CAPÍTULO 4:

LUCES PARA MI ROBOT

Luces para mi robot

Es un capìtulo muy importante en nuestro curso, ya que vamos a realizar el primer código de programación escrito y un circuito electrónico.

Desarrollar este pensamiento abstracto, necesario para la programación, es lo más difícil. Sin embargo, hemos visto a muchas personas desarrollarlo desde cero. No sólo eso, muchas de esas personas ahora tienen un empleo como programadores.

Si quieres aprender a programar desde cero debés tener la actitud correcta. La motivación es importante, pero la motivación se agota fácilmente.

Necesitás desarrollar una disciplina mientras adquirís el gusto por la programación. Debés saber que vas a necesitas cientos de horas de aprendizaje para lograr hacer las cosas más básicas. Y tenés saber que habrán momentos de mucha frustración, dale lugar al error y a la experimentación.

Esa actitud, el tiempo suficiente y una buena guía, nosotros estamos para eso, es todo lo que necesitas para aprender a programar.

Consejo:

Tomate tu tiempo a la hora de armar el circuito electrónico, muchas fallas pueden venir por malas conexiones.

Diodo LED

Si tenemos que dar una definición concreta, un LED es un diodo emisor de luz, lo que en inglés se conoce como *Light Emitting Diode*.

Cada LED está formado por un **encapsulado** y tiene dos terminales o patas; generalmente una pata es más larga que la otra para facilitar su distinción. La más larga se denomina **ánodo** (polo positivo) y la más corta es el **cátodo** (polo negativo).

Para que el LED sea capaz de emitir luz es necesario que el ánodo esté conectado al positivo y el cátodo al negativo. Más adelante veremos que cada LED se suele usar con una resistencia, la cual varía según el diodo en cuestión.

¿Cómo circula la corriente en un LED?

Un diodo led es un dispositivo electrónico capaz de permitir el paso de la corriente eléctrica en un único sentido. Esto significa que será imposible que la corriente circule en el sentido opuesto ya que el LED sólo puede recibir energía, emitiendo luz cuando esto ocurre, o mantenerse apagado cuando la corriente eléctrica no fluye a través de él.

Cuando los LED emiten luz se encuentran en **polarización directa**. Si el diodo se conectara con sus patas al revés sería imposible que emita luz, aún cuando esté conectada la batería, ya que se encontraría en **polarización inversa**, es decir, con los polos invertidos lo que impediría el traspaso de la energía.

Ventajas de la iluminación LED

Las ventajas de la iluminación LED frente a la iluminación incandescente son muchísimas, pero sólo vamos a listar tres de las más representativas:

- **Consumen menos energía**, ya que no se desperdicia energía en calor para generar la iluminación. Si lo comparamos con calentar un filamento para hacerlo brillar nos damos cuenta que estamos empleando solo la energía que se necesita para dar iluminación sin calentar absolutamente nada.

- **Tienen mayor vida útil** que cualquier otro tipo de iluminación.

- **Podemos obtener diferentes tipos de colores con solo variar la corriente que circula por ellos** en el caso de los LEDs RGB. Esto es una ventaja increíble, sobre todo en los conciertos y espectáculos, algo que actualmente también encontramos en las fachadas de algunos monumentos y museos.

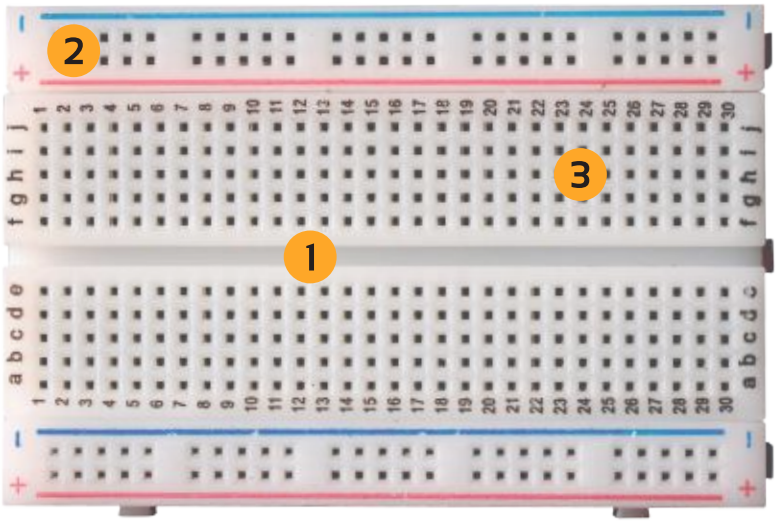
La protoboard

Como el nombre lo indica, se trata de una tableta para montar prototipos, lo que significa que estos montajes son eventuales, por lo que probamos y volvemos a desmontar los componentes, quedando así la protoboard libre y lista para el próximo experimento.

La protoboard, esencialmente, es una placa agujereada con conexiones internas dispuestas en hileras de modo que forman una matriz de taladros a los que podemos directamente "pinchar" componentes y así formar el circuito electrónico deseado. En esta gran cantidad de orificios es donde se pueden insertar con facilidad los terminales de los elementos que conforman el circuito.

Estructura de una protoboard

Si observamos con detenimiento veremos que los orificios están etiquetados con números en forma horizontal (1,2,3,...) y con letras (a,b,c,d...,j) en forma vertical. Esto es así para evitar errores en la interconexión de los diferentes elementos del circuito.



Internamente las protoboard están compuestas por láminas que unen diferentes líneas de orificios. En base a esto, podemos dividirla en tres regiones:

1. • **Canal central:** es la región localizada en el medio de la tableta. Se utiliza para colocar los circuitos integrados.

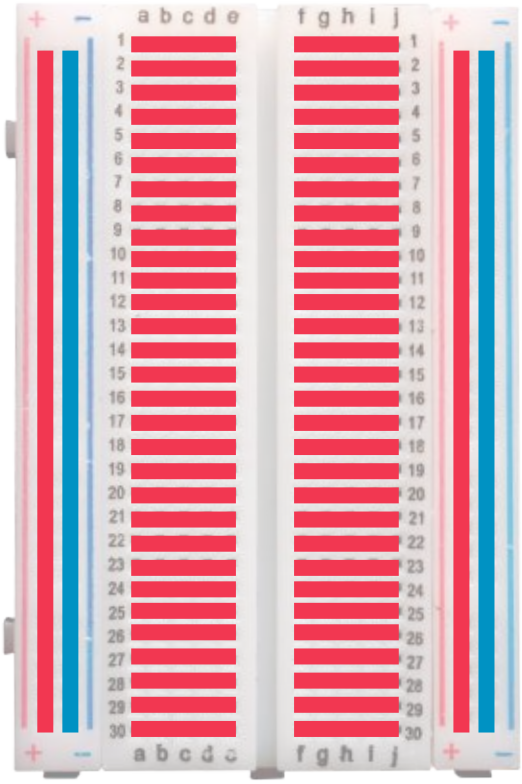
2. • **Buses:** estos se localizan en ambos extremos de la protoboard. Las líneas rojas representan los buses positivos o de voltaje (5V), mientras que las líneas azules representan los buses negativos o de conexión a tierra (0V). Los buses conducen de acuerdo a esas líneas y no existe conexión física entre ellas. Normalmente la fuente de poder se conecta en estos buses para proveer de corriente eléctrica a toda la protoboard.

   Nota: Los circuitos integrados se colocan en la parte central de la protoboard con una hilera de patas en la parte superior del canal central y la otra hilera en la parte inferior del mismo. Es habitual conectar los buses de una sección con los de la otra para que ambos lados de la tableta reciban alimentación eléctrica.

