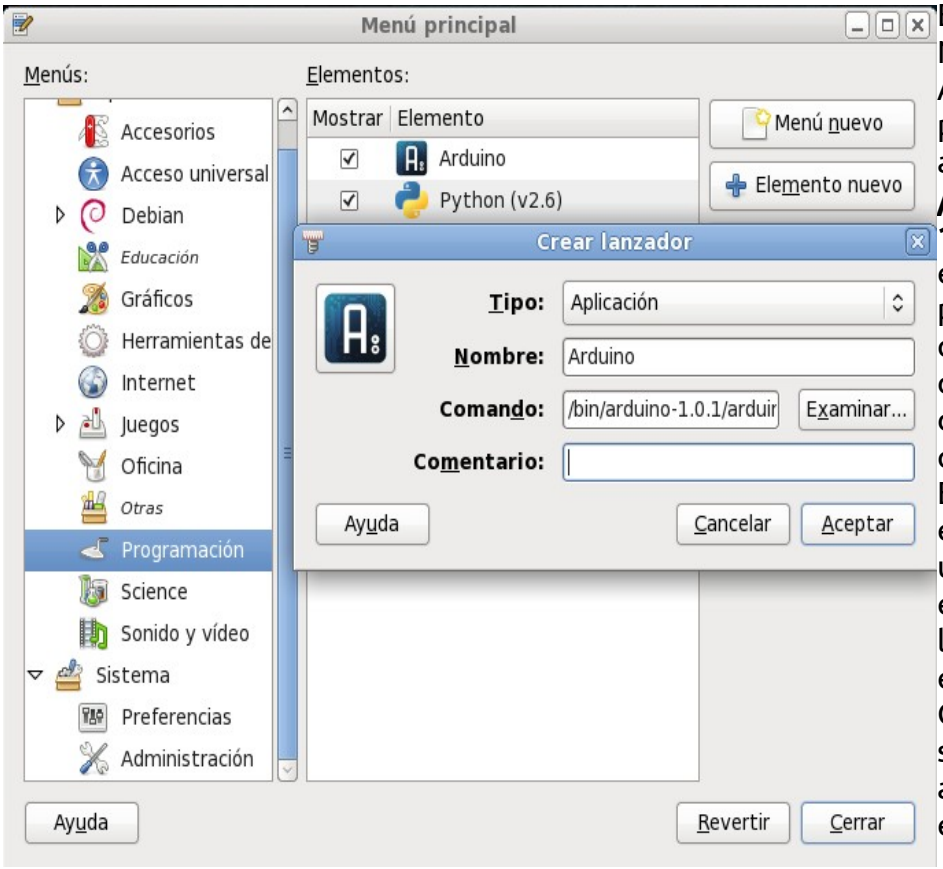
Descargamos el paquete y lo guardamos en el escritorio.

Abrimos una terminal y le pasamos la ruta del Escritorio partiendo de la raíz (esto depende del sistema de ficheros propio), en mi caso:
**cd /home/BlindOwl/Escritorio**

Descomprimimos el contenido del paquete con: **tar -xvzf arduino-1.0.1-linux.tgz**


Se extraerá todo el contenido en el escritorio en una carpeta llamada arduino-1.0.1 que deberemos cortarla y pegarla en **/bin**, o en otro directorio que queramos.

Ahora hacemos click derecho encima de Aplicaciones y en Editar Menús, se abrirá un menú principal con el que podemos gestionar nuestras aplicaciones.
Vamos a crear un lanzador para nuestro IDEArduino en la sección de Programación que podremos elegir en la parte izquierda del menú, pulsamos Elemento Nuevo y se abre una ventana para crear el lanzador.




En esta ventana en Nombre pondremos Arduino, en comando le pasamos la ruta del archivo, en este caso: **/bin/arduino-1.0.1/arduino** en comentario podemos poner lo que queramos, una descripción, etc, yo lo dejo en blanco que queda mas curioso.
En la parte izquierda de esta ventana aparece un recuadro para elegir el icono especificando la ruta donde se encuentra.
Cuando tengamos todo seleccionado aceptamos y se creará el lanzador.

Configuración de IDEArduino con la placa:

Conectamos la placa al pc y abrimos el programa, en **Herramientas → Tarjeta**, seleccionamos la tarjeta que tengamos, en el caso de este tutorial escogemos Arduino Uno, en **Herramientas → Puerto Serial** debemos tener marcado algo como /dev/ttyACM# o /ttyUSB# (puede variarel numero en #), por ultimo en **Herramientas → Programador** seleccionamos AVRISP mkll.

Ahora ya deberíamos poder comunicarnos con la placa, para asegurarnos cargaremos un programa y nos fijaremos si los leds TX y RX parpadean al grabarlo en el microcontrolador.

### ·*Instalando y configurando IDEArduino en Windows:*

Se da por sentado que ya tenemos instalado java, si no es así hay que descargar el ejecutable desde la pagina oficial e instalarlo http://www.java.com/es/download/

Descargamos el programa en su última versión de la página oficial http://arduino.cc/en/Main/Software y lo descomprimimos en el pc, se creara una carpeta arduino-1.0.1, la abrimos y ya podemos abrir nuestro entorno de programación.

Para instalar el driver de la placa abrimos el Administrador de dispositivos (con la placa conectada) y nos aparece en el listado **Otros dispositivos → Dispositivo desconocido**, hacemos click encima de el y en la pestaña "Controlador" pulsamos "Actualizar controlador" y después "Buscar software de controlador en el equipo", especificamos la ruta de la carpeta drivers que se encuentra dentro de la arduino-1.0.1 que descomprimimos y aceptamos, automáticamente se instala el controlador.

Abrimos el entorno de programación y en **Herramientas → Puerto Serial** seleccionamos el puerto COM# que nos aparece en el administrador de dispositivos cuando instalamos el driver.

Probamos a grabar un programa de prueba y nos fijamos si parpadean los leds TX y RX, si lo hacen y no sale ningún error en el IDEArduino esta todo correcto.

