E. E. S. T. Nº 5 – TIGRE

ELECTRÓNICA

SISTEMAS DE CONTROL

Arduino

TRABAJO PRÁCTICO Nº 5

TEMA: Entradas y salidas digitales.

Funciones de programador. Operadores comparativos.

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

Funciones de programador

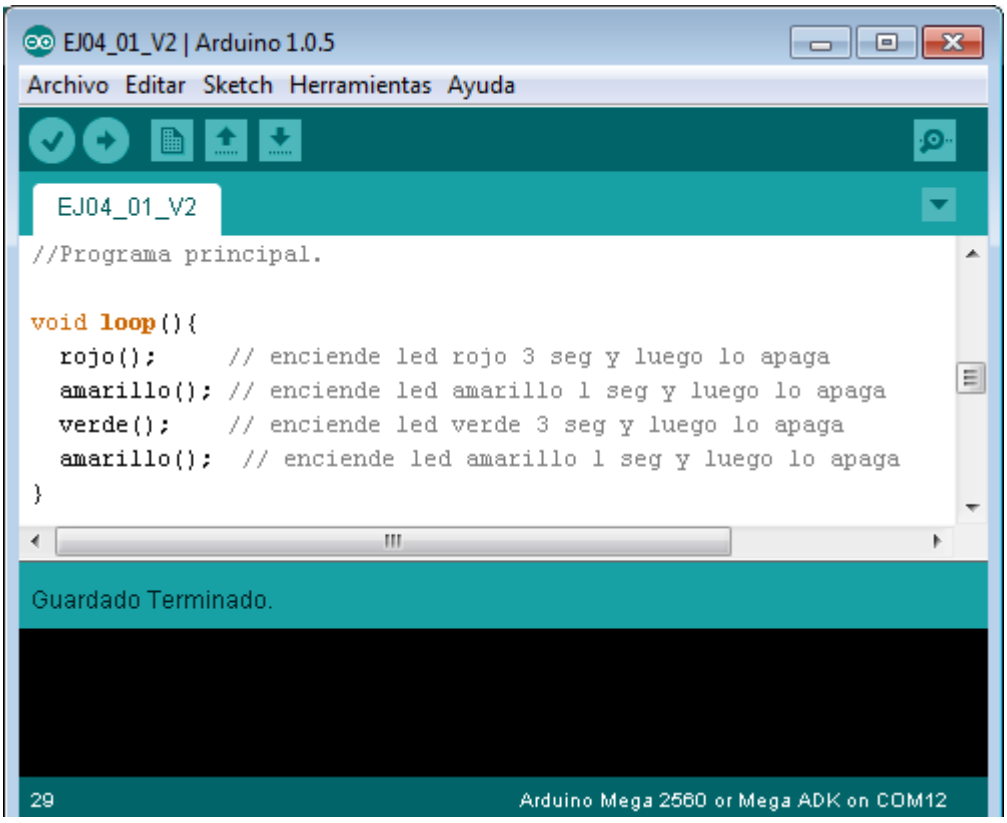
Una **función** es un conjunto de acciones de un lenguaje de programación que resuelven un determinado **problema**.

En principio podemos decir que hay dos tipos de funciones: las funciones **propias** del **lenguaje** y las funciones de **programador**.

Las funciones del lenguaje son aquellas que provee el lenguaje con el cual se está programando. Ud ya ha utilizado una de estas funciones: **`digitalWrite`**.

Usar las funciones del lenguaje nos ahorra mucho tiempo porque muchas situaciones ya están **resueltas**. Los únicos requisitos para poder utilizar las funciones del lenguaje es saber de su existencia y conocer qué **datos** hay que proveerles para que puedan realizar su tarea. Recordemos que en el caso de **`digitalWrite`** debíamos proveerle del **pin** y el **valor** deseado: por ejemplo **`digitalWrite(8, HIGH);`** coloca el **pin 8** del Arduino en estado **alto**.

Antes de avanzar con el conocimiento de nuevas funciones que nos provee el lenguaje de Arduino, me gustaría detenerme en uno de los ejercicios del trabajo práctico anterior, por ejemplo en el primero: el que tenía que hacer una secuencia de luces de un semáforo de tránsito. No sé como lo resolvió pero seguro no lo hizo como le muestro a continuación:



¿Fácil, no? Más sencillo imposible. ¿No queda más claro así el programa?

