Introducción

En el día anterior vimos el encendido de uno o varios led de modo cíclico. Hoy veremos cómo ese diodo led lo podemos encender a voluntad y cómo variar el nivel de encendido.

Utilizaremos como materiales:

- Placa Arduino
- Placa Shield y/o protoboard
- Diodos led
- Diodo led RGB
- Pulsador
- Resistencias de 330Ω y 10KΩ.

Entradas digitales

Cualquiera de las patillas digitales se Arduino pueden funcionar como entrada o salida. Para configurar una patilla como entrada, basta con poner como vimos *const int* en el IDE. En nuestro caso, en BitBloq bastará con buscar el sensor de entrada que vayamos a conectar, lo arrastremos y lo conectemos a la patilla que hayamos seleccionado como entrada.

Como entrada digital podemos utilizar cualquier sensor que origine una señal de salida uno/cero cuando se active. La entrada digital más común es un simple pulsador. En nuestro caso se tomará un pulsador y lo conectaremos según el esquema de la figura. En este caso, el pulsador estará continuamente dando una tensión de 0V y al pulsar pondría la entrada a nivel +5V conectando directamente a la salida de señal. La resistencia de 10kW impide que al pulsar se cortocircuiten +5V y GND.

El segundo de los esquemas genera una salida continua a nvel alto (+5V), y al pulsar hace cambiar la salida del nivel alto a bajo. Sistema utilizado en industria (Lógica positiva), para sensores de seguridad, dado que si se rompiera un conductor funcionando como en el primer caso, el sistema lo ignoraría y podríamos tener accidentes. En el caso de lógica positiva, si por algún roedor o deterioro se rompiera el conductor, la señal de entrada cambiaría a 0V, lo que el Arduino lo interpretaría como que ha variado, mostrándonos así que hay un fallo. La resistencia en este caso sigue teniendo la misma función que en el primer esquema.

Ejercicio 1.- Encendido de un led al pulsar un botón

Para este ejercicio usaremos la función IF-ELSE junto a las sentencias DIGITAL READ y DIGITAL WRITE

La función IF-ELSE tendría la siguiente estructura y significado:

IF (Condición) → → { Instrucción; }	Si se cumple la condición → → Ejecuta la instrucción
ELSE { Instrucción alternativa }	En caso contrario ejecuta esta otra instrucción

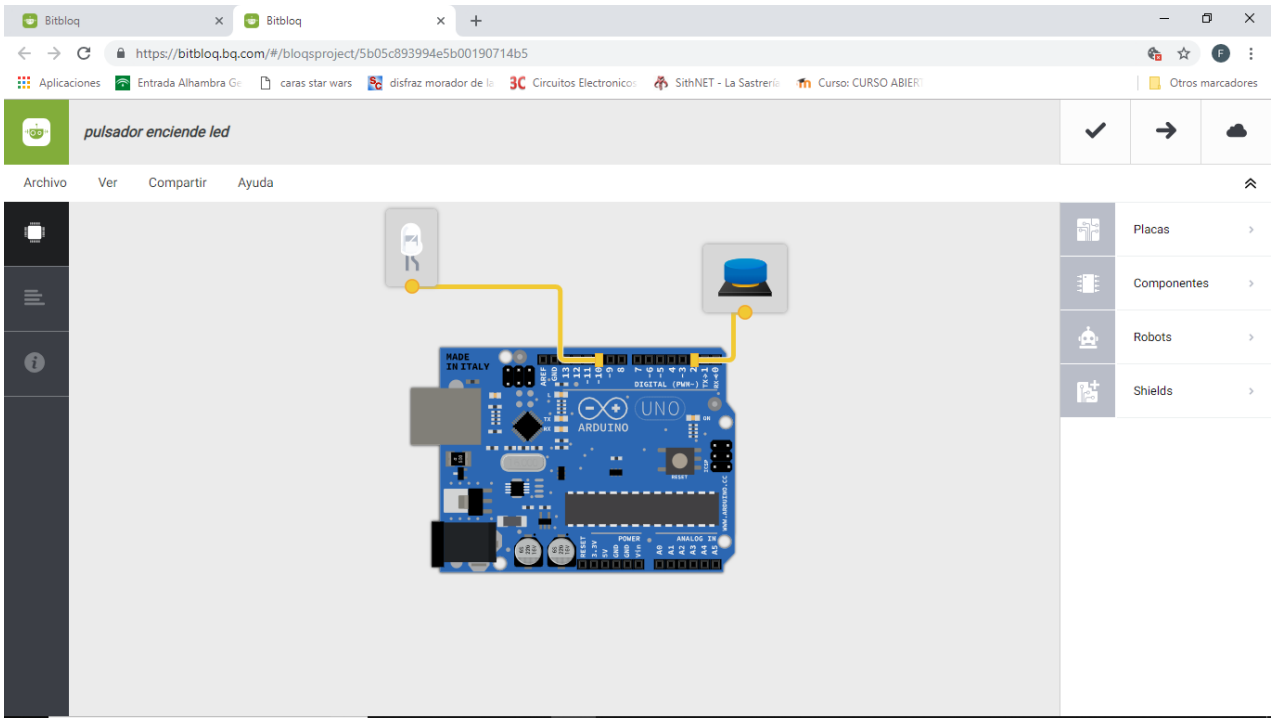
Las instrucciones que utilizaremos en este caso serán las sentencias DIGITAL READ y DIGITAL WRITE