# 3- Programación y prácticas.

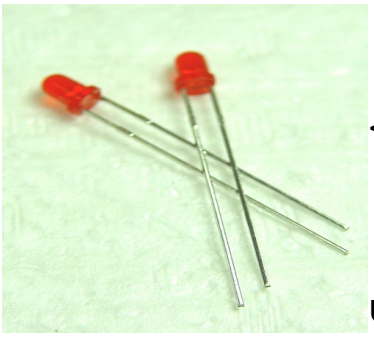
Una importante característica de arduino es su facilidad en el tema de la programación ya que utiliza una especie de lenguaje C simplificado donde muchas funciones han sido eliminadas creando un lenguaje muy simple pero muy versátil gracias a la posibilidad de añadir librerías según los diferentes proyectos en los que quieras embarcarte.

Para empezar con la programación en arduino empezaremos con el parpadeo de un diodo LED (el hola mundo de arduino) para familiarizarnos un poco con la estructura y luego pasar a un ámbito mas teórico.

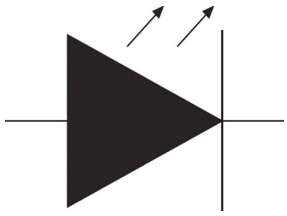
-Una cosa¿Qué es un diodo LED?
Un diodo led es un tipo de diodo que ha sido contaminado en su proceso de montaje para que al pasar corriente eléctrica emita luz.

-Aaaah mas o menos como una bombilla
No exactamente, ya que los diodos tienen una polarización ,es decir que no da igual su conexión con la pila ya que la patilla larga debe ir al polo positivo de la pila y la patilla corta a tierra (Ánodo y cátodo).

-Ok me fijaré en la conexión del diodo

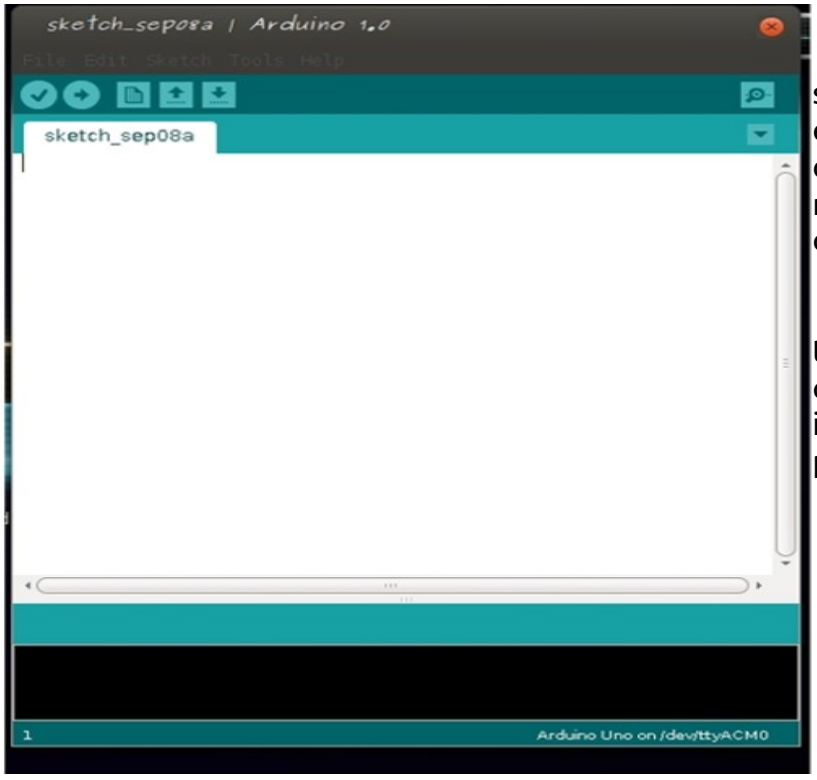


<----------Diodos Led Rojo



Una vez aclarado que es un diodo empecemos.

-Abrimos Arduino IDE


Nos saldrá una ventana similar a esta, aquí es donde escribiremos el programa (las diversas acciones que realice nuestra placa con los distintos componentes electrónicos).

Una vez abierto el programa tenemos que saber que hace cada botón que vemos en la interfaz y como se mete el programa en la placa arduino.

***Fíjense en la imagen de abajo (la de los botones numerados)***


El botón 1 lo que hará es comprobar si la sintaxis del programa es correcta.

El botón 2 sube el programa (si es correcto) a nuestra placa arduino que debe estar conectada al ordenador con un cable como este:
*(El USB tipo A al ordenador y el tipo B conectado a nuestra placa)*

El botón 3 es útil para cuando los programas empiezan a ser más largos, lo que hace es abrir una nueva hoja para seguir con el programa.
El botón 4 sirve para abrir ejemplos de programas con las diversas librerías.
El botón 5 sirve para guardar nuestro programa en el ordenador.
El botón 6 abre una ventana en la que se pueden ver/interactuar en aquellos programas que establecen una comunicación directa con el ordenador.

-Un momento
-¿Qué pasa?
-¿Qué es eso de las librerías?
Muy buena pregunta aunque estamos retrasando bastante el parpadeo de nuestro diodo led.

Una cosa muy importante en arduino son las librerías dicho de manera sencilla son las que hacen que nuestra placa pueda realizar las diferentes acciones que queremos que desarrolle de una forma mas sencilla, de esta forma reducimos mucho el código.