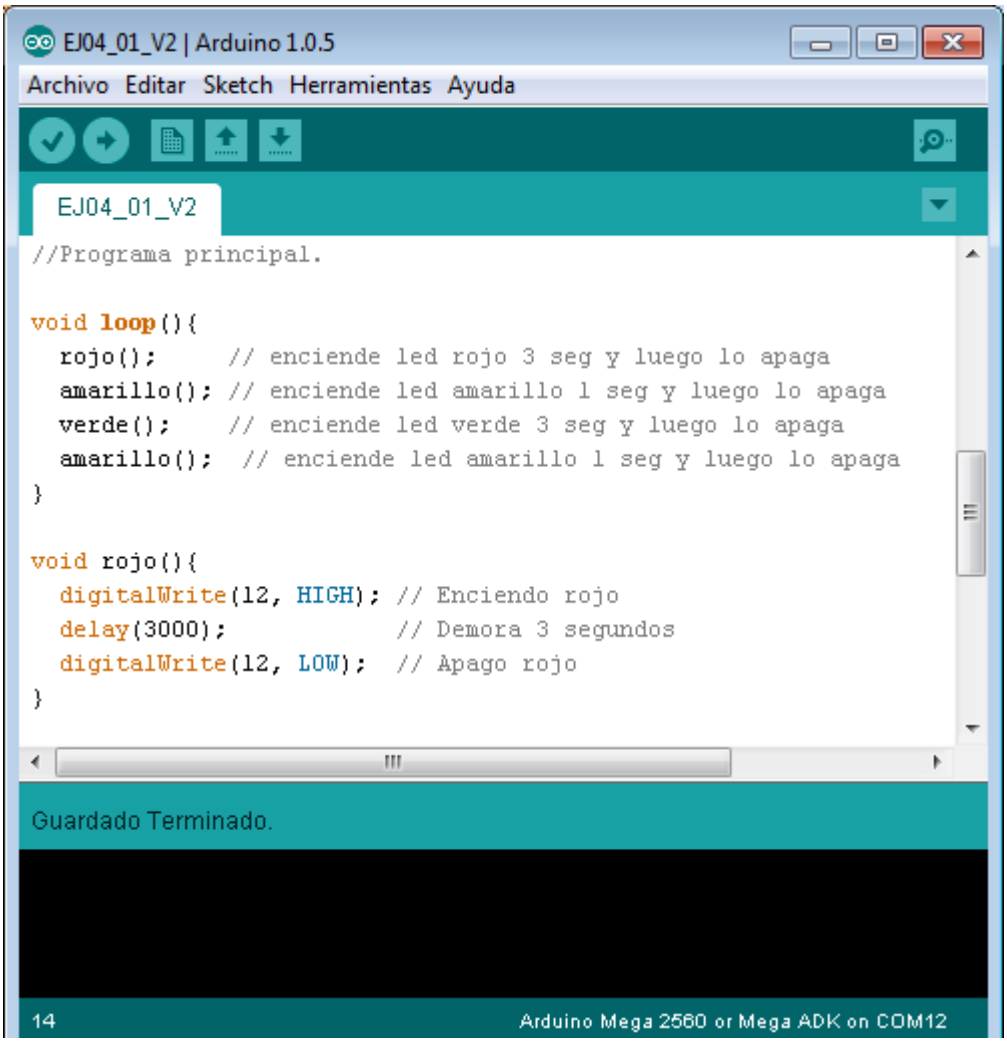
Ud me dirá: "***Pero yo no sabía que podía colocar rojo(); para encender el LED rojo***."

Yo le contesto que **`rojo();`** es una función que no viene con el lenguaje Arduino sino una función que construí para resolver este problema: es una **función de programador**.

Le voy a mostrar como lo hice para que pueda **adaptarlo** a su programa.

Seguramente UD. configuró el LED rojo en un determinado pin como salida: en mi caso usé el pin 12.

Ahora le muestro como construí la función **`rojo()`**.



Observe que una función de programador debe estar **definida** en algún lado, en este caso lo hice **debajo** del bloque **`void loop()`**.

Yo la llamé **rojo**, porque el programador se reserva el derecho de bautizar a su criatura con el nombre que desee.

Es muy importante la palabra **`void`** delante del nombre, ya que indica que esta función devuelve como resultado **ningún valor**, es decir, simplemente hace algo pero no devuelve un resultado.

Otro elemento importante son los **paréntesis**: allí irían los **parámetros** de la función, que son los datos necesarios para llevar a cabo una tarea. Pero como en este caso me pareció innecesario, lo dejé vacío.