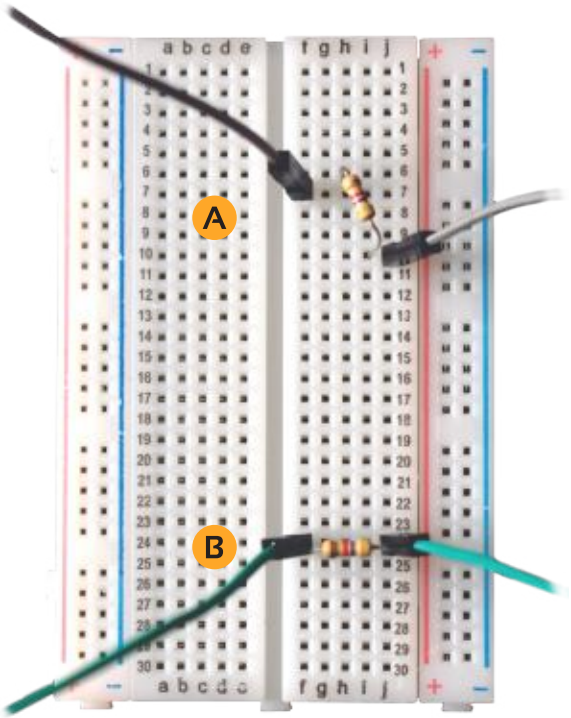
3 **Pistas:** estas se localizan en la parte central de la tableta y, a diferencia de los buses, conducen según la hilera en que se encuentran. Es decir que, mientras los buses recorren la proto de manera horizontal, las pistas lo hacen de manera vertical.

¿Cómo se conduce la energía en las protoboards?

Para que se entienda mejor, en el siguiente diagrama veremos el sentido en el que viaja la corriente eléctrica a través de la tableta perforada ya que estas vías están determinadas por una serie de laminillas internas que se encuentran recubiertas por el plástico de la carcasa y no pueden verse a simple vista.



Referencias
Conexión a tierra (0V)
Corriente eléctrica (5V)



Nota: Al trabajar en una misma hilera B la corriente tiende a circular por el paso de menor resistencia, por lo que si colocamos una resistencia en la misma hilera que los cables la electricidad correrá de cable a cable sin pasar por la resistencia. Por lo tanto, **recomendamos trabajar con los cables en dos hileras diferentes, usando las resistencias como puente entre ambas** como se puede ver en A

Ventajas de las protoboards

Las protoboard tienen la ventaja de ser de rápida ejecución, sin necesidad de realizar soldaduras, ni empalmes ni tampoco utilizar otras herramientas. Si el circuito bajo prueba no funciona de manera satisfactoria, se puede modificar sin afectar los elementos que lo conforman. Sin embargo los circuitos que montemos deberán ser más bien sencillos para evitar fallos y complicaciones en exceso al tener cruzados demasiados cables y complementos.

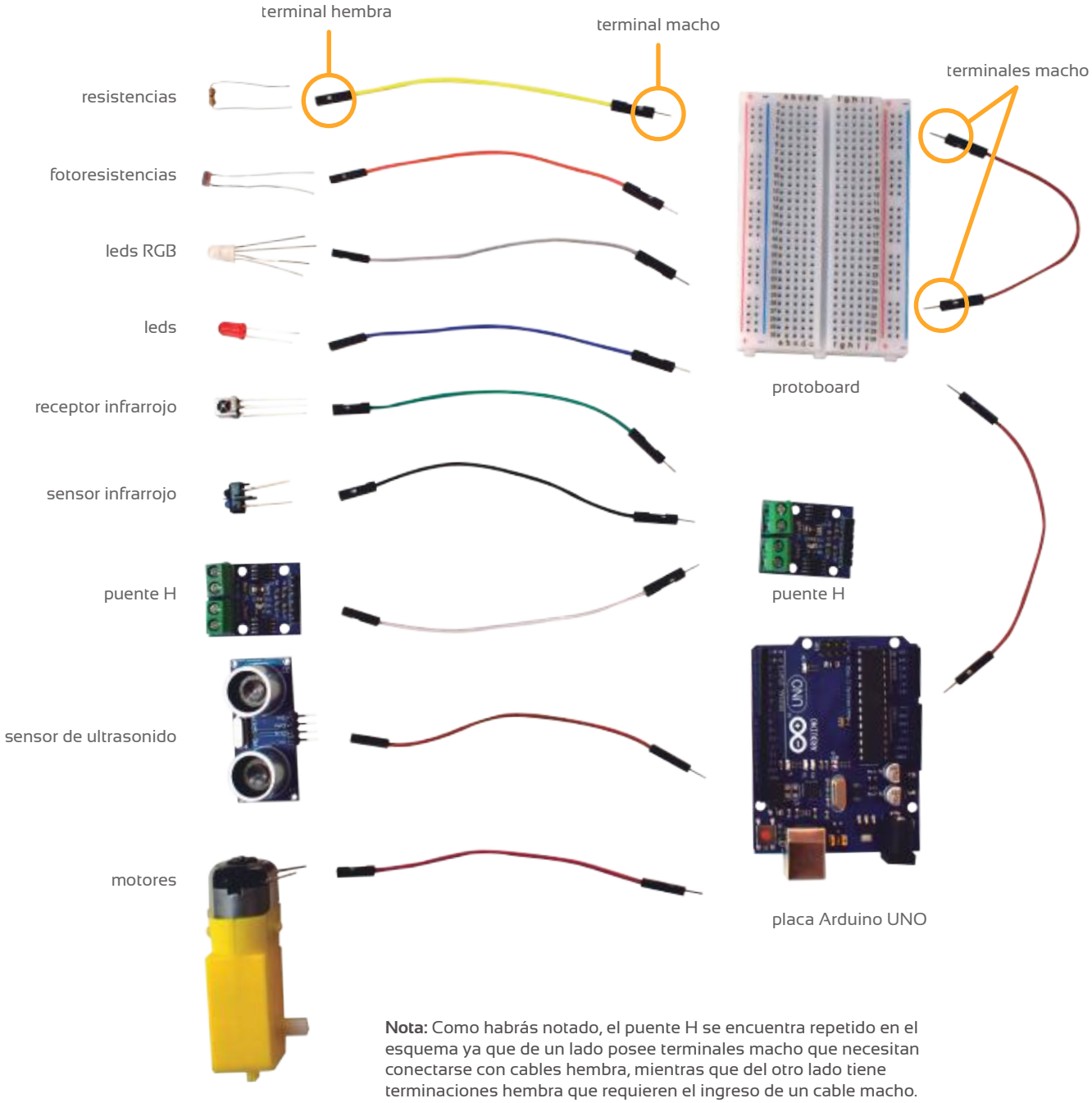
Se les puede conectar casi cualquier tipo de componente electrónico, incluyendo diferentes tamaños de circuitos integrados. Los únicos elementos que no se pueden conectar a la protoboard son elementos que tienen terminales muy gruesos. Normalmente los elementos se conectan sin problemas en forma externa, a veces con ayuda de cables.

Cable puente

Al igual que cualquier enchufe de nuestra casa, encontramos terminales hembras (el enchufe clásico de pared) y machos, como los enchufes de los electrodomésticos. Su conexión es intuitiva: el macho y el hembra se conectan entre sí, mientras que hembra-hembra o macho-macho no lo hacen.