La estructura que tienen son:

digitalWrite (Led, LOW);	Escribe en modo digital en la patilla llamada LED un nivel Bajo.
digitalRead (int) == HIGH);	Lee el pin digital "int" y comprueba si es igual a == nivel alto;

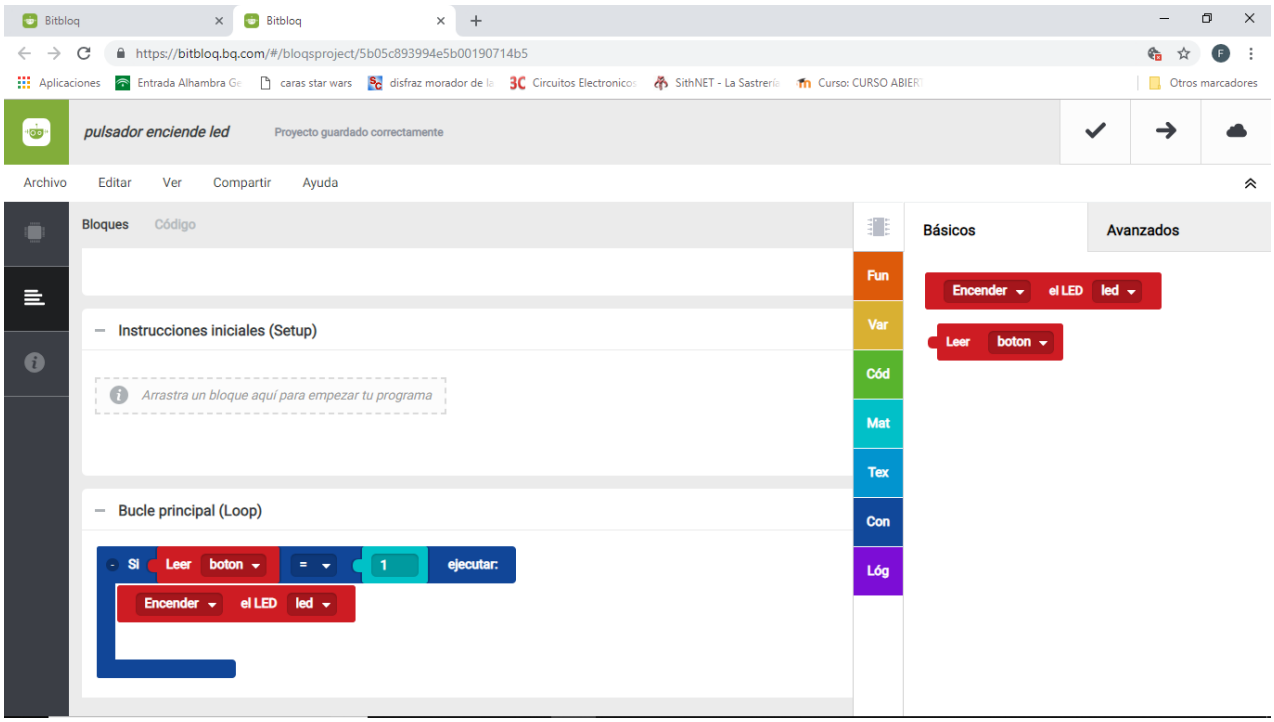
Programamos con BitBloq.Bq

1.- En la pantalla de hardware seleccionaremos la placa, el pulsador y un led.