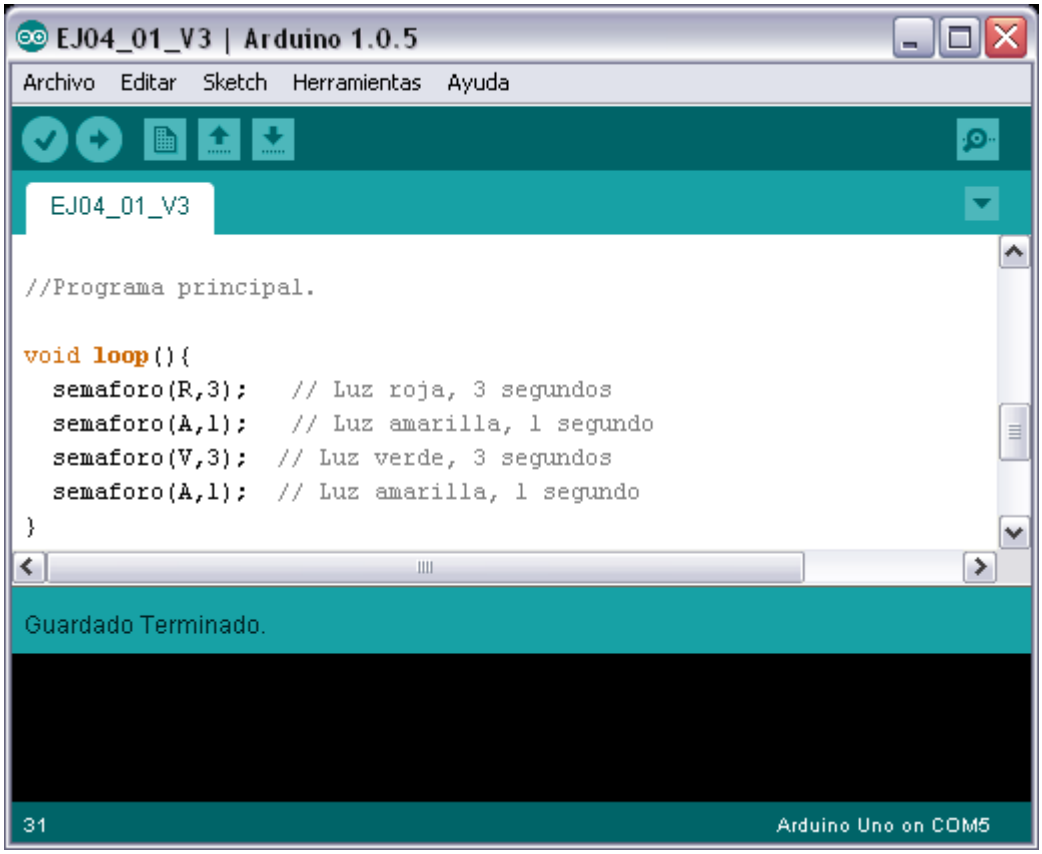
Dentro del cuerpo de la función he colocado las acciones que se van llevar a cabo cuando invoque a **`rojo();`** en el programa principal.

Actividad

EJ-0501) Realice la versión 2 de su sketch del semáforo (EJ-0401) para que los encendidos de cada una de las luces se hagan mediante **tres funciones** diferentes: **`rojo()`**, **`amarillo()`** y **`verde()`**. Utilice como guía para llevar a cabo la tarea las imágenes y explicaciones de las páginas anteriores.

Una versión mejorada

Si hilamos un poco más fino veremos que las diferencias entre las funciones **`rojo()`**, **`amarillo()`** y **`verde()`** son muy pocas. Observe este modelo de sketch.



En vez de usar tres funciones diferentes usamos **una** sola función: **`semaforo`**.

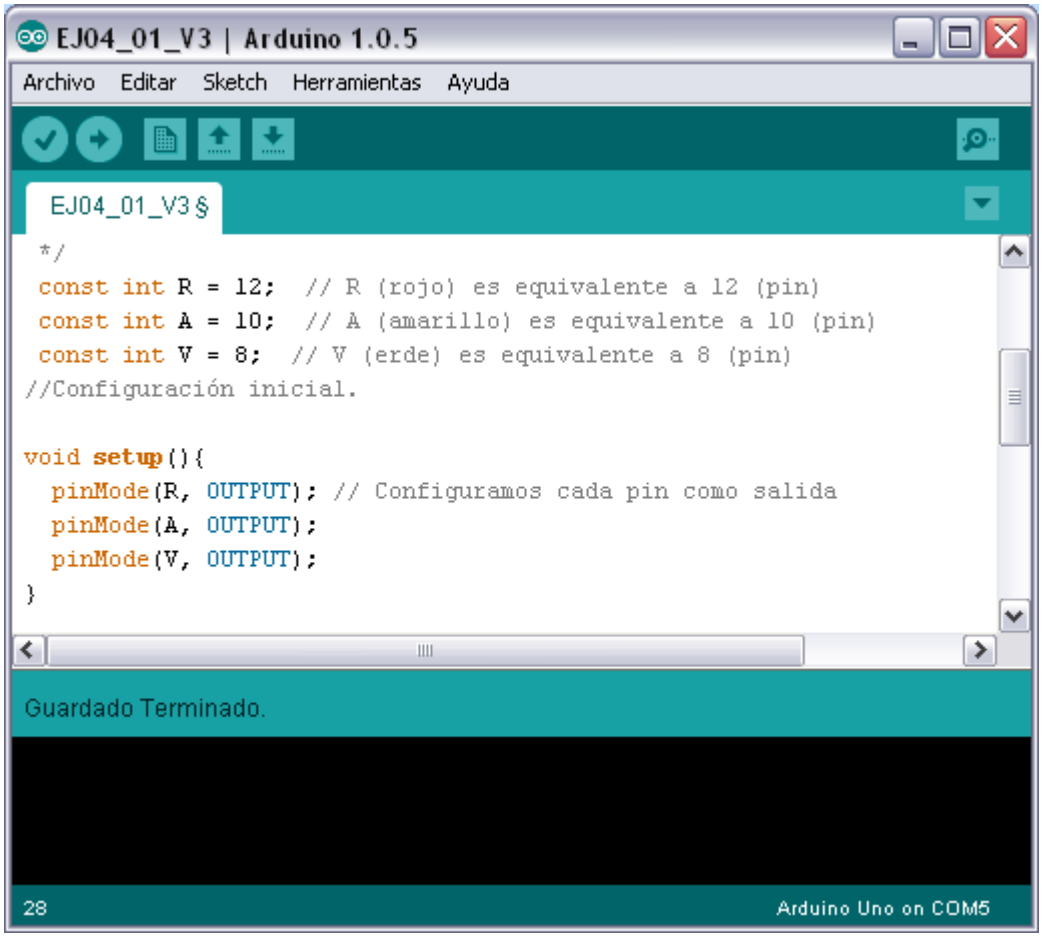
En este caso **`semaforo`** necesita de dos datos (parámetros) para hacer su tarea: el primero es el **color** (R, A o V) que en realidad es el **pin** donde está conectado el LED.