En la pagina de arduino viene esto sobre las librerías:
*Las Librerías proveen funcionalidad extra a nuestro sketch, por ejemplo: al trabajar con hardware o al manipular datos.*

Arduino de por si trae unas librerías para realizar montajes pero nosotros podemos agregar nuevas librerías según lo que queramos hacer. Por ahora no explicaré como agregar nuevas librerías ya que las que trae por defecto nos bastaran para realizar los proyectos. Una vez dicho todo esto empecemos con nuestro primer programa

```
void setup()
{
  pinMode(13, OUTPUT);
}
void loop()
{
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

-¿Qué es todo esto?
-Jajaja esto son las instrucciones que harán que nuestro diodo led parpadeé

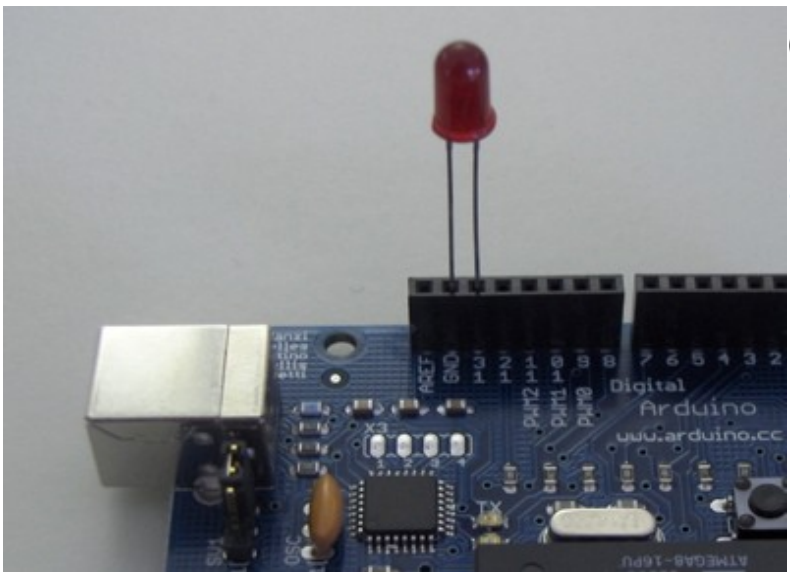
-Pues explica cada cosa que con esto me has matado
En todo programa de arduino hay dos partes:
**void setup()** y **void loop()**

**void setup()** es la parte en la que nosotros diremos a la placa cuales son los pines que debe utilizar y de que modo.
**void loop()** es la parte donde diremos las acciones que tiene que realizar la placa con esos pines.

En este caso en **void setup()** estamos diciendo que el pin 13 de la placa actué como salida.
En **void loop()** estamos usando la función **digitalWrite()** que lo que hace es enviar una señal eléctrica si esta en HIGH y si esta en LOW no enviarla.
Y **delay()** que lo que hace es esperar un tiempo hasta que se realice la siguiente acción en este caso 1000 milésimas de segundo que son 1 segundo (*Esta función se mide en milésimas).*

En definitiva lo que estamos haciendo es enviar una señal eléctrica al pin 13 donde irá conectado el diodo en ese momento se encenderá y al segundo se apagará así cíclicamente.

Así sería la conexión correcta del diodo según lo que hemos escrito. Sólo nos quedaría subir el programa a la placa con el botón 2(*Imagen anterior*) conectar el diodo correctamente y esperar resultados.




(*La patilla larga del diodo en el pin 13)*
*Si no lo entendiste tranquilo que abajo viene el verdadero comienzo del tutorial.*

# Buen Comienzo!!

Después de haber visto nuestro primer programa y familiarizarnos un poco, vamos a empezar por donde tal vez deberíamos haber comenzado.

Empecemos analizando funciones usuales dentro de void setup()

-**pinmode(pin,modo);** pin es la ranura que vas a utilizar de tu placa arduino para tus proyectos (*OJO, no todos los pines tienen la misma función unos son de entrada y otros de salida*).
Por un lado está pin y por el otro modo que será Input *(entrada)* o output *(salida)*
Input para detectar estados como el de un interruptor si esta encendido o apagado y output para enviar señales a ese pin.

-**Serial.println(value);** por otro lado tenemos otra función que se encargará de mostrar en nuestro monitor un valor en este caso el (value).

Hay mas funciones pero de momento quiero que se entiendan bien estas para abordar contenidos más avanzados

Ahora veremos funciones dentro de void loop()
-**digitalRead(pin);** Lee un valor como puede ser el de un interruptor lee o HIGH o LOW.
-**digitalWrite(pin,value);** En pin se selecciona el pin con el que se trabajará en value envia una señal eléctrica HIGH o LOW

## Representación de valores

A veces quieres usar una variable que contenga un valor ya sea para no usar números y facilitar la lectura del programa o para futuras estructuras y funciones que nos será de utilidad, pues bien Arduino facilita esto gracias a lo siguiente:
Usando una función de la que hablaremos ahora ,dejando un espacio, el nombre que quieres dar a una variable y un igual con el que diremos a que valor numérico queremos asignar arduino ya entiende que esa nueva variable se refiere a tal número.

-Que lio ¿no?
-Bastante xD pero ahora veremos que es muy fácil con un ejemplo

```
void setup()
int led = 13;
{
  pinMode(led, OUTPUT);
}
void loop()
{
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

-Ohh el programa del diodo led pero cambiaste el número 13 por led.
-Exacto ya que ahora led = corresponde al pin 13

Aunque parece una tontería por ahora mas adelante creedme que os será útil

Importante!!!! int es una función que no cubre todos los valores tiene sus límites para otros valores existen otras funciones os las dejaré aquí para que os vayan sonando.

| **Numeric types** | **Bytes** | **Rango** | **Uso** |
|---|---|---|---|
| int | 2 | -32768 a 327676 | Representar valores positivos y negativos |
| unsigned int | 2 | 0 a 65535 | Representar solo valores positivos |
| long | 4 | -214783648 a 2147483648 | Representar valores positivos y negativos muy extensos |
| Unsigned long | 4 | 4294967295 | Representar valores positivos muy extensos |
| float | 4 | 3.4028235E+38 a -3.4028235E+38 | Represetar números decimales |
| double | 4 | Mismo que float | Como float |
| boolean | 1 | False (0) o true(1) | Representar valores verdaderos o falsos |
| char | 1 | -128 a 127 | Representar un solo carácter |
| byte | 1 | 0 a 255 | Similar a char pero para valores no asignados |
| *string | | | Representar conjunto de números tipicamente usado ya que contienen texto |

# Control de flujo

Hay diferentes sentencias con las que podemos hacer los programas más complejos que nos ayudarán a hacer funciones que nos ayudarán en nuestros proyectos. Empecemos...