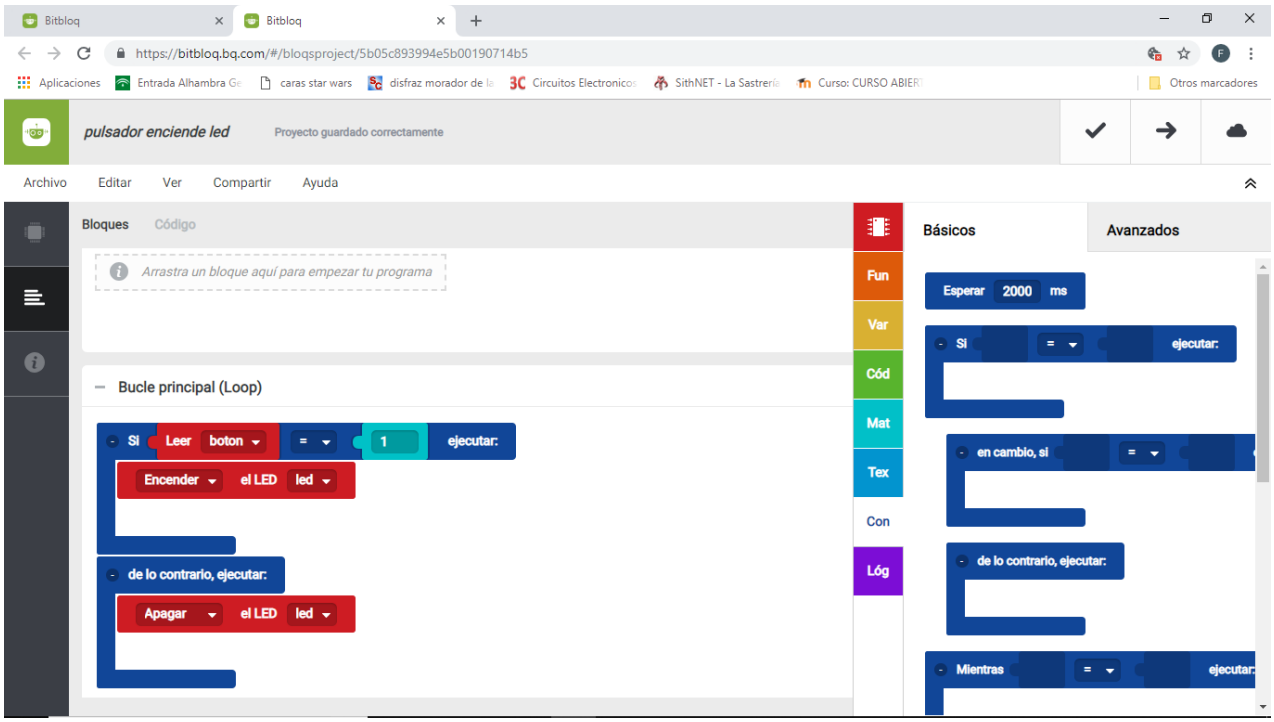
2.- Nos pasamos a la pantalla de software y nos saltamos al *void loop( ).*

3.- En la pestaña CON (control) cogemos la función En el cuadro de al lado de si pondremos la condición (Leer el pulsador) que encontraremos en la pestaña Componentes y la condición en la pestaña MAT ver si está a nivel alto).



En el hueco inferior ponemos la orden que queremos que haga. Quedando la instrucción como en la imagen

4.- Igualmente ocurre con la función *ELSE*. En la pestaña CON picamos en la instrucción *ELSE* (De lo contrario) la arrastramos y ponemos la instrucción correspondiente.



Si hiciéramos la programación con el IDE de Arduino, tendríamos que usar el siguiente código:

```
/***  Included libraries  ***/

/***  Global variables and function definition  ***/
const int boton = 2;      // Defino las entradas y salidas, pulsador, patilla 2 y led salida 10
const int led = 10;

/***  Setup  ***/
void setup() {
      pinMode(boton, INPUT);     // Defino que el pulsador será una entrada y el led una salida
      pinMode(led, OUTPUT);

}

/***  Loop  ***/
void loop() {
      if (digitalRead(boton) == 1) {
            digitalWrite(led, HIGH);
      } else {
            digitalWrite(led, LOW);
      }
}
```

EJERCICIO 2.- Pulsador con memoria

En este ejercicio vamos a modificar el programa anterior para que al pulsar 1 vez se encienda el led y se mantenga encendido hasta que se accione nuevamente el pulsador. Para poder hacer este ejercicio necesitamos conocer el concepto de variable.

VARIABLE es una forma de nombrar y almacenar un valor para su uso posterior por parte del programa, como los datos de un sensor o un valor intermedio utilizado en un cálculo.

Declarando variables

Antes de que se utilicen, todas las variables deben ser declaradas. Declarar una variable significa definir su tipo y, opcionalmente, establecer un valor inicial (inicializar la variable). Las variables no tienen por qué inicializarse (se les asigna un valor) cuando se declaran, pero se les suele dar un valor aunque no se use.

Los datos que guardamos en las variables pueden ser de diferentes tipos, vamos a listar algunos de ellos. Para una referencia completa de los tipos de variables en Arduino se puede consultar esta página web.