El segundo **parámetro** que necesita **`semaforo`** es el **tiempo** de duración en segundos.

Para que este sketch funcione correctamente hay que relacionar cada pin de salida que utilicemos con una letra. Esto se hace a través de las **constantes**.

La palabra clave **`const`** delante de un identificador significa **constante**, es decir, su valor no se puede cambiar durante la ejecución del programa. Si intenta asignar un valor a una variable **`const`** recibirá un error del compilador.

En mi caso como conecté el LED **rojo** en el pin **12**, el **amarillo** en el **10** y el **verde** en el **8**, tuve que definir esas tres constantes al principio del código antes de **`void setup().`**



Cuando se **define** una **constante** se establece una relación entre un **identificador** (letra o palabra) y un **valor** (que en este caso representa el número de pin).

Es importante la palabra reservada **`const`** (const) y el tipo **`int`** (entero).

Usar constantes tiene grandes **ventajas**:

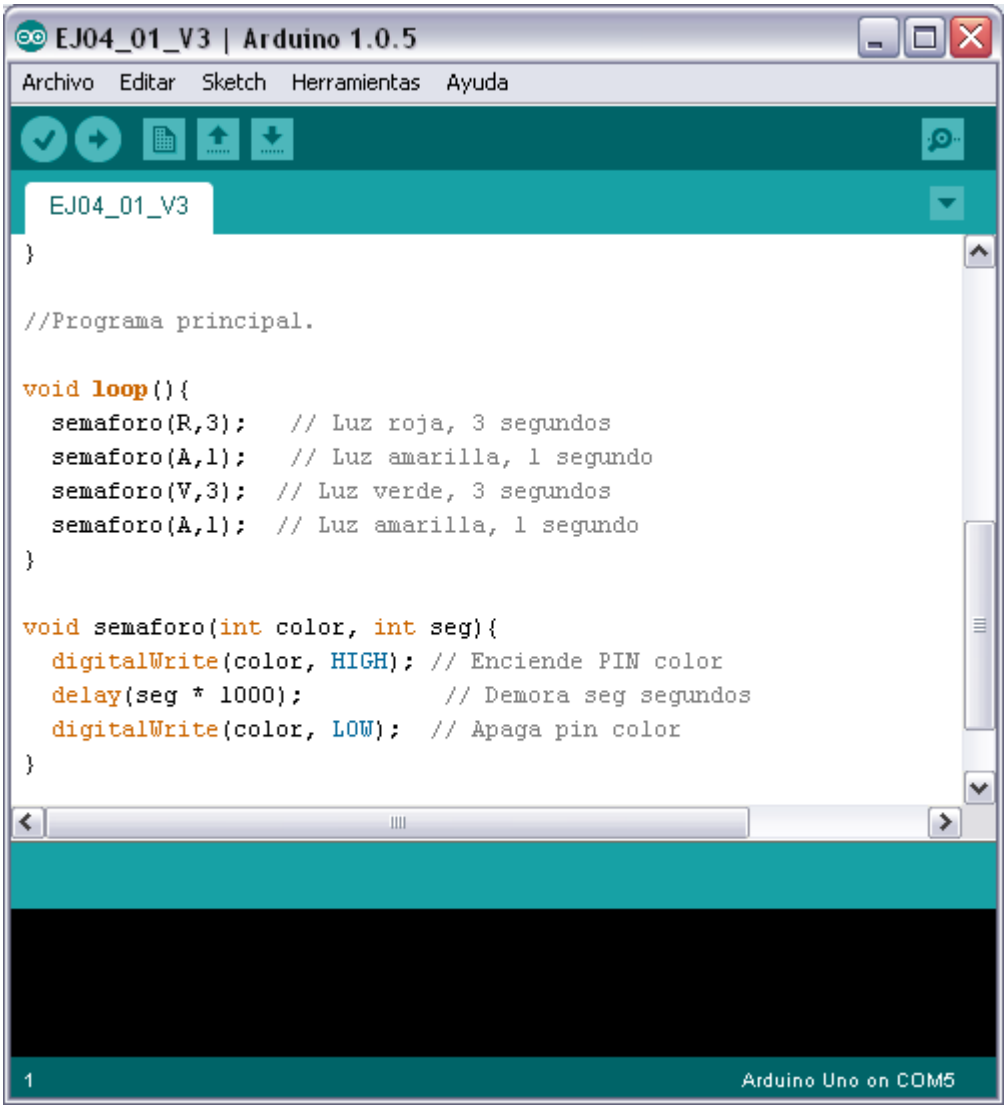
1. El identificador R es más fácil de relacionar con el led rojo que el número 12 donde está conectado. Tranquilamente se pudo haber usado ROJO en vez de R.
2. Si en todo nuestro programa usamos R y decidimos más tarde cambiar el pin donde conectamos el LED rojo, solamente tenemos que cambiar una sola línea, la de la **definición** de la constante.