## if y else

Las sentencias **if** comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa las ignora y ejecuta las sentencias especificadas dentro de **else**.

**Ejemplo con if**

```
if(cualquiervariable ?? valor)
{
hacer algo;
}
```

*// los signos ?? = representan símbolos matemáticos como:  >(mayor que),<(menor que),>=(mayor o igual que),<=(menor o igual que),==(igual que)*

**Ejemplo con else**

```
if(inputPin == HIGH)
{
Haceralgo;                  //En el caso de que el valor de inputPin sea igual a HIGH se ejecuta
}                          // la sentencia Haceralgo

else
{
Hacerotracosa;              //En caso de no serlo, es decir inputPin es igual a LOW se ejecuta la
}                          // sentencia Hacerotracosa
```

Tranquilos que os explico otra sentencia y os pongo un code de referencia ;)

# for

La sentencia **for** se usa para repetir un bloque de declaraciones encerradas en llaves un número específico de veces. Un contador de incremento se usa a menudo para incrementar y terminar el bucle. Hay tres partes separadas por punto y coma (; ), en la cabecera del bucle.

```
for(inicializacion; condicion; expresión)
{
haceralgo;
}
```

La inicialización de una variable local, o contador de incremento, sucede primero y una sola una vez. Cada vez que pasa el bucle, la condición siguiente es comprobada. Si la condición devuelve TRUE, las declaraciones y expresiones que siguen se ejecutan y la condición se comprueba de nuevo. Cuando la condición se vuelve FALSE, el bucle termina.

**Ejemplo con for**

```
for(int i=0; i<20; i++)
{
digitalWrite(13, HIGH);
delay(250);
digitalWrite(13, LOW);
delay(250);
}
```

**Bueno después de esto empecemos con otra práctica**

Primero abre el arduino IDE y como ya sabemos insertaremos el programa y cuando acabemos lo subiremos a la placa como expliqué anteriormente y montaremos el circuito según el diagrama que os he dejado abajo.

**Práctica II**

Con esta práctica se pretende profundizar un poco mas en la programación de arduino realizando el montaje de 4 interruptores que serán controlados cada uno con su interruptor correspondiente.

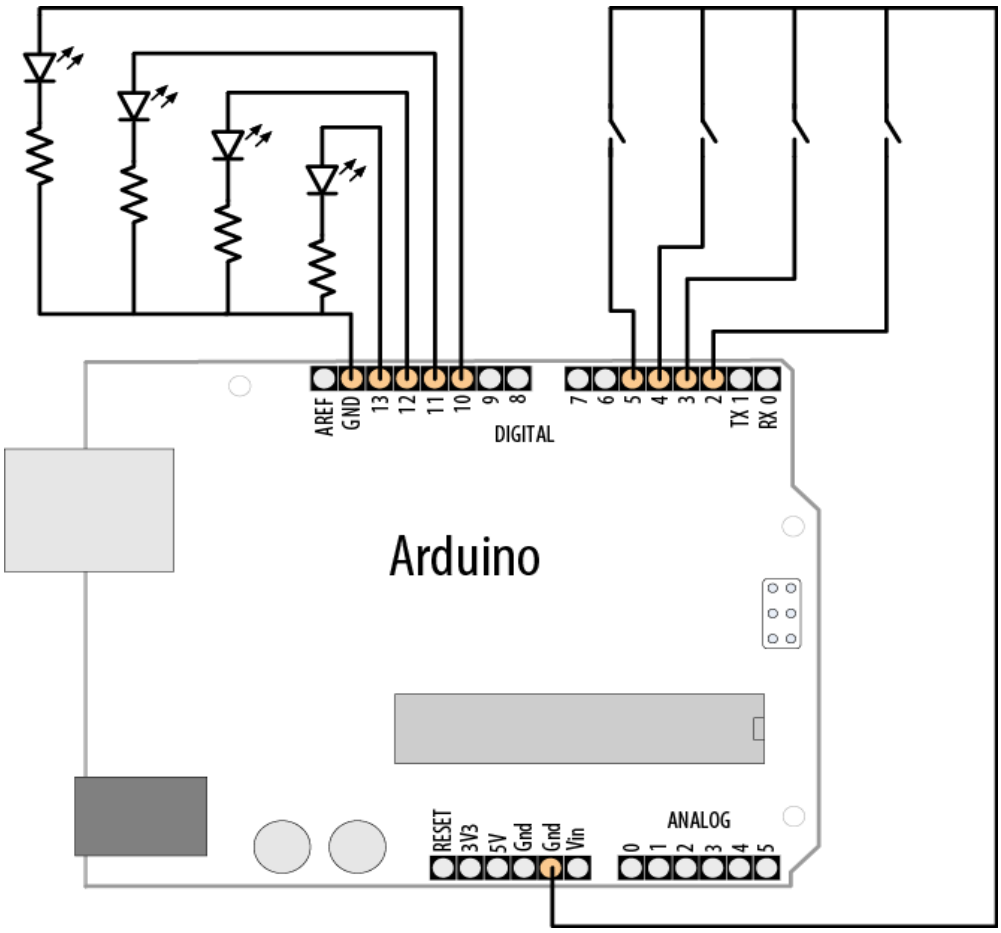
Si activamos el pulsador presionándolo el led se encenderá, sino el led permanecerá apagado.

**Código:**

```
int inputPins[] = {2,3,4,5};
int ledPins[] = {10,11,12,13};

void setup()
{
for(int index = 0; index < 4; index++)
  {
   pinMode(ledPins[index], OUTPUT);
   pinMode(inputPins[index], INPUT);
   digitalWrite(inputPins[index],HIGH);
  }
}

void loop()
{
  for(int index = 0; index < 4; index++)
  {
   int val = digitalRead(inputPins[index]);
   if (val == LOW)
   {
digitalWrite(ledPins[index], HIGH);
   }
else
   {
digitalWrite(ledPins[index], LOW);
   }
  }
}
```




## Explicación del código

Empecemos por las variables **int** , estas contienen un array de pines que la primera (**int inputPins[] = {2,3,4,5};** ) se encargará de los pulsadores y su estado, y el segundo (**int ledPins[] = {10,11,12,13};** )controla la señal de los leds.