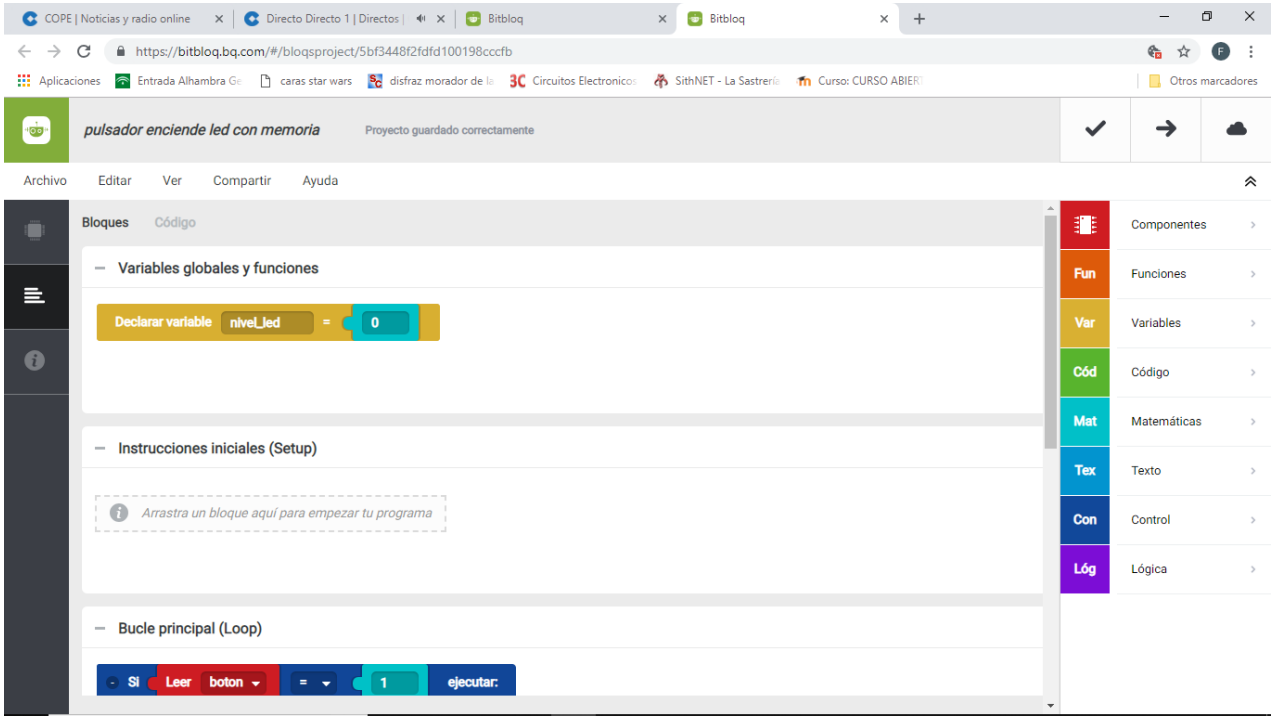
- **char**, se utilizan para almacenar caracteres, ocupan un byte.
- **byte**, pueden almacenar un número entre 0 y 255.
- **int**, ocupan 2 bytes (16 bits), y por lo tanto almacenan número entre 2^{-15} y 2^{15}-1, es decir, entre -32,768 y 32,767.
- **unsigned int**, ocupa también 2 bytes, pero al no tener signo puede tomar valores entre 0 y 2^{16}-1, es decir entre 0 y 65,535.
- **long**, ocupa 32 bits (4 bytes), desde -2,147,483,648 a 2,147,483,647.
- **unsigned long**.
- **float**, son números decimales que ocupan 32 bits (4 bytes). Pueden tomar valores entre -3.4028235E+38 y +3.4028235E+38.
- **double**, también almacena números decimales, pero disponen de 8-bytes (64 bit).

Siempre que elegimos un tipo de dato debemos escoger el que menos tamaño necesite y que cubra nuestras necesidades, ya que ocuparán espacio en la memoria de nuestra placa y podría darse el caso de que nuestro programa requiera más memoria de la disponible.