La misma imagen muestra como se usa la función **`pinMode`** para configurar los pines como salida.

Y ahora falta definir la función semáforo que va a reemplazar a las viejas funciones **`rojo()`**, **`amarillo()`** y **`verde()`** que ya quedaron obsoletas.

Veamos en detalle como funciona la función semáforo.

Una función tiene dos etapas: cuando es **definida** y cuando es **usada**. Hasta ahora le mostré como se usaba, ahora veremos como está construida.



La primera **diferencia** que encontramos es que hay cosas escritas dentro de los paréntesis de semáforo: una variable entera denominada **`color`** y otra también entera denominada **`seg`**.

Estas variables van a almacenar una **copia** de los datos (argumentos) que el programador escriba cuando **invoque** a la función en el programa principal.

Por ejemplo cuando en el programa dentro de **`loop`** se escriba **`semaforo(A,5);`** la variable **`color`** valdrá **`A`** (10 en este ejemplo) y la variable **`seg`** tendrá el valor 5 (que representará 5 segundos).

Observe como son utilizadas estas variables en el cuerpo de la función tanto para el **encendido** y el **apagado** del LED correspondiente como en el **cálculo** de la **demora**.

Es importante que entienda este proceso. Si tiene alguna duda consulte con el profesor.

EJ-0502) Escriba la versión 3 del ejercicio anterior (EJ-0401_V2) usando como referencia los fragmentos de código que le fui mostrando en las páginas anteriores. Tendrá que adaptar las constantes al hardware que UD diseñó. El resto del sketch debería funcionarle correctamente.

Entradas digitales

Otro modo de configurar un pin digital del Arduino es como **entrada**. La manera de hacerlo es a través de la función **`pinMode`**.

Arduino a las entradas digitales las interpreta de la siguiente manera:

> Un valor de tensión **entre 3V** y **5V** lo interpreta como un 1 lógico o **HIGH**.
>
> Un valor de tensión **entre 0V** y **2V** lo interpreta como un 0 lógico o **LOW**.