Los arrays son colecciones de variables consecutivas del mismo tipo. Cada variable en la colección se llama elemento. El número de elementos se llama la dimensión del array. El ejemplo anterior muestra un uso común de los arrays en el código de Arduino: Aquí los pins se conectan a pulsadores y LEDs . Las partes importantes de este ejemplo son la declaración de la matriz y el acceso a los elementos de la matriz.

En este caso los arrays son de cuatro elementos.
El primer elemento se fija igual a 2, el segundo a 3, y así sucesivamente:

**int inputPins []** = {2,3,4,5};

Esto declara un array de cuatro elementos con el valor inicial de cada elemento puesto a cero. Este array tiene una dimensión de cuatro y puede contener, como máximo, cuatro valores.

El primer elemento del array es el elemento [0]:
**int primerElemento = inputPin[0];** *// este es el primer elemento*
El último elemento es uno menos que la dimensión, por lo que en el ejemplo anterior, con una dimensión de cuatro, el último elemento es el elemento 3:

**int ultimoElemento = inputPin [3];** *// este es el último elemento*
Puede parecer extraño que una matriz con una dimensión de cuatro tiene el elemento de último acceso con arreglo [3], pero debido a que el primer elemento es array [0], los cuatro elementos son:
array [0], array [1], array [2], array [3]

En el esquema anterior, se accede a los cuatro elementos mediante un bucle for:

**for (int index = 0; index <4; index + +)** *// el index pasa el array por el bucle* comprobando que cumpla con las condiciones establecidas.

```
{
// Obtener el número de PIN de acceso a todos los elementos de las matrices de pines
pinMode (ledPins [index], OUTPUT);   // declara LED como salida
pinMode (inputPins [index], INPUT);   // declara pulsador como entrada
digitalWrite(inputPins[index],HIGH);  // activa las resistencias pull-up
}
```

Este bucle se desplazará por el index variable con valores comenzando en 0 y terminando en 3.

Es un error común es acceder accidentalmente un elemento que está más allá de la actual dimensión de la matriz. Este es un error que puede tener muchos síntomas diferentes y deben tomarse medidas para evitarlo.

Una manera de mantener sus arrays bajo control es establecer la dimensión del array mediante el uso de una constante de la siguiente manera:

**const int PIN_COUNT = 4;** *// se define una constante para el número de elementos*

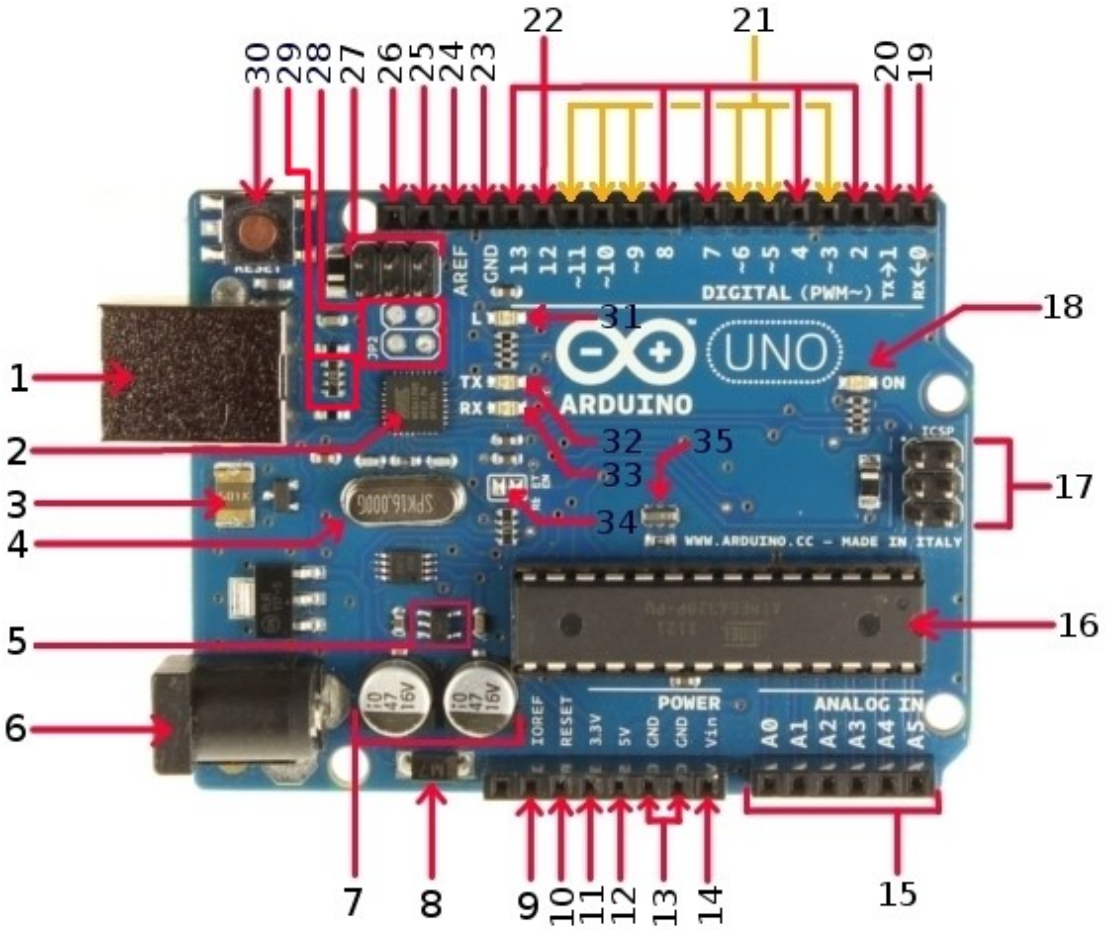
**for (int index = 0; index<PIN_COUNT, index + +)** *//en PIN_COUNT ya ponemos directamente 4 sin declararlo antes, así reducimos código*

El compilador no notificará un error si accidentalmente se trata de almacenar o leer más allá del tamaño del array.