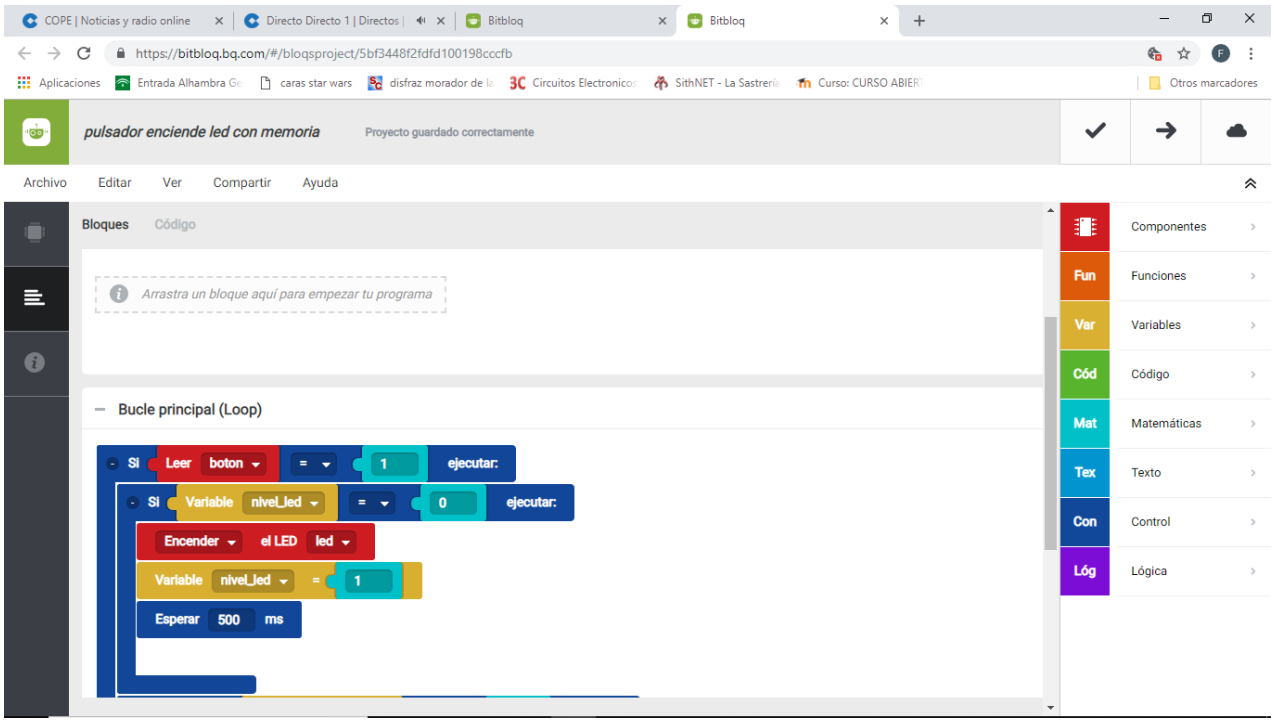
Programando con BitBloq.Bq

1.- Si partimos del programa de la práctica anterior no necesitamos ir a la parte de Hardware a modificar nada.

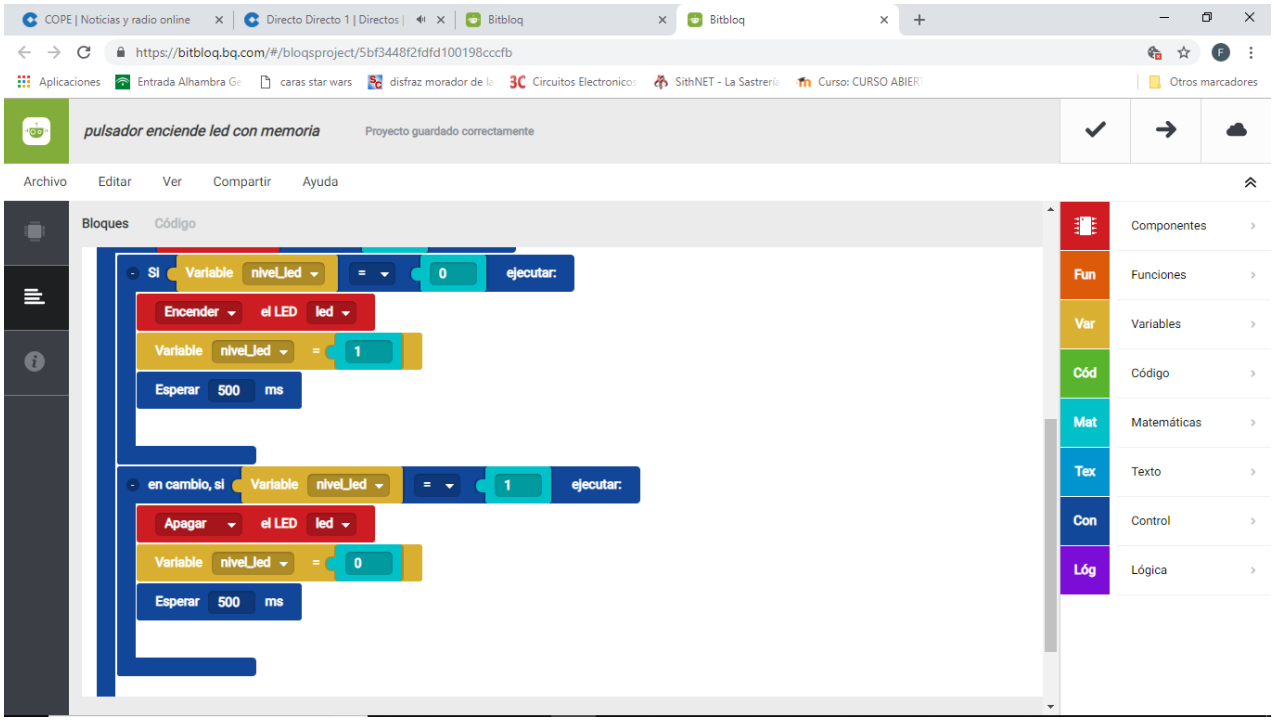
2.- En la pestaña amarilla VAR picamos en declarar variable y la arrastramos a la declaración de variables globales; ponemos un nombre a la variable a voluntad y en la pestaña MAT arrastramos un valor inicial 0.