Un cable puente es un cable con un conector en cada punta (ya sea macho o hembra), que se usa normalmente para interconectar entre sí los componentes en un circuito electrónico. En el kit vamos a encontrar cables de 10cm. y 20cm. los cuales nos van a facilitar los distintos ensambles eléctricos. Algunos tienen una terminación macho y otra hembra, y otros tienen ambos terminales machos.

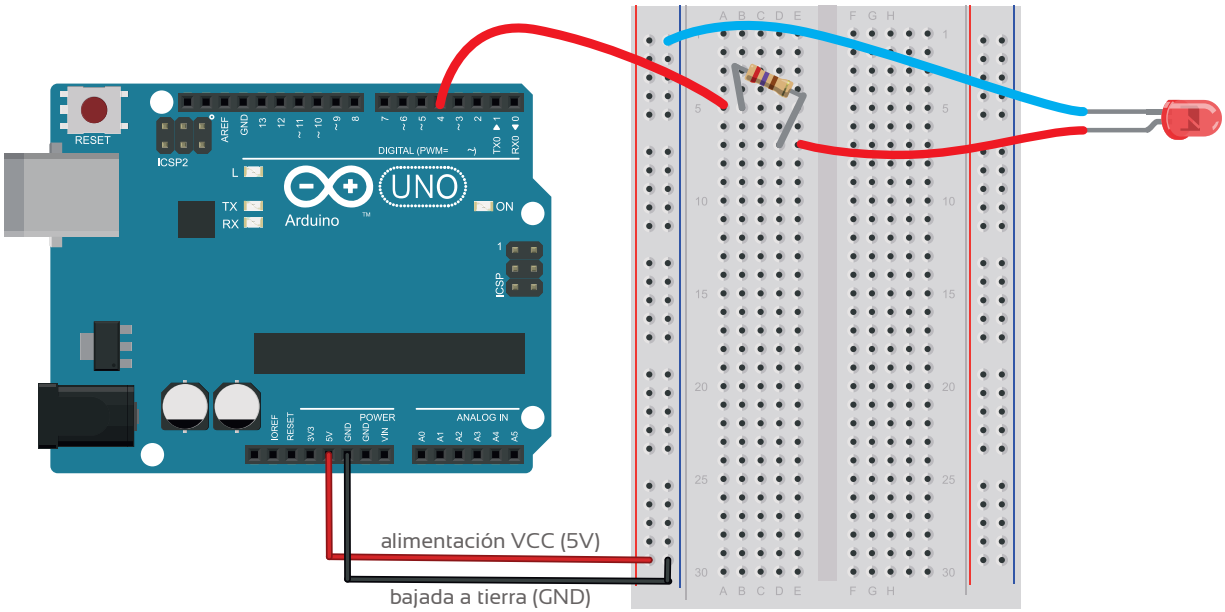
A continuación veremos qué componentes de nuestro robot precisan un conector hembra y cuáles un conector macho.



Nota: Como habrás notado, el puente H se encuentra repetido en el esquema ya que de un lado posee terminales macho que necesitan conectarse con cables hembra, mientras que del otro lado tiene terminaciones hembra que requieren el ingreso de un cable macho.

Conexión de componentes a la protoboard

En las siguientes imágenes veremos tres maneras diferentes de construir el mismo circuito. Esto significa que no precisas replicar exactamente la posición de cada cable y componente tal y como aparece en los ensambles de ejemplo, sino que puedes manejarte entre ciertos márgenes para que el espacio de trabajo te resulte más cómodo.

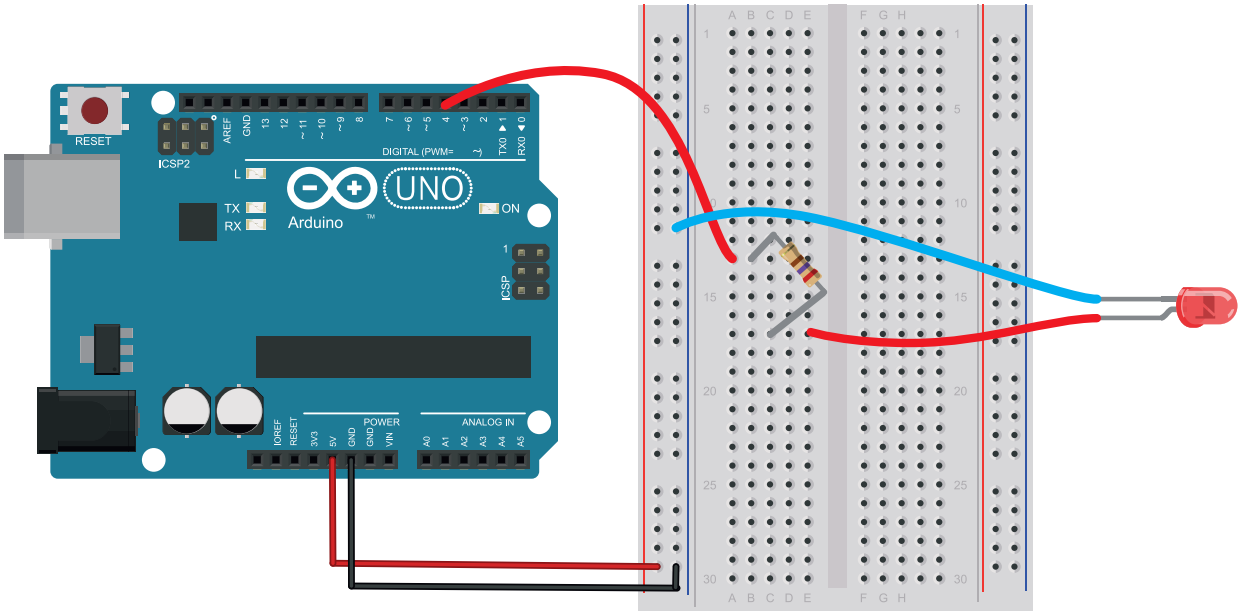


Opción 1

Para ejemplificar las maneras alternativas en que podemos lograr un mismo circuito usaremos un led rojo. Siempre haremos que la electricidad parta del pin digital 4 (+), moderando el voltaje con una resistencia de 270Ω , circulando por el led y terminando con la bajada a tierra al bus negativo de la protoboard (-).

Notas:

- Usamos el pin 4 como ejemplo, podremos usar cualquier pin digital.
- Las resistencias no deber ser conectadas sobre la misma pista, sino oblicuas o cruzadas.