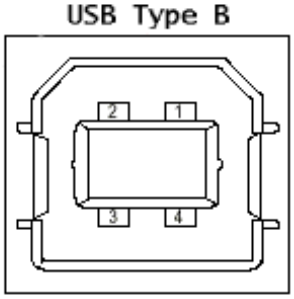
```
{
 for(int index = 0; index < 4; index++) //se realiza de nuevo el bucle
 {
  int val = digitalRead(inputPins[index]); //se lee el estado de los pulsadores
  if (val == LOW)                          //comprueba si el estado del pulsador
  {
digitalWrite(ledPins[index], HIGH);  //enciende el led si el pulsador esta activo (cerrado)
  }
else
  {
digitalWrite(ledPins[index], LOW); //si el pulsador esta (abierto) el led está apagado
  }
 }
}
```

# 4- Descripción de los componentes de la placa Arduino UNO r3.

En este apartado trataré de explicar la mayoría de los componentes de la placa así como algunos detalles y recomendaciones a tener en cuenta.




**1:** Puerto USB tipo B, se utiliza para comunicarse con la placa y tambien para la alimentación de esta si la corriente demandada no excede de 500mA, en el punto 3 veremos porqué.




1=Vbus (5V)

2=D–

3=D+

4=GND

Pin 1=> Alimentación con un voltaje de 5V DC
Pin 2 y 3 => Sirven para la transmisión de datos del BUS
Pin 4 = Masa o tierra

Para conectar la placa al pc debemos utilizar un cable con conexión USB A-B, en un extremo tiene conexión A (para PC) y en otra del tipo B (para la placa).
Este tipo de cable es muy común en algunos periféricos de pc como impresoras y escaners, creo que no nos será difícil encontrar uno.

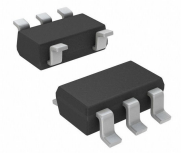
**2:** ATMEGA16U2 => Es el chip encargado de convertir la comunicación del puerto USB a serie.

**3:** Fusible rearmable de intensidad máxima 500mA => Aunque la mayoría de pc's ya ofrecen protección interna se incorpora un fusible con la intención de proteger tanto la placa Arduino como el bus USB de sobrecargas y cortocircuitos.

Si circula una intensidad mayor a 500mA por el bus USB(Intensidad máxima de funcionamiento), el fusible salta rompiendo la conexión de la alimentación.

**4:** Cristal oscilador de 16MHz necesario para el funcionamiento del reloj del microcontrolador ATMEGA16U2.

**5:** Regulador de voltaje LP2985 de 5V a 3.3V que proporciona una corriente de alimentación máxima de 150 mA.



**6:** Conector de alimentación hembra de 2,1 mm, debe suministrarse un voltaje de entre 7-12 V (límites desde 6V a 20V como máximo).

Para alimentarla si es un pequeño proyecto como los que haremos en este articulo son suficientes los 5V que nos proporciona el puerto USB, pero si deseamos alimentarla en estos u otros montajes mayores necesitaremos un cable con una conexión macho de 2,1 mm como la que se muestra en la imagen.

**7:** Condensadores de 47µF de capacidad y 16V

**8:** Diodo M7 en la entrada de alimentación de la placa. Con este diodo conseguimos establecer el sentido de circulación de la intensidad, de esta forma si se produce una contracorriente debido a la apertura de un relé u otros mecanismos eléctricos, el diodo bloqueará dicha corriente impidiendo que afecte a la fuente de alimentación.

**9:** IOREF=> Este pin esta conectado al de 5V situado a su lado y sirve para indicarle a la shield el voltaje de funcionamiento de la placa arduino y en un futuro para establecer la compatibilidad con placas AVR que funcionan también a 5V.

**10:** RESET => Esta planteado para trasladar el botón de reset a algunas shields, pero también podremos utilizarlo en algunos proyectos en el que el botón reset de la placa arduino no esté accesible. Si suministramos en este pin un valor de 0V resetearemos el microcontrolador.

**11:** Es una fuente de tensión de 3.3V (generada por el regulador de voltaje del punto 5) que como se ha dicho en el punto 5 proporciona una corriente máxima de 150mA.

OJO: Las placas que usan el chip FTDI para hacer la conversión de USB a serial (como por ejemplo la Duemilanove) tienen un regulador de tensión interno con salida a 3.3V pero proporcionan una intensidad máxima de 50mA. Debemos de tenerlo en cuenta para no superar dicha intensidad.

**12:** Este pin tiene como salida una tensión de 5V regulada por el regulador de la placa.

OJO: El suministro de tensión a través de los pines de 5V o 3.3V no pasa por el regulador, y puede dañar la placa. No es aconsejable utilizarlos como alimentación de la placa, están diseñados como salida de tensión no como entrada.

**13:** Pines de toma de tierra.

**14:** Vin => Podemos emplear este pin para alimentar la placa si utilizamos una fuente de alimentación externa, (no la podemos usar como alimentación si ya estamos alimentando a través del puerto USB o de otra fuente de tensión por el conector de 2,1 mm señalado en el punto 6, entonces en este caso podremos usar esta clavija para acceder al voltaje de alimentación), es decir, o usamos Vin exclusivamente para alimentación o si alimentamos por otro medio como toma del voltaje de alimentación.