3.- Nos bajamos al bucle del *Loop* que habíamos escrito antes. Como se ve en la figura anterior, hemos puesto “SI Leer botón = 1 ejecutar…..” Llegados a este punto tenemos dos acciones posibles, encender el led si está apagado o apagarlo si está encendido. Pues hay que escribir eso mismo en lo que se llaman instrucciones concatenadas, pero tendremos que modificar el valor de la variable cuando acabemos el bucle y poner un retardo al terminar cada uno de los bucles para que nos dé tiempo a dejar de pulsar.



En la imagen se ve el primero de los condicionantes, si el led está apagado: Enciende el led, pone la variable a 1 para indicar al programa que está encendido el led y espera. Así tenemos tiempo de soltar el pulsador.



En esta última figura se pueden ver ya los dos bucles, el primero que ya conocemos y el segundo que indica al programa lo contrario del primero, si el led (Variable Enciende_Led) está encendido, apaga el led, cambia la variable a cero y espera.

Para hacer este ejercicio programando mediante el IDE de Arduino tendríamos un código muy parecido al siguiente:

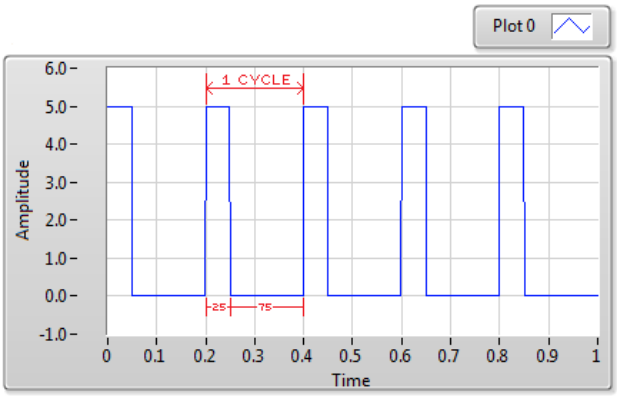
```
/***  Included libraries  ***/

/***  Global variables and function definition  ***/
const  int  led  =  10;                          // defino los pines de entrada y salida y la patilla que uso
const  int  boton  =  2;

float  nivel_led  =  0;                          // Defino la variable que indicará al programa el estado del led

/***  Setup  ***/
void  setup()  {                                 // Indico cada pin si es entrada o salida
      pinMode(led,  OUTPUT);
      pinMode(boton,  INPUT);

}

/***  Loop  ***/
void  loop()  {
      if  (digitalRead(boton)  ==  1)  {         // Le digo que lea el estado del pulsador
            if  (nivel_led  ==  0)  {
                  digitalWrite(led,  HIGH);      // Indico que si está apagado lo encienda
                  nivel_led  =  1;
                  delay(500);
            }  else  if  (nivel_led  ==  1)  {   //Indico que si está encendido lo apague
                  digitalWrite(led,  LOW);
                  nivel_led  =  0;
                  delay(500);
            }
      }
}
```

EJERCICIO 3.- Encendido progresivo de un led

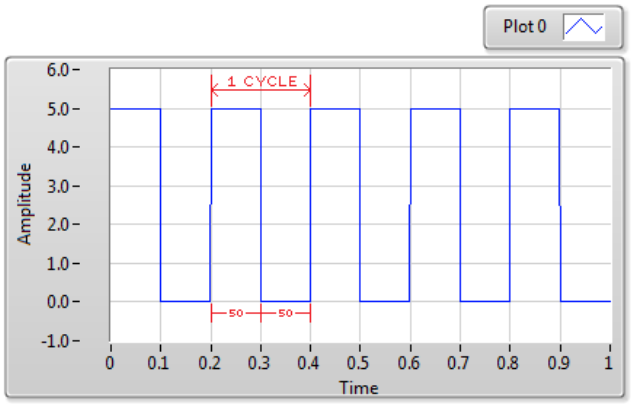
Conocemos ya las salidas digitales, con ellas podemos encender/apagar leds, relés, motores a velocidad constante, etc… ¿Pero si queremos variar la tensión de una salida entre 0 y 5V? No es posible de hacer sin un conversor digital analógico (DAC) externo.