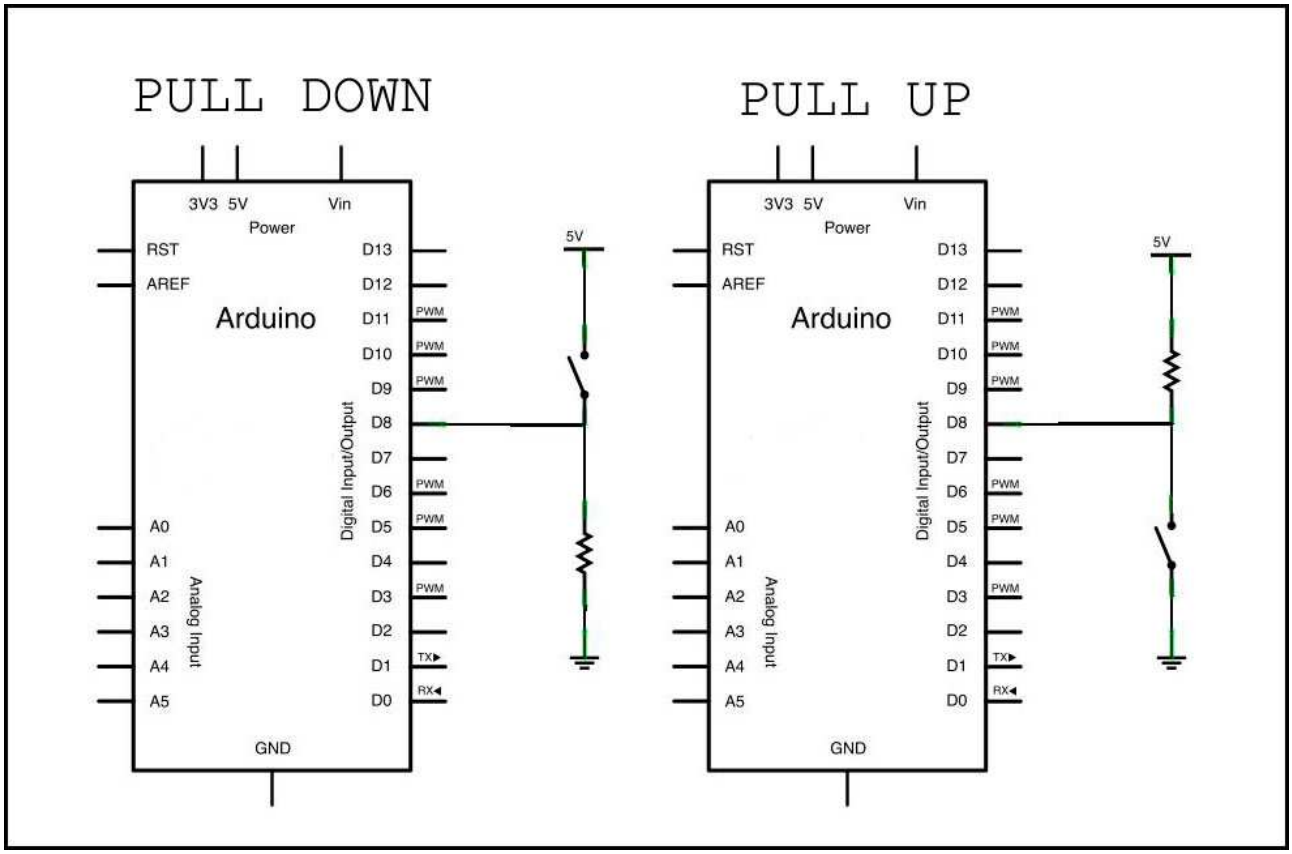
Los pines de Arduino (Atmega) por defecto son de **entrada**, por lo que no sería necesario configurarlos explícitamente como entradas con **`pinMode()`**. Sin embargo es una buena costumbre hacerlo.

Se dice que los pines configurados como entradas están en estado de **alta** impedancia.

Una forma de explicar esto es que los terminales de entrada hacen demandas extremadamente **pequeñas** de corriente en el circuito que están muestreando. Se suele decir que equivale a que haya una resistencia en serie de 100 megaohmios frente al pin.

Diseño del hardware para una entrada digital

Un dispositivo conectado a una entrada digital debe estarlo de modo tal que puedan estar presentes los dos estados en forma correcta. Veamos como debe conectarse correctamente un pulsador a una entrada digital del Arduino. Hay **dos** modos:



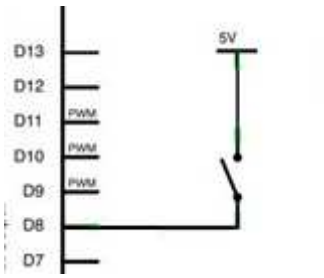
En las dos imágenes de la página anterior observamos una fuente de 5V, un **pulsador** conectado al pin digital 8 y una **resistencia**. La **diferencia** entre ambas configuraciones es el valor presente en la entrada cuando no está activado el pulsador y el valor que está presente cuando se lo pulsa.

En la conexión **PULL DOWN** se puede observar que cuando el pulsador está **abierto** la entrada digital está conectada a masa a través de la resistencia: esto quiere decir en esta posición la entrada presenta **0V** o un **0 lógico**. Cuando se **presiona** el pulsador se hace presente en el pin de entrada una tensión de **5V**, es decir un **1 lógico**.

En la conexión **PULL UP** sucede todo al revés: se presenta un **1 lógico** cuando **no se presiona** el pulsador y un **0 lógico** cuando **se activa**.

Ambas conexiones son **válidas** y debe tenerse en cuenta cuando se escriba el **código**.

Es muy **tentador** conectar un pulsador en una entrada digital como muestra la imagen de la derecha.

