Sesión 5-

Las entradas digitales de Arduino

5.1 Objetivos

- Conocer las entradas digitales.
- Leer el primer pulsador.
- Presentar los valores booleanos.
- Un operador: Negación.

5.2 Material requerido.

Arduino Uno o similar. Un PC con el entorno de Arduino correctamente instalado y configurado.

Una Protoboard.

Un diodo LED.

Un pulsador.

Dos resistencias de 330 Ohmios.

Algunos cables de Protoboard.

5.3 Entradas digitales

Con frecuencia en electrónica necesitamos saber si una luz está encendida o apagada, si alguien ha pulsado un botón o si una puerta ha quedado abierta o está cerrada.

A este tipo de señales todo / nada, SI / NO, TRUE /FALSE, 0/1 se les llama digitales, y podemos manejarlas con los pines de 0 al 13 de Arduino y por eso hablamos de pines digitales.

Muchos de los sensores y actuadores que vemos en el mundo real son digitales:

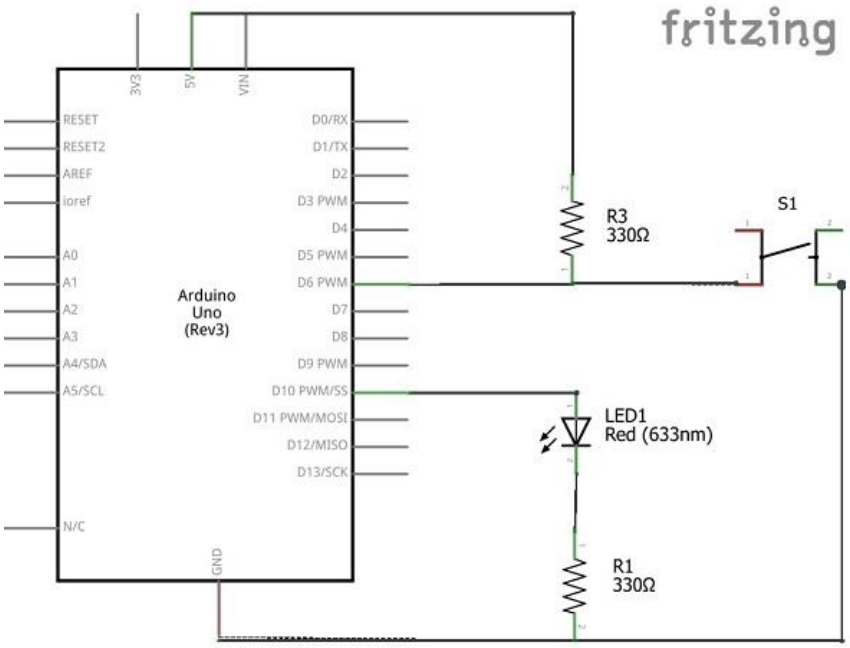
- *Como actuadores digitales, tenemos luces, alarmas, sirenas, desbloqueo de puertas, etc.*
- *Como sensores digitales podemos mencionar botones y pulsadores, Finales de carrera, desbordamiento de nivel, sensores de llamas, humo o gases tóxicos.*

Hemos visto que Arduino pueden usar los **pines digitales** como salidas todo o nada para encender un LED. De la misma manera podemos leer valores, todo o nada, del mundo exterior.

En esta sesión veremos que los **pines digitales** de Arduino pueden ser usados tanto de entrada como de salida. Vamos a leer un botón o pulsador externo y vamos a encender o apagar un LED en función de que el botón se pulse o no.

5.4 Esquema electrónico del circuito.

Montaremos un circuito con un diodo LED y resistencia conectado al pin digital 10 de Arduino, tal y como vimos en las sesiones previas y ademas un segundo circuuito con un pulsador S1 conectado al pin 6 con una resitencia como se muestra en el diagrama siguiente.