Sin embargo Arduino permite simular esas variaciones analógicas usando lo que se conoce como modulación por ancho de pulsos PWM. Esta función es posible usando la instrucción analogWrite ( ) para generar señales PWM, esta función se aplica a ciertos pines que están marcados en la placa con los símbolos ~, esto es, los pines 3 – 5 – 6 – 9 – 10 y 11.

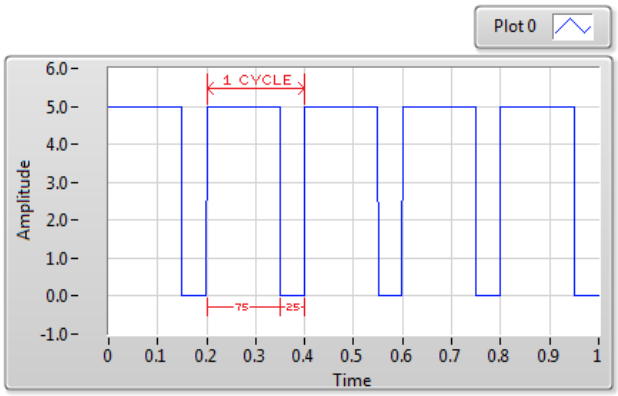
Emular el control analógico lo hace fraccionando y troceando la salida digital. Una señal digital puede ser cero (0V) o uno (+5V). Sin embargola señal PWM o modulada en ancho de pulsos trocea cada uno de los ciclos de trabajo de la onda cuadrada y lo modifica, recortando parte de dicho ciclo. De este modo consigue que el valor uno esté solamente una porción del ciclo, que variará del 0 al 100% del ancho de la señal. Así esta modulación del ancho e pulso se puede apreciar en las figuras:



Ciclo de trabajo del 25%



Ciclo de trabajo del 50%



Ciclo de trabajo del 75%

Claro, todo esto necesita de una instrucción que modifique esa salida. La instrucción es la de:

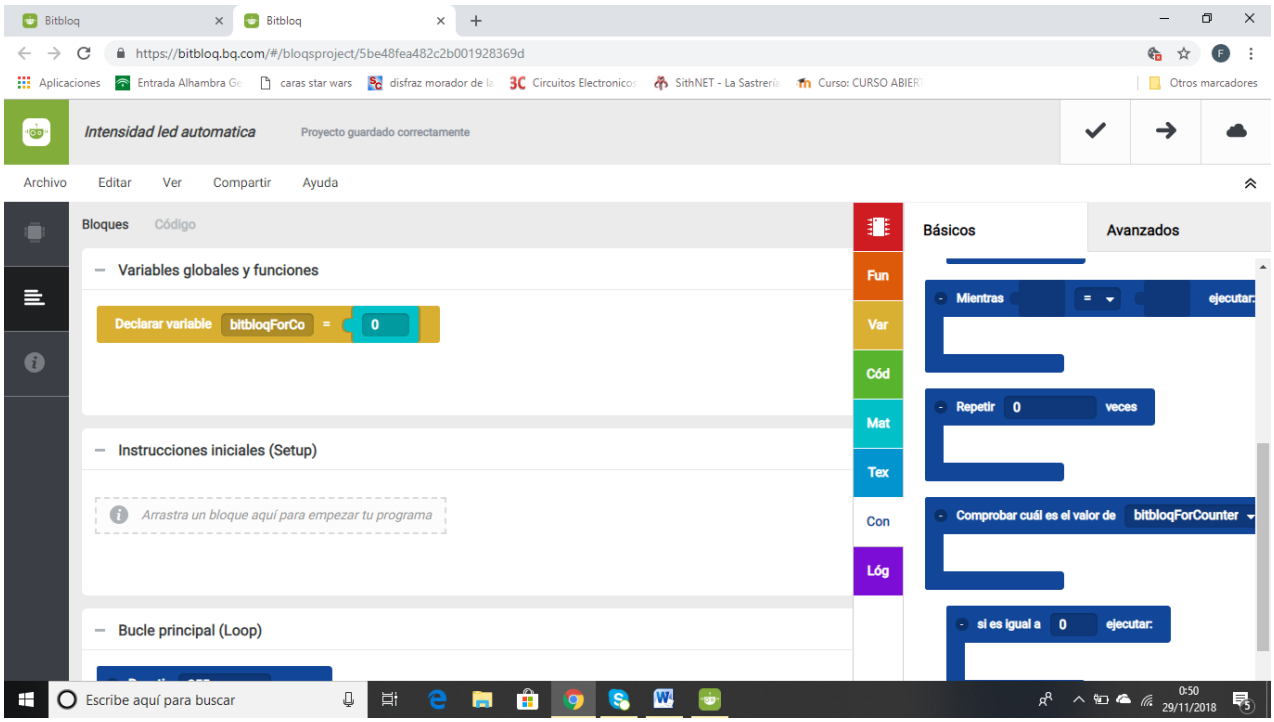
analogWrite (salida,valor); La orden analogWrite tiene dosparámetros:

- El parámetro salida corresponde al pin seleccionado para salida PWM.
- El valor (o variable) es el porcentaje que se va a encender a la salida. En todo caso variará entre 0 y 255 (2^8 Bit).