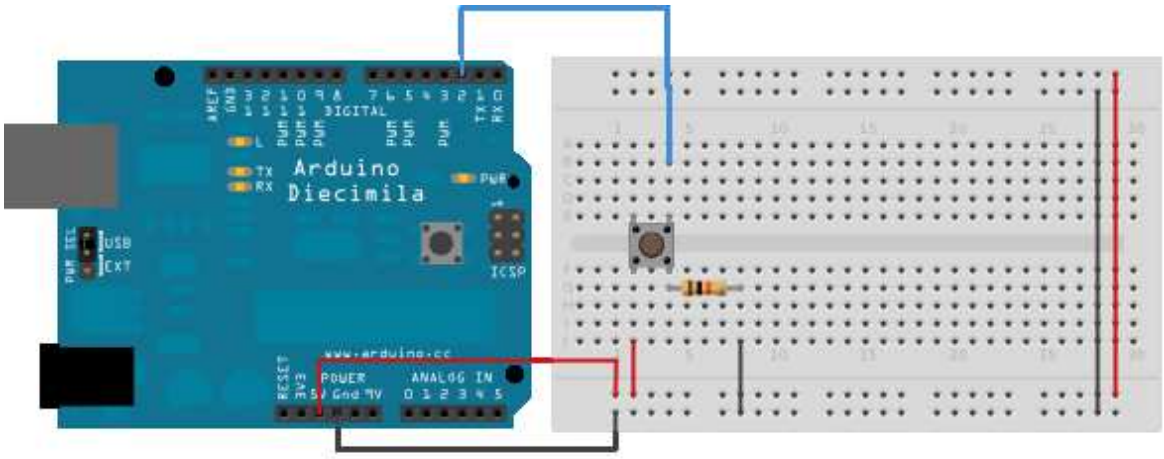


Pero esta manera es **incorrecta**. Observe que cuando se presione el pulsador en el pin D8 va haber una tensión de 5V con lo cual interpretará el estado **alto** sin problemas.

Pero cuando el pulsador está **abierto** el pin se encuentra al "**aire**" y eso es los que debe **evitarse**.

Cuando un pin está al **"aire"** es susceptible de recibir señales incorrectas denominadas **ruido** electromagnético lo que provocaría la interpretación **errónea** de una señal de tensión de entrada ya que no es producto de la activación del pulsador sino de la presencia de factores exteriores al circuito.

La siguiente imagen muestra como debería conectar un pulsador a su placa Arduino.



La imagen muestra un Arduino Diecimila pero funciona igual en un Arduino Uno.

Observe que en este caso se conectó una resistencia de **pull down** de **10 K** que es el valor típico para este tipo de circuitos.

El **pin 2** se va a utilizar como **entrada digital** y además la alimentación de protoboard se toma de los pines **power** de la placa Arduino.

Software: sketch Button

En esta oportunidad vamos a analizar uno de los ejemplos que trae el entorno de desarrollo de Arduino. Se trata del sketch ***Button*** que se puede cargar desde ***Archivo – Ejemplos – 02.Digital – Button.***

Es un buen ejercicio empezar **familiarizarse** con estos **ejemplos** porque puede ser un buen punto de partida para aprender las **potencialidades** del Arduino.

Vamos a ir tratando de explicar cada una de las partes sin hacer una traducción literal sino de **interpretar** el código.

Button

```
/*
  Button

 Turns on and off a light emitting diode(LED) connected to digital
 pin 13, when pressing a pushbutton attached to pin 2.


 The circuit:
 * LED attached from pin 13 to ground
 * pushbutton attached to pin 2 from +5V
 * 10K resistor attached to pin 2 from ground

 * Note: on most Arduinos there is already an LED on the board
 attached to pin 13.


 created 2005
 by DojoDave <http://www.0j0.org>
 modified 30 Aug 2011
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/Button
 */
```

La mayoría de las veces encontramos al principio del código un comentario de varias líneas que tiene el **nombre** del sketch, el **objetivo** del mismo, como conectar el **hardware** para que el programa funcione, el **autor**, **fecha** de creación, tipo de **licencia** e **información** adicional que puede servir para tomar contacto con el desarrollador o profundizar sobre el tema.

Aquí nos interesa la descripción del programa: **“*Enciende y apaga un LED conectado al pin 13 cuando se presiona el pulsador conectado en el pin 2”.***

En la parte explicativa del circuito, el autor nos comenta como va conectado el pulsador (desde +5V con una resistencia de 10K a tierra) y el LED en el pin 13, que en muchos **Arduino** ya vienen integrados a la placa y no es necesario agregarlo exteriormente. Es el esquema de hardware que se observa en la imagen de la página anterior.

```
// constants won't change. They're used here to
// set pin numbers:
const int buttonPin = 2;     // the number of the pushbutton pin
const int ledPin =  13;      // the number of the LED pin

// variables will change:
int buttonState = 0;         // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}
```

Las siguientes líneas sirven para **definir** los pines mediante dos **constantes** enteras: **`buttonPin`** va a identificar al **pulsador** (**pin 2**) y **`ledPin`** identifica al **LED** (**pin 13**).