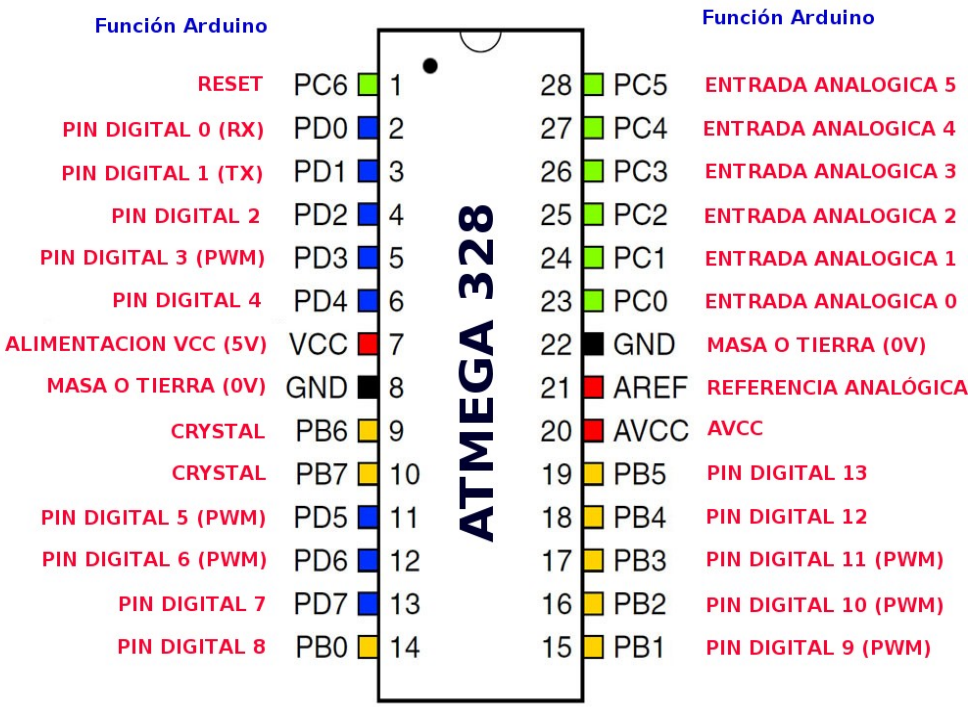
**15:** Entradas analógicas, cada una de ellas proporciona 10 bits(1024 valores).
Por defecto se mide de tierra a 5 voltios, aunque es posible cambiar la cota superior de este rango usando el pin AREF y la función analogReference().

El microcontrolador Atmega328P-PU que usa Arduino lleva integrado un conversor analógico-digital (A/D) de 6 canales. Tiene una resolución de 10 bits, retornando enteros desde 0 a 1023. Mientras que el uso principal de estos pines por los usuarios de Arduino es para la lectura de sensores analógicos, estos pines tienen también toda la funcionalidad de los pines de entrada-salida de propósito general (GPIO).

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.

**16:** ATMEGA328P-PU=> Es un microcontrolador de la marca Atmel que constituye el procesador central de la placa. Este es el elemento que programamos para manipular sus entradas y salidas.
Características:

Números de pines= 28
Voltaje de funcionamiento= 5V
Memoria Flash= 32Kbytes
EEPROM= 1Kbytes
SRAM= 2Kbytes
Velocidad del reloj= 16MHz
Corriente por pin = 40 mA




*Descripción de los pines:*
En este apartado explicaré la función de los pines del microcontrolador Atmega328. La mayoría de ellos conectan directamente con las entradas y salidas de la placa y serán explicados profundamente en su punto correspondiente.

PC6 →1 : Cuando alimentamos la placa arduino bien por usb o con alimentación externa en este pin tendremos un voltaje de aproximadamente 5V (microcontrolador funcionando) y se resetea el microcontrolador cuando tenemos un nivel de voltaje bajo durante más tiempo que la duración mínima del impulso. En el caso de la placa arduino se produce el reseteo cuando ponemos la rama de alimentación a tierra (0V) pulsando el botón de reset.

Para asegurarnos de un reseteo correcto pulsaremos el botón durante un par de segundos.

Botón en estado normal NA (normalmente abierto), el pin 1 está a un voltaje de 5V.
Botón en estado pulsado (cerrado), el pin 1 está a un voltaje de 0V. Se resetea el microcontrolador.

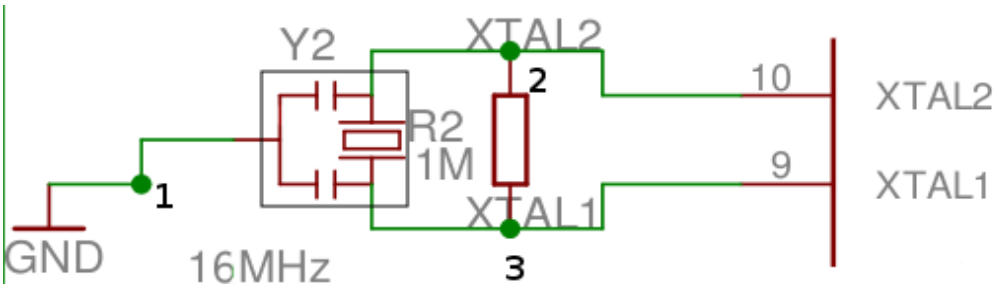
PD0 →2 y PD1 →3: Se corresponden con la entrada (RX) y la salida (LX) para la transmisión de datos TTL (Transistor Transistor Logic).

VCC → 7: Aliementación del microcontrolador con 5V de corriente continua.
AVCC→20: Es la toma de la tensión de alimentación (5V) para el conversor A/D.

GND→8 y GND→22: La abreviatura GND proviene del inglés (Ground), tierra en español o comunmente conocida en la electrónica como masa. Es un punto del circuito que se encuentra a un potencial de 0V.

PB6→9 y PB7→10: Son los pines a los que está conectado el oscilador formado por un resonador de 16Mhz en paralelo con una resistencia de 1MΩ.

El circuito está conectado de la siguiente forma:




El recuadro gris representa el  resonador cerámico de 16 Mhz.

Este circuito oscilador genera la base de tiempos para el funcionamiento del reloj del microcontrolador.