Llegados a este punto podemos realizar un pequeño programa que haga incrementar el brillo de un led progresivamente.

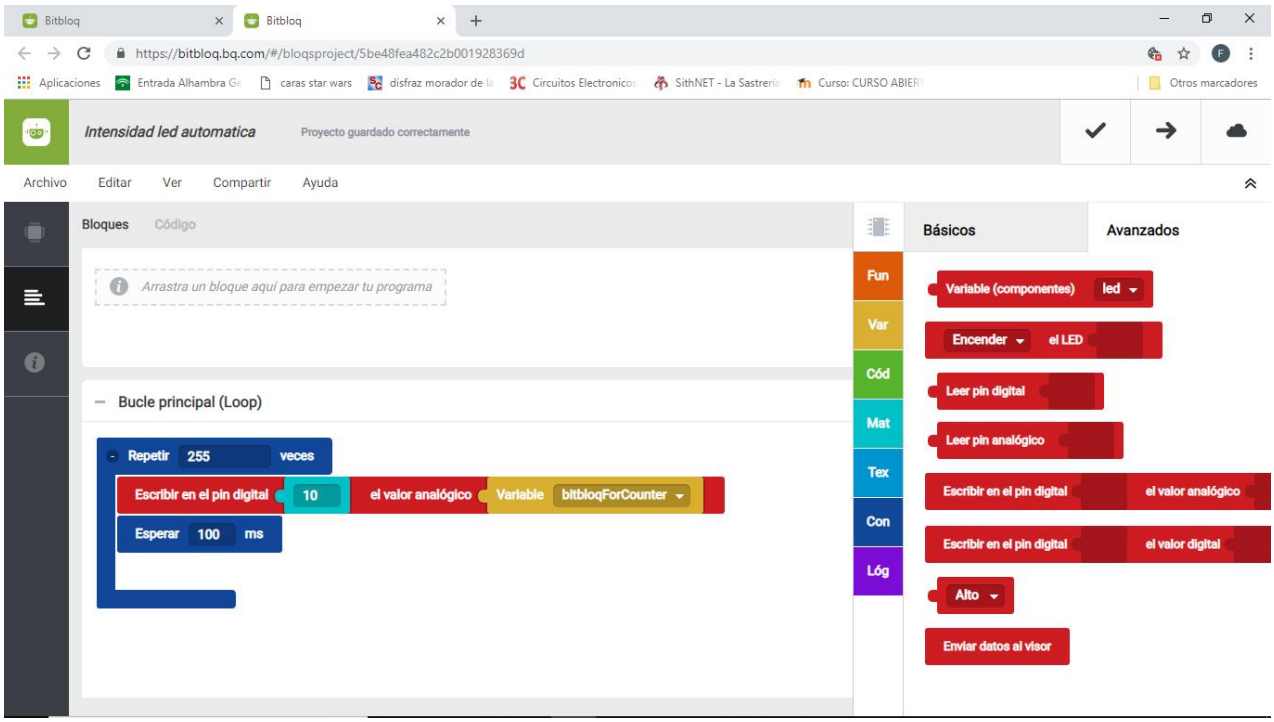
1.- Seleccionamos la placa y ponemos el led conectado en este caso al pin 10.

2.- Vamos a definir una variable que llamaremos de un modo especial “bitbloqForCounter”, y la igualaremos a cero.



3.- En la pestaña de control CON tomamos la función Repetir, que fijaremos en 255 veces.

4.- En la pestaña COMPONENTES, en la carpeta *avanzados* cogeremos la función escribir en el pin digital …… el valor analógico.



4.- Para que sea lento le añadiremos un retardo, en este caso 100mS (0’1 Seg).

Al transferir el programa a la placa el diodo irá aumentando progresivamente luminosidad hasta que llegue al valor máximo, después se apagará e iniciará nuevamente el ciclo.

Si programásemos con el IDE de Arduino, el código sería el siguiente:

```
/***  Included libraries  ***/

/***  Global variables and function definition  ***/
const int led = 10;

float cuenta = 0;     // Creo la variable cuenta que servirá para variar su valor entre 0 y 255 que será el nivel de
                      // modulación de pulsos.

/***  Setup  ***/
void setup() {
      pinMode(led, OUTPUT);

}

/***  Loop  ***/
void loop() {
      for (int cuenta = 0; cuenta < 255; cuenta += 1) {
            analogWrite(10, cuenta);
            delay(100);
      }
}
```