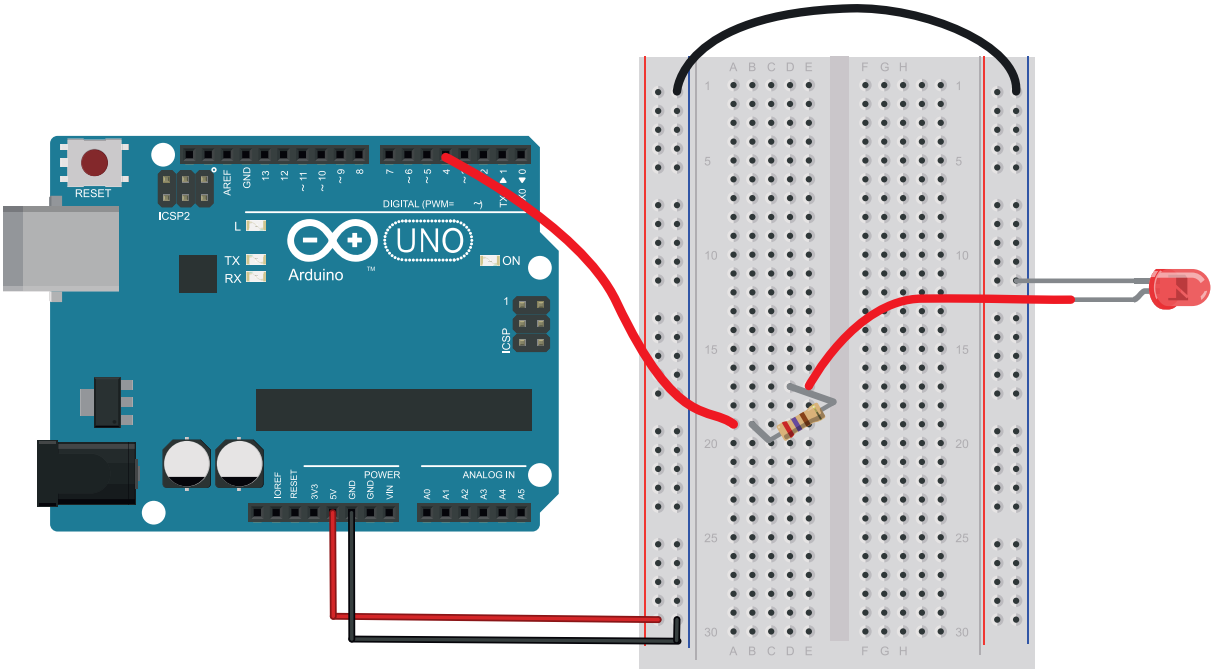


Opción 2

Nos tomemos un momento para comparar las diferencias entre este circuito y el de la opción 1, ambos diseñados para cumplir con la misma función. Observamos un cambio de posición y distancia en la conexión de la resistencia.

Notas:

- La resistencia está colocada con la banda roja hacia el otro lado, pero su funcionamiento no se altera, ya que no posee terminales positivas y negativas.
- Puedes ubicar sus terminales en las pistas que quieras, con menor o mayor diferencia, manteniendo siempre la posición oblicua.



Opción 3

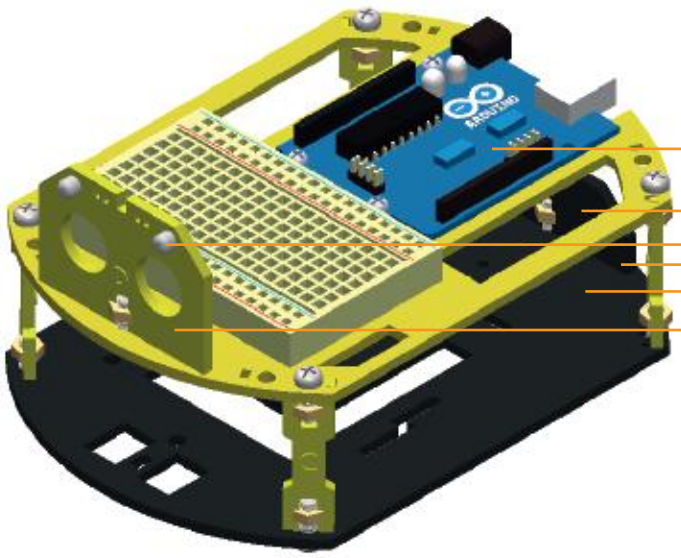
Esta última opción cumple la misma función que el uno y el dos. Vemos que la resistencia nuevamente ha cambiado de posición pero lo importante es que se mantiene con sus pines conectados en dos pistas diferentes por lo que la vemos *inclinada* o en posición oblicua. El otro cambio, tal vez más notable, es que el led rojo está bajado a tierra sin necesidad de un cable, sino conectado a través de su propio pin. Sin embargo, para que esta bajada a tierra (GND) sea posible fue necesario conectar el bus negativo de la banda izquierda con el bus negativo de la banda derecha a través del cable superior negro.

Notas:

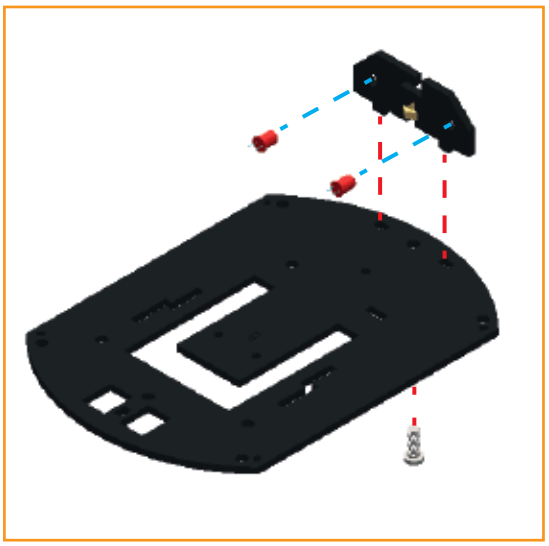
- Puedes conectar los complementos directamente en los pines de la protoboard (a los buses + y -, ó a cualquiera de las pistas).
- La protoboard trabaja en dos mitades, debes alimentar ambas partes a través de puentes para usar cada una de ellas. La izquierda esta siendo alimentada por el Arduino, la derecha debe alimentarse generando *"puentes"* de positivo a positivo y de negativo a negativo (en este ejemplo sólo tendimos el puente negativo, ya que no necesitamos la bonda positiva).

Ensamble

A continuación veremos la estructura y componentes necesarios para realizar el primer proyecto, con el que encenderemos 2 luces led blancas y 2 rojas.

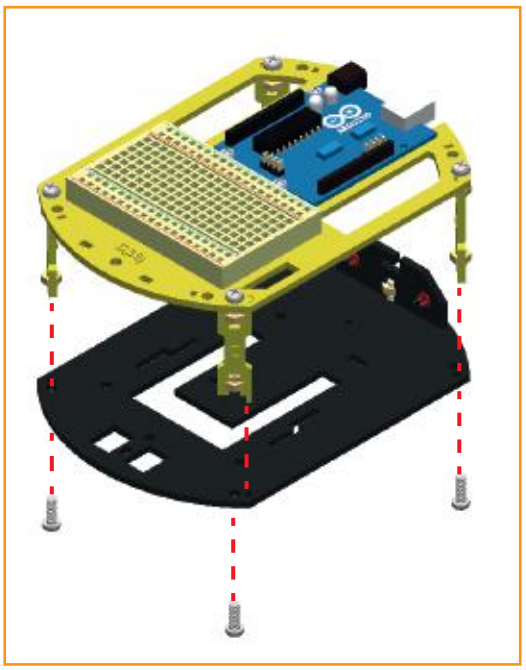


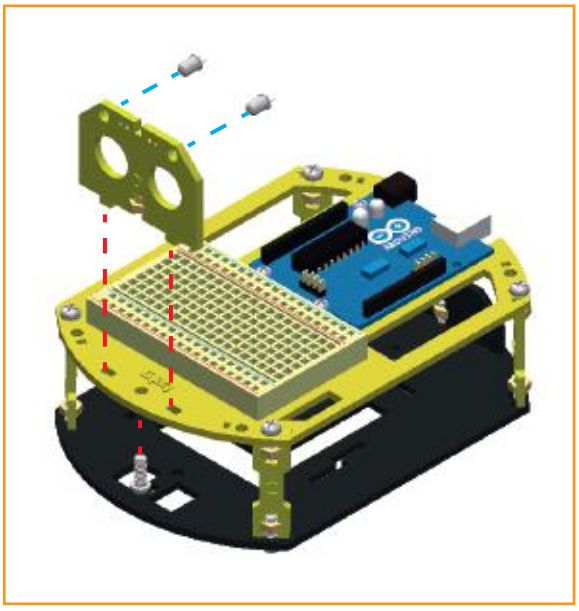
Materiales:
-Estructura de Hola Mundo
-Pieza B
-2 leds Blancos
-Pieza D
-2 leds Rojos
-Pieza G
-Bulón 4x12mm x 6 unidades
-Tuerca 4mm x 6 unidades