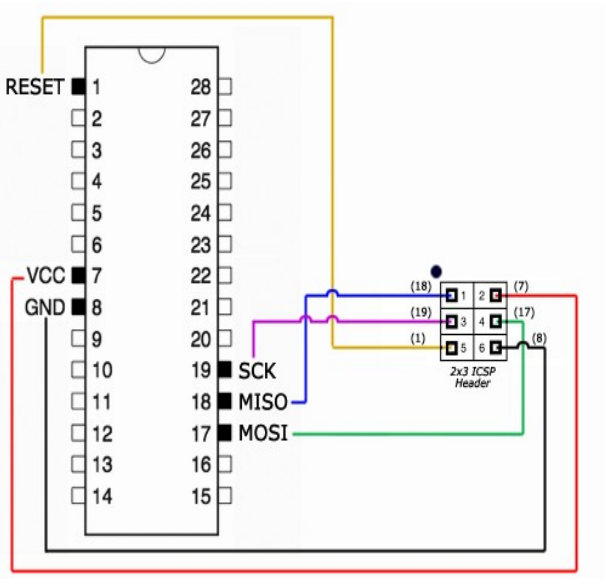
PD2→4, PD4→6, PD7→13, PB0→14, PB4→18, PB5→19: Pines digitales que pueden ser configurados como entradas o salidas.

PD3→5, PD5→11, PD6→12, PB1→15, PB2→16, PB3→17: Pines digitales que soportan modulación por ancho de pulso, (PWM) del inglés Pulse Width Modulation.

AREF→21: Es el pin de referencia analógica para el conversor A/D

PC0→23, PC1→24, PC2→25, PC3→26, PC4→27, PC5→28: Entradas analógicas.

**17:** ICSP => Estos pines sirven para la programación del ATMEGA328P-PU a través del puerto serie, de ahí las siglas ICSP (In Circuit Serial Programming), se utilizan para grabar  el bootloader en el microcontrolador o modificarlo a través de este puerto sin necesidad de sacarlo del zócalo. El bootloader ya viene grabado de fábrica en este microcontrolador.



Podemos identificar el pin1 del ISCP en la placa fijándonos el pequeño punto blanco que esta grabado sobre ella, ese punto nos indica que se trata del pin numero 1, igual ocurre en los chips, microcontroladores y otros circuitos integrados.

**18:** Led de color verde que se enciende cuando la placa esta alimentada.

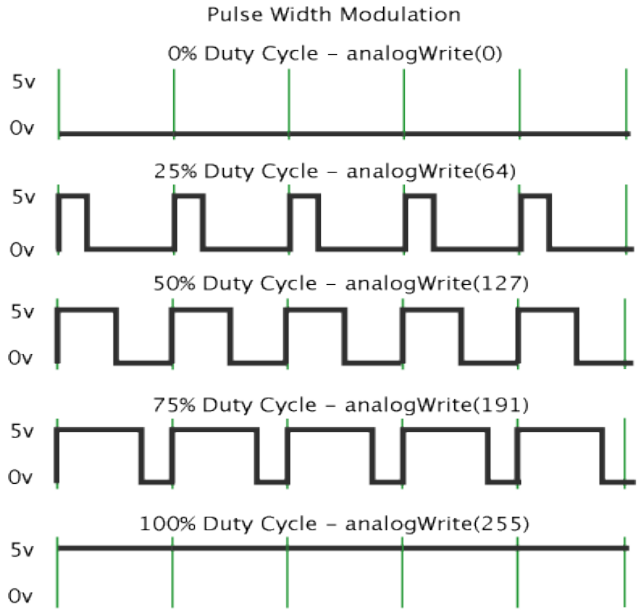
**19:** RX=> Entrada de datos TTL

**20:** TX=> Salida de datos TTL

**21:** PWM=> Pines digitales con PWM (Pulse Width Modulation) o Modulación por Ancho de Pulso. Se pueden configurar como entradas o salidas

Explicado brevemente una señal PWM es una onda digital cuadrada, donde la frecuencia es constante, pero la fracción de tiempo en que la señal está encendida (el ciclo de trabajo) puede variar entre el 0 y el 100%.

Como vemos en la imagen con analogWrite() podemos ir variando la frecuencia de la señal dando los valores apropiados que van desde 0 a 255.




**22:** Pines digitales que pueden configurarse como entradas o salidas, estos no disponen de modulación por ancho de pulso.

Una particularidad de los pines 2 y 3 es que pueden ser configurados para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor. Con attachInterrupt () podremos manejar esta característica de dichos pines.

**23:** Tierra (0v)

**24: Tensión** de referencia para las entradas analógicas. Podemos controlar dicho voltaje con analogReference().

**25:** SDA => (Serial Data) linea de datos.

**26:** SCL => (Serial Clock) linea de reloj.
Estos dos pines SDA y SCL sirven para comunicarse con dispositivos I2C / TWI usando la librería Wire.

**27:** DFU-ICSP=> Puerto ICSP para el microcontrolador ATMEGA16U2, como en el caso del ATMEGA328P-PU se emplea para comunicarnos con el microcontrolador por el serial, para reflashearlo con el bootloader, hacer algunas modificaciones, ponerlo en modo DFU, etc...

**28:** JP2=> Pines libres del ATMEGA16U2, dos entradas y dos salidas para futuras ampliaciones.

**29:** Encapsulados de resistencias.

**30:** Pulsador para resetear el microcontrolador central (ATMEGA328P-PU). Su funcionamiento ya esta explicado en el punto 16 en el que se detallan los pines del ATMEGA328.

**31:** Led conectado en paralelo entre el pin 13 y GND, cuando este pin tiene un valor HIGH(5V) el LED se enciende y cuando este tiene un valor LOW(0V) este se apaga

**32:** Led TX

**33:** Led RX

Estos Leds TX y RX se encienden cuando se transmiten datos a traves del puerto serie, es decir cuando estamos grabando el programa en el microcontrolador.

**34:** RESET-EN: Significa Reset enabled, en el habla hispana reset habilitado. Esta habilitado el auto-reset, para deshabilitarlo por cualquier tipo de seguridad (por ejemplo un proyecto que tenemos funcionando y no queremos que nadie lo reinicie) debemos desoldar los pads RESET-EN y limpiarlos de forma que estean aislados el uno del otro.

**35:** Resonador cerámico de 16 Mhz para el microcontrolador ATMEGA328P-PU. Los resonadores cerámicos son menos precisos que los cristales osciladores, pero para el caso hace perfectamente la función y ahorramos bastante espacio en la placa.
Como se detalla en el punto 16, el resonador genera da base de tiempos para el funcionamiento del microcontrolador.