Luego se define la **variable** entera **`buttonState`** y se le asigna el valor **inicial 0**. A diferencia de las constantes anteriores, **`buttonState`** puede **variar** su valor durante la ejecución del programa ya que, como su nombre lo indica, va a almacenar el **estado** del **pulsador**: **0** quiere decir pulsador **inactivo** y **1** quiere decir pulsador **activado**.

Como en los programas anteriores, dentro de **`void setup()`** se configuran los pines mediante **`pinMode`**, en este caso, **`ledPin`** como **salida** y **`buttonPin`** como **entrada**.

Por último tenemos el programa principal dentro del bloque **`void loop().`**

```
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Y aquí aparecen algunas instrucciones conocidas como **`digitalWrite`** pero otras nuevas como **`digitalRead`** y la estructura de decisión **`if..else.`**

La primera instrucción dentro de loop sirve para **leer** el estado de la **entrada** y **almacenarlo** en la variable **`buttonState`**. Para leer la entrada se usa la función **`digitalRead`**.

`digitalRead()`

Descripción

Lee el valor de un pin digital y devuelve su estado: **HIGH** o **LOW.**

Sintaxis

`digitalRead(pin);`

Parámetros

pin: el número de pin que se desea leer.

Devuelve

Esta función devuelve el valor que representa el estado del pin **HIGH** (alto) o **LOW** (bajo), por lo tanto no puede ser utilizada sola como lo hacíamos con **`digitalWrite`**.

En síntesis la instrucción: **`buttonState = digitalRead(buttonPin);`**

sirve para leer (**`digitalRead)`** el estado del pulsador (**`buttonPin)`** y el valor devuelto por la función se almacena en la variable **`buttonState`**.

El operador **=** se llama de **asignación** y le dice al microcontrolador que **evalúe** cualquier valor o expresión en el lado **derecho** del signo igual y lo **almacene** en la **variable** a la **izquierda** del signo igual.

A continuación vemos la estructura **`if..else`** que quiere decir **`si…sino`**

```
if (condición)    // si se cumple la condición
 {
   … //Acciones que se llevan a cabo cuando condición es verdadera
 }
else              // sino
 {
   … //Acciones que se llevan a cabo cuando condición es falsa
 }
```

La estructura **`if`** verifica si la **condición** encerrada entre paréntesis es **verdadera** o **falsa.** Para ello se utiliza un operador de relación o comparativo:

Operadores Comparativos

== (igual a)

!= (distinto de)

< (menor que)

> (mayor que)

<= (menor o igual que)

>= (mayor o igual que)

Dentro del paréntesis de una instrucción **`if..else`** se realiza una comparación que puede arrojar dos resultados: **verdadero** o **falso**. En el ejemplo que estamos analizando vemos

```
if (buttonState == HIGH)
```

que significa: si **`buttonState`** es **equivalente** a alto, o sea, si se **presionó** el pulsador.

Si lo anterior es cierto, se ejecutan las acciones dentro del siguiente bloque, en cambio si es falso, se ejecutarán las acciones que están dentro del bloque que sigue a la palabra **`else`**.

En el ejemplo cuando la entrada esté en estado **HIGH** se ejecuta **`digitalWrite(ledPin, HIGH)`** para encender el LED y cuando la entrada **no esté** en estado HIGH se ejecutará **`digitalWrite(ledPin, LOW)`** para apagar el LED.

Es obvio decir que no se utilizaría un Arduino para encender y apagar un LED, pero este ejemplo ilustra de qué manera se puede leer el estado de un pin de entrada, almacenarlo en una variable y que el programa esté preparado para decidir qué acción tomar de acuerdo al mismo.

Antes de terminar, una aclaración. Es un error común confundir = con ==.

- Se usa `=` cuando se desee **guardar o asignar** un valor a una variable.
- Se usa `==` cuando se desee **comparar** dos valores y ver si son **equivalentes**.

Actividades

EJ-0503) Conecte en el protoboard el pulsador de acuerdo al modelo de hardware mostrado y compruebe el funcionamiento del sketch ***Button***.

EJ-0504) Modifique el EJ-502 para que el semáforo se comporte de la siguiente manera:

- Cuando **no se accione** el pulsador que **parpadee** el LED amarillo.

  Cree una función para tal efecto pero que no sirva solo para el amarillo sino para el **color** que se desee y se pueda elegir la **duración** del mismo en **milisegundos**.
- Cuando se **accione** el pulsador que haga la secuencia **normal** del semáforo usando la función **`semaforo`**.