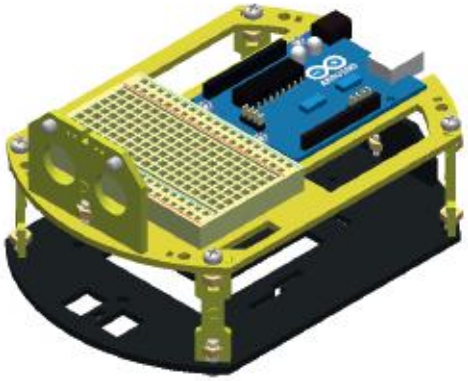
Paso 1:
Colocar los leds rojos en las perforaciones de la pieza G. Luego abulonar la pieza G en el chasis B.

Paso 2:
Colocar las columnas C del ensamble del capìtulo "Hola mundo" en las perforaciones de la pieza B, logrando una estructura de 2 pisos.



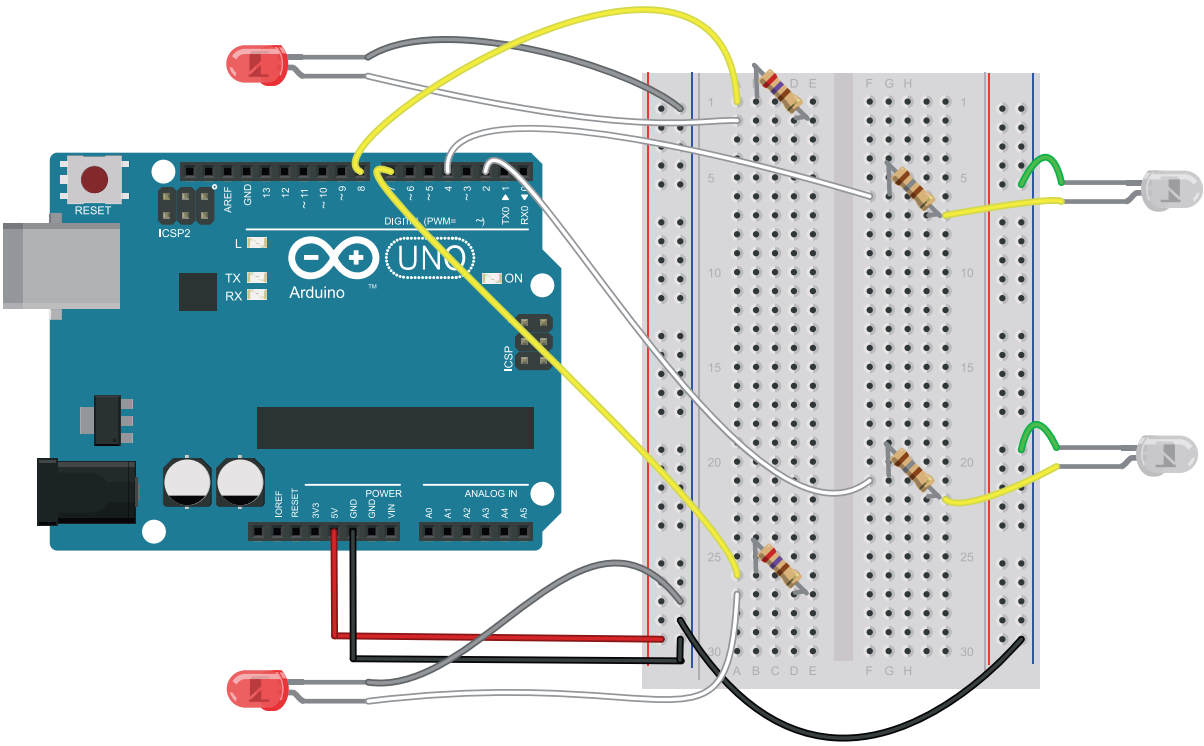


Paso 3:
Colocar los leds blancos en las perforaciones de la pieza D. Luego abulonar la pieza D en el chasis A.



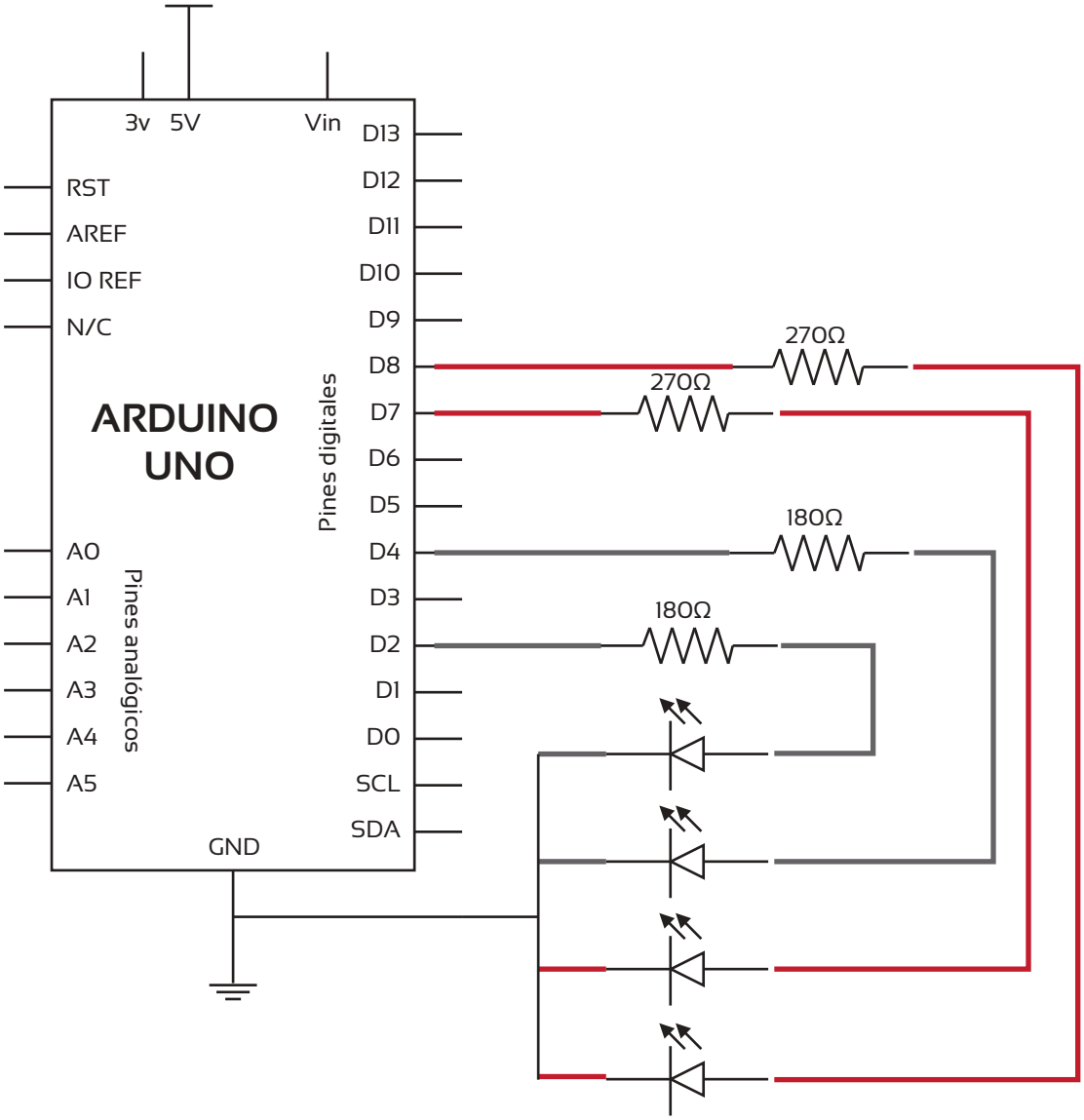
Circuito electrónico

Finalmente observamos las conexiones de nuestro circuito electrónico.