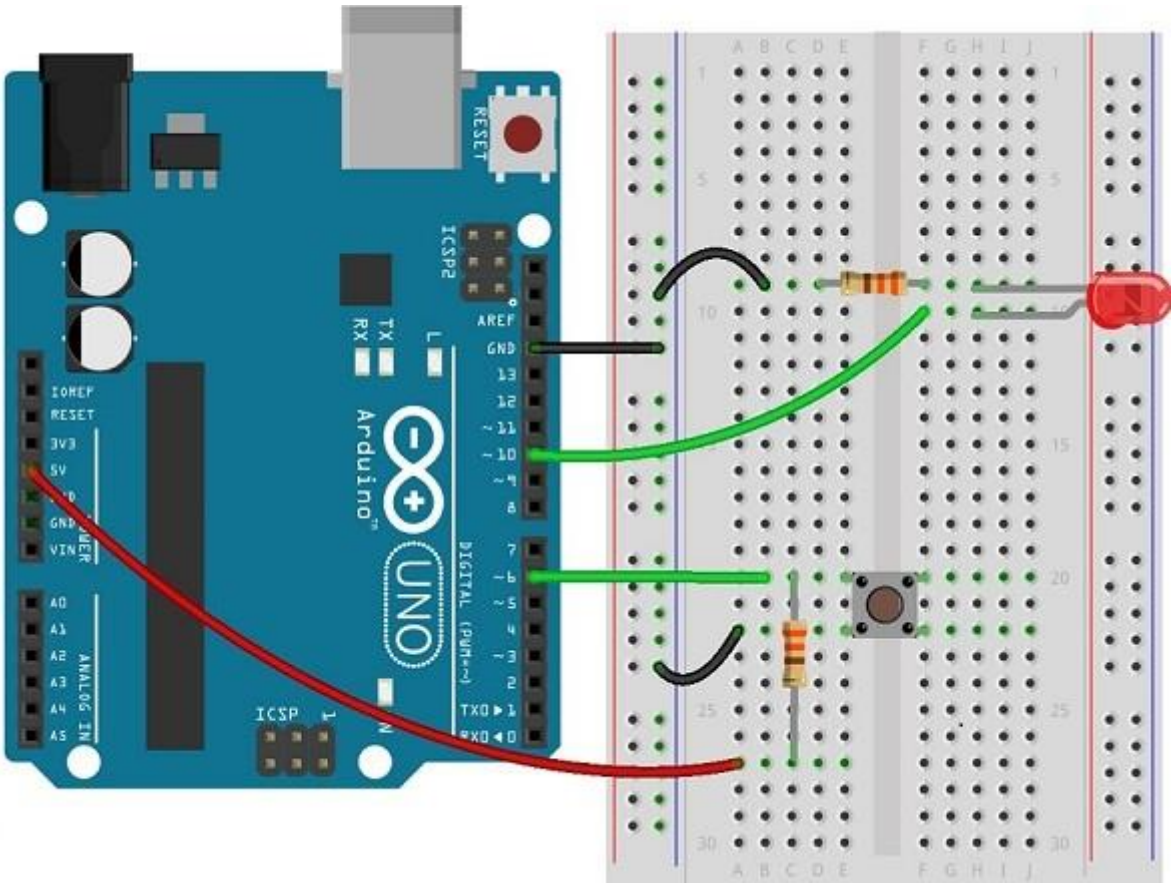


Obsérvese que mientras no pulsemos S1 el pin 6 de Arduino está conectado a 5V a través de la resistencia R3 forzando una lectura de tensión alta (HIGH). En cambio cuando pulsemos S1 cerraremos el circuito del pin 6 a Ground con lo que leerá tensión baja, LOW. En ambos casos tenemos un valor de tensión definido.

Si no pusiéramos la resistencia R3, al pulsar S1 leeríamos correctamente LOW en el pin 6. Pero al dejar de pulsar S1 el pin 6 estaría en un estado flotante, que es ni HIGH ni LOW sino indeterminado. Como esto es inaceptable en circuitos digitales forzamos una lectura alta con R3.

- *A esta resistencia que fuerza el valor alto en vacio se le conoce como **pullup** Si la conectáramos a masa para forzar una lectura a Ground se le llamaría **pulldown** resistor.*
- *Esta resistencia es clave para que las lecturas del pulsador sean consistentes. El circuito, simplemente, no funcionará bien si se omite (volveremos sobre esto).*

Y aquí tenemos el esquema para protoboard del circuito.



- *En este esquema hemos seguido la práctica habitual de usar cables negros para conectar a masa y cables rojos para conectar a tensión (5V).*
- *Obsérvese que el pulsador S1 tiene cuatro pines (el que está sobre la resistencia horizontal). Esto es porque cada entrada del interruptor tiene dos pines conectados. En nuestro circuito simplemente ignoramos los pines secundarios.*

5.5 Leyendo los pulsadores

Empecemos haciendo un programa que haga que el LED se encienda cuando pulsamos el botón y se apague cuando lo soltamos. Para ello pediremos a Arduino que configure el pin digital 10 (D10) como salida para manejar el LED, y el pin digital 6 (D6) como entrada para leer el botón.