Componentes electrónicos:
-2 leds Blancos
-2 resistencias de 180Ω
-2 leds Rojos
-2 resistencias de 270Ω o 220Ω
-4 cables M-M
-8 cables M-H

Circuito electrónico en símbolos



Código de programación

Desarrollaremos el código de programación para el parpadeo de las luces y haremos algunos comentarios en el costado de las líneas para ir explicando cada una de sus partes.

NOTA: Podrán observar que hay muchas lineas de código vacías, no es necesarios espaciarlas, siemplemente lo hacemos así para que vayan bien separados los comentarios para cada parte de código).

```
1  int ledBlancoDerecha = 2;
2  int ledBlancoIzquierda = 4;
3  int ledRojoDerecha = 7;
4  int ledRojoIzquierda = 8;
5  int ledTest = 13;
6
7
8  void setup() {
9
10
11
12
13
14
15   pinMode(ledBlancoDerecha, OUTPUT);
16   pinMode(ledBlancoIzquierda, OUTPUT);
17   pinMode(ledRojoDerecha, OUTPUT);
18   pinMode(ledRojoIzquierda, OUTPUT);
19
20
21
22   digitalWrite(ledBlancoDerecha, LOW);
23   digitalWrite(ledBlancoIzquierda, LOW);
24   digitalWrite(ledRojoDerecha, LOW);
25   digitalWrite(ledRojoIzquierda, LOW);
26
27
28   pinMode(ledTest, OUTPUT);
29   digitalWrite(ledTest, LOW);
30
31   digitalWrite(ledTest, HIGH);
32   delay(500);
33   digitalWrite(ledTest, LOW);
34   delay(500);
35   digitalWrite(ledTest, HIGH);
36   delay(500);
37   digitalWrite(ledTest, LOW);
38   delay(500);
39   digitalWrite(ledTest, HIGH);
40   delay(500);
41   digitalWrite(ledTest, LOW);
42   delay(500);
43 }
44
45
46 void loop() {
47
48
49   digitalWrite(ledBlancoDerecha, HIGH);
50   digitalWrite(ledBlancoIzquierda, HIGH);
51   digitalWrite(ledRojoDerecha, HIGH);
52   digitalWrite(ledRojoIzquierda, HIGH);
```

//Definimos variables: todas las variables tienen que declararse para ser utilizadas. Para declarar una variable se comienza por definir su tipo como int (entero), long (largo), float (coma flotante), etc, asignándoles siempre un nombre, y, opcionalmente, un valor inicial.

// Setup: Inicialización del pin de salida: La función setup(), recordemos, se invoca una sola vez cuando el programa empieza. Se utiliza parainicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en unprograma aunque no haya declaración que ejecutar.

//Inicializamos los pines de salida: Una funcion es una porcion de codigo que podemos ejecutar por medio de un nombre. pinMode sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida).

//y ahora apagamos todos los leds. Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW.

//apagamos el ledTest.

//Realizamos un saludo inicial, igual que el de "Hola mundo", haciendo parpadar el ledTest 3 veces, con una demora de 500ms.

//Cerramos la llave de la función loop, abierta en la línea 8

// loop : esta funcion corre llega al final y vuelve a comenzar. Funciona como un bucle.

//encendemos luces delanteras y traseras

```
53    delay(1000);
54
55
56
57
58
59    parpadeo(ledBlancoDerecha, 5, 500);
60
61
62    parpadeo(ledRojoDerecha, 10, 250);
63
64
65    parpadeo(ledRojoIzquierda, 5, 1000);
66
67
68    parpadeo(ledBlancoIzquierda, 10, 750);
69
70
71
72    digitalWrite(ledBlancoDerecha, LOW);
73    digitalWrite(ledBlancoIzquierda, LOW);
74    digitalWrite(ledRojoDerecha, LOW);
75    digitalWrite(ledRojoIzquierda, LOW);
76    delay(2000);
77
78  }
79
80  void parpadeo(int ledParpadea, int veces,
81  int tiempo) {
82
83
84
85
86  for (int j=0; j <= veces ; j++) {
87     digitalWrite(ledParpadea, LOW);
88     delay(tiempo);
89     digitalWrite(ledParpadea, HIGH);
90     delay(tiempo);
91
92
93
94
95
96   }
97
98   }
```

//Hacemos una espera de un segundo

//Aca estamos llamando a una funcion que se llama “parpadeo”, la cual no esta pre definida en librerías y la vamos a escribir más abajo.

//Hacemos parpadear el led blanco de la derecha 5 veces intervalo 1/2 segundo = 500 milisegundos.

//Hacemos parpadear el led blanco de la derecha 10 veces intervalo 1/4 segundo = 25 milisegundos.

//Hacemos parpadear el led blanco de la derecha 5 veces intervalo 1 segundo = 1000 milisegundos.

//Hacemos parpadear el led blanco de la derecha 10 veces intervalo 3/4 segundo = 750 milisegundos.

//ahora apagamos todas las luces y esperamos 2 segundos antes de que se repita el ciclo.

//Cerramos la llave de la función loop, abierta en la línea 46

//Ahora vamos a escribir la funcion "parpadeo". Para eso vamos a definir sus tres parámetros:
1- ledParpadea
2- veces
3- tiempo

//declaracion de funcion con sus parametros entre los parentesis La declaración la estructura for se usa para repetir un bloque de sentencias encerradas entre llaves unnúmero determinado de veces. Cada vez que se ejecutan las instrucciones del bucle se vuelve a testear la condición. La declaración for tiene tres partes separadas por (;)vemos el ejemplo de su sintaxis: (inicialización; condición; expresión).

//Cerramos la llave de for, abierta en la línea 86

//Cerramos la llave de la función parpadeo, abierta en la línea 80

MATERIAL COMPLEMENTARIO

Funciones

Antes de adentrarnos en el entorno de programación de Arduino es necesario que tengamos muy claro qué es y cómo está formada una función ya que ellas son la base de la programación en código.

Las funciones son bloques de programación que podemos ejecutar en cualquier momento al llamarlas por su nombre. Para ello es importante que tengamos cuidado al escribir los nombres ya que Arduino distingue entre mayúsculas y minúsculas por lo que no es lo mismo escribir un nombre todo en mayúsculas, todo en minúsculas o sólo con la primera letra en mayúscula. (Ej. MOTOR ≠ motor ≠ Motor).

Las funciones pueden devolvernos un valor o simplemente ejecutar una serie de comandos de programación, y pueden recibir algún parámetro o ninguno según lo que queramos realizar.

La sintaxis de cualquier función es:

```
tipo nombre (parámetros) {
        //programación de la función
}
```

Nota: Arduino nos permite insertar comentarios en el código para que podamos ubicarnos con mayor facilidad. Estos comentarios son ignorados por el programa y no ocupan espacio en la memoria así que pueden ser usados con generosidad.

Para insertar un comentario de una línea usamos "**//**". Si deseamos hacer un comentario más largo introducimos el mensaje entre "**/***" y "***/**".

· **El "tipo" indica el tipo o clase de dato que va a devolver la función.** Si nuestra función no devuelve ningún dato se escribe "**void**", con lo que indicamos que sólo ejecutará una rutina sin devolver ningún valor. En cambio, si deseamos que nuestra función devuelva un valor de tipo número entero deberíamos escribir "**int**".

· **El "nombre" es la palabra clave con la que identificamos una función.** A través del nombre podemos llamar a esa función en particular en alguna parte de nuestro código.

Entre paréntesis se escribirán, sólo si es necesario, los parámetros que se deben pasar a la función para que ésta se ejecute.

Para definir el principio y el final de un bloque de instrucciones, declaraciones y sentencias, se utiliza las llaves "**{}**". Una llave de apertura "**{**"siempre debe ir seguida de una llave de cierre "**}**" ya que las llaves sin cerrar provocan errores de compilación.

Es muy importante que todas las instrucciones de programación finalicen con punto y coma "**;**" para que el arduino entienda que allí termina la instrucción. Si te olvidas de poner fin a una línea con un punto y coma esto se traducirá en un error de compilación.

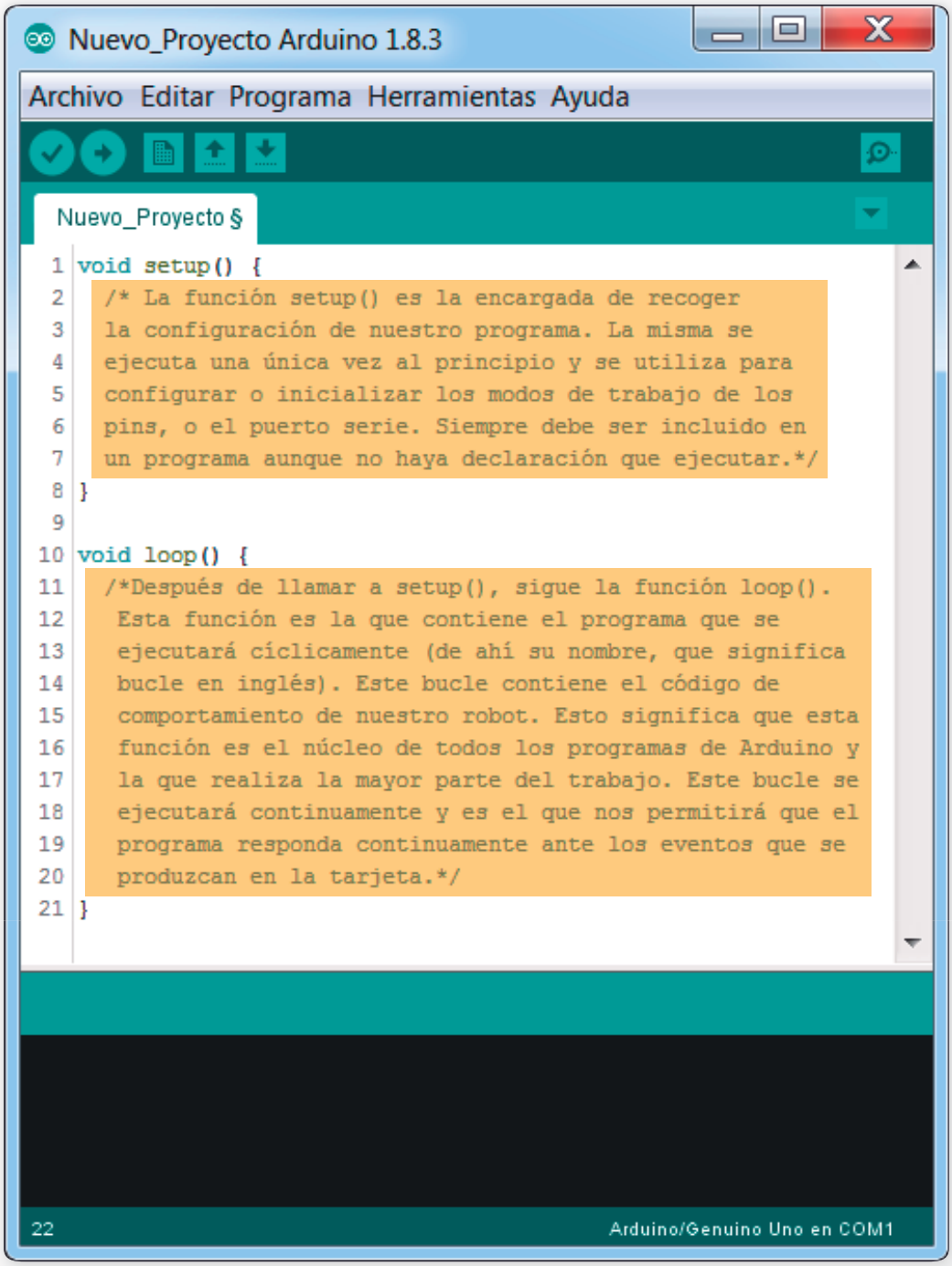
Por ejemplo:

```
void funcionDeEjemplo () {
        linea1;
        linea2;
        linea3;
}
```

Cuando en algún punto del programa llamemos a la función "funcionDeEjemplo()", ésta ejecutará el código que se encuentra entre llaves { } comenzando por la "linea1", siguiendo por la "linea2" y luego la "linea3".

Funciones estructurales

Ahora que ya estamos familiarizados con las funciones, veremos la estructura básica del lenguaje de programación de Arduino. El lenguaje de Arduino es bastante simple y se compone de dos partes o funciones fundamentales, las cuales deben estar siempre presentes en nuestro código para que nuestra programación funcione, tanto así que al crear un proyecto nuevo el programa nos brinda estas dos funciones por defecto: **void setup ()** y **void loop ()**.



La función pinMode

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida). Los pines que podemos declarar como entradas o salidas a través de esta función van del 0 al 13.

Su sintaxis correcta es:

```
pinMode(pin, INPUT ó OUTPUT);          // configura ese 'pin' como entrada o salida
```

Modalidad INPUT (entrada)

INPUT configura el modo de trabajo de pin digital, donde "pin" es una variable con el valor correspondiente al número del pin a utilizar y se elige el modo de trabajo. **Si configuras el pin como INPUT, sólo podrás usarlo para leer si hay 5V ó 0V en el pin.**

Su sintaxis correcta es:

```
pinMode(pin, INPUT);        // configura el 'pin' como entrada
```

Ejemplo:

```
pinMode(7, INPUT);          // configura el pin 7 como entrada
pinMode(btn, INPUT);        // configura el pin 'btn'  como entrada
```

Modalidad OUTPUT (salida)

Si declaras un pin como OUTPUT, sólo podrás usarlo para activarlo aplicando 5V en el pin, o desactivarlo aplicando 0V (cero voltios) **en el pin.** Cuando estos pines establecidos como salida están activados pueden proporcionar 40 mA (miliamperios) de corriente eléctrica a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie adecuada) pero no alcanza para alimentar cargas de mayor consumo, como motores, relés o selenoides.