Normalmente en programas sencillos basta con poner el número de pin en las instrucciones. Pero a medida que el programa se complica esto tiende a provocar errores difíciles de detectar.

Por eso es costumbre definir **variables** con los números de pin que usamos, de forma que podamos modificarlos tocando en un solo lugar (y no teniendo que buscar a lo largo del programa). Vamos a escribir esto un poco más elegantemente:

```
int LED = 10 ;
int boton = 6;

void setup()
   {
      pinMode( LED, OUTPUT) ; // LED como salida
      pinMode( boton  , INPUT) ;          //botón  como entrada
   }
```

- *Atención: C++ diferencia entre mayúsculas y minúsculas y por tanto LED, Led y led no son lo mismo en absoluto. Del mismo modo, pinMode es correcto y en cambio pinmode generará un error de compilador fulminante.*

- *He usado la variable boton sin acento porque no es recomendable usarlos ni la ñ en los nombres de variables, porque pueden pasar cosas extrañas.*

Vimos que para encender el LED bastaba usar digitalWrite(LED, HIGH). Para leer un botón se puede hacer algo similar: digitalRead(botón). Veamos cómo podría ser nuestro loop:

```
void loop()
   {
      int valor = digitalRead(boton) ;        // leemos el valor de boton en valor
      digitalWrite( LED, valor) ;
   }
```

¿Fácil no? Aunque el LED está encendido hasta que pulsamos el botón y se apaga al pulsar.

¿Cómo podríamos hacer lo contrario, que el LED se encienda al pulsar y se apague si no? Bastaría con escribir en LED lo contrario de lo que leamos en el botón.

Existe un operador que hace eso exactamente el **operador negación** " ! " . Si una valor dado x es HIGH, entonces !x es LOW y viceversa.

- *Un operador es un símbolo que relaciona varios valores entre sí, o que modifica el valor de una variable de un modo previsible.*

- *Ejemplos de operadores en C++ son los matemáticos como +,-,* , / ; y hay otros como la negación ! o el cambio de signo de una variable : – x. Iremos viendo más.*

De hecho este tipo de operaciones son tan frecuentes que C++ incorpora un tipo llamado **bool** o booleano que solo acepta dos valores TRUE (cierto) y FALSE y son completamente equivalentes al 1 / 0, y al HIGH / LOW

Este nuevo programa sería algo así:

```
void loop()
   {
      int valor = digitalRead(boton) ;      // leemos el valor de boton en valor
      digitalWrite( LED, !valor) ;         //Escribimos valor en LED
    }
```

Hemos definido valor como bool, porque podemos usar el valor de tensión alto como TRUE y el valor bajo como FALSE.

SI el botón no está pulsado el D6 leerá TRUE y por tanto pondrá LED a FALSE. En caso contrario encenderá el LED.

De hecho podríamos escribir una variante curiosa del blinking LED usando el operador negación:

```
void loop()
   {
      bool valor = digitalRead (LED) ;
      digitalWrite(  LED, !valor) ;
      delay ( 1000) ;
   }
```

- *Podemos leer la situación actual de un pin (nos devuelve su estado actual), aún cuando lo hayamos definido como salida, En cambio no podemos escribir en un pin definido como entrada.*

La primera linea lee la situación del LED y la invierte en la segunda línea, después escribe esto en LED. Y puestos a batir algún record, podemos escribir el blinking led en solo dos líneas:

```
void loop()
   {
      digitalWrite(  LED , ! digitalRead( LED)) ;
      delay ( 1000) ;
   }
```

- *Las instrucciones dentro de los paréntesis se ejecutan antes que las que están fuera de ellos. Por eso el digitalRead se ejecuta antes que el digitaWrite..*

5.6 Resumen de la sesión

- Hemos visto una forma de leer señales digitales del mundo exterior además de poder enviarlas:
  - digitalRead(pin)
  - digitalWrite(pin , valor)
- Hemos conocido un nuevo componente: el pulsador.
- Conocemos un nuevo tipo en C++, el booleano y un nuevo operador de negación.