Su sintaxis correcta es:

```
pinMode(pin, OUTPUT);       // configura el 'pin' como salida
```

Ejemplo:

```
pinMode(7, OUTPUT);         // configura el pin 7 como salida
pinMode(led, OUTPUT);       // configura el pin 'led'  como salida
```

Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados llamados constantes. Estas constantes se utilizan para hacer los programas más fáciles de leer y están clasificadas en grupos.

• **TRUE/FALSE** (cierto/falso)
Es una constante booleana que distingue un valor verdadero de uno falso.
FALSE está asociado a LOW o 0 mientras que TRUE se asocia a HIGH o cualquier valor distinto de cero.

• **HIGH/LOW** (alto/bajo)
Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital de los pin. ALTO se define en la lógica como nivel 1, ON, o 5 voltios, mientras que BAJO según la lógica es nivel 0, OFF, ó 0 voltios.

• **INPUT/OUTPUT** (entrada/salida)
Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

Variables

A diferencia de las constantes, una variable debe ser declarada y, opcionalmente, asignarle un valor. Como su nombre lo indica, el valor de una variable puede cambiar continuamente; esto nos sirve para nombrar y almacenar un valor numérico para su uso posterior en el programa. **En función del lugar en donde se lleve a cabo la definición de la variable esto determinará en que partes del programa se podrá hacer uso de ella.**

Una vez que una variable ha sido asignada, o re-asignada, podemos probar su valor para ver si cumple ciertas condiciones o podemos utilizar directamente su valor. Sin embargo, **todas las variables tienen que declararse antes de que puedan ser utilizadas**.

1. **Para declarar una variable se comienza por definir su tipo**, es decir, el tipo de dato con el que trabajaremos:

• **Byte** Trabaja con números enteros cortos (8 bits) sin decimales. Tienen un rango entre 0 y 255.

• **Int** Números enteros. Comprende valores enteros de 16 bits sin decimales en un rango desde 32,767 hasta -32,768.

• **Long** Trabaja con números enteros más largos (de 32 bits) sin decimales. Su rango es desde -2147483648 hasta 2147483647.

• **Float** Flotante significa que el valor numérico contiene decimales. Estos valores tienen más definición que los de 32 bits (long) pero hacen que las comparaciones no sean tan exactas y demoran más en procesarse por lo que se recomienda evitar usarlos si es posible.

Una vez definido el tipo de variable el siguiente paso obligatorio es darle un nombre. Para que sea más sencillo seguir el programa se recomienda que cada variable tenga un nombre descriptivo que facilite leer el código y entender qué hace.

2 Por ejemplo "contactoSensor", "pulsador", "variableEntrada", etc.

Una variable puede llevar cualquier nombre o palabra que no sea una de las palabras reservadas en el entorno de Arduino. Si lo necesitas, hay un .pdf dedicado a ellas.

Utilización de una variable

En función del lugar de declaración de la variable se determinará el ámbito de su aplicación, o, lo que es lo mismo, la capacidad de ciertas partes de un programa para hacer uso de ella.

- **Variable global**
  Es aquella que puede ser vista y utilizada por cualquier función e instrucción de un programa. Esta variable se declara al comienzo del programa, afuera del setup().

- **Variable local**
  Es aquella que se define dentro de una función o como parte de un bucle. Sólo es visible y sólo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que sólo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

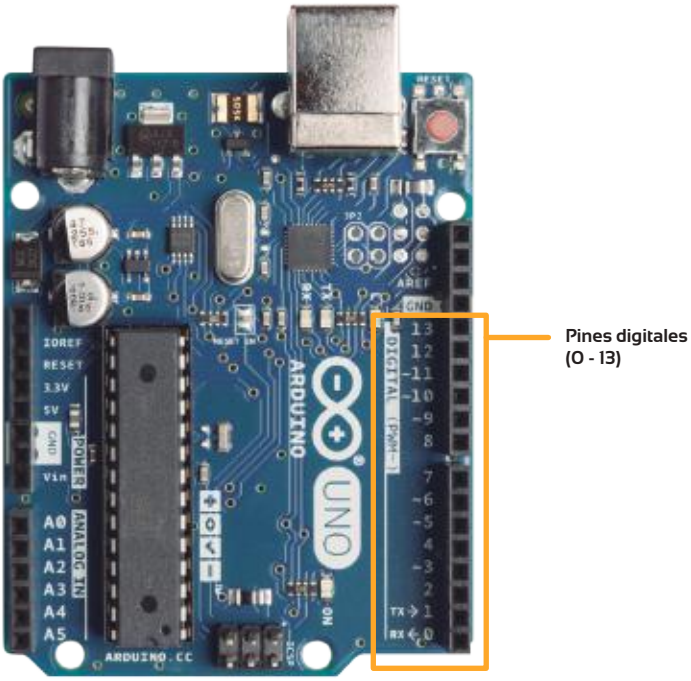
El siguiente ejemplo muestra cómo declarar estos diferentes tipos de variable y la visibilidad de cada variable:

```
int valor; // 'valor' es visible para cualquier función

void setup() {
        // no es necesario configurar
}

void loop() {
        for (int i=0; i<20;) // 'i' sólo es visible dentro del bucle for
        {
        i++;
        }
        float f; // 'f' es visible sólo dentro del bucle loop
}
```

Nota:
Para comprender mejor este ejemplo necesitarás haber visto los apartados sobre bucles y estructuras de control.

Otras funciones asociadas a los pin

Los pin, además de poder configurarse como entradas o salidas, pueden tener un funcionamiento analógico o digital según dónde los conectemos en la placa de Arduino.



Funciones digitales

Los pin digitales se conectan en los puertos de la derecha designados como "digital", numerados del 0 al 13.

• **digitalRead** (significa "lectura digital"): sirve para leer el valor de un pin definido como digital, dando un resultado **HIGH** (alto, 5V) o **LOW** (bajo, 0V). El pin se puede especificar de dos maneras, ya sea como una variable definida previamente o como una constante (es decir, con un valor entero entre 0 y 13).

Su sintaxis correcta es: **digitalRead(pin)**

Ejemplo:

```
valor = digitalRead(Pin); // hace que 'valor' sea igual al estado leído en el pin 'Pin'

valor = digitalRead(12); // hace que 'valor' sea igual al estado leído en el pin 12
```

• **digitalWrite** (significa "escritura digital"): esta función envía al pin definido previamente como **OUTPUT** el valor **HIGH** (alto, 1) o **LOW** (bajo, 0) según lo definamos en la salida. Al igual que en digitalRead, el pin puede especificarse como una variable o una constante.

Su sintaxis correcta es: **digitalWrite(pin, valor)**

Ejemplo:

```
digitalWrite(Led, HIGH); // escribe en el pin 'Led' un valor HIGH (1)
```

```
digitalWrite(8, LOW); // escribe en el pin 8 un valor LOW (0)
```

Antes de pasar a las siguientes funciones, veamos un ejemplo integrador donde se aplican los conceptos vistos hasta ahora. La siguiente programación está pensada para leer un pulsador conectado a una entrada digital y escribir esa lectura en el ´pin´ de salida LED:

```
//definimos las variables globales
int led = 13; // asigna a LED el valor 13
int boton = 7; // asigna a botón el valor 7
int valor = 0; // define el valor y le asigna el valor 0

void setup(){
        pinMode(led, OUTPUT); // configura el led (pin13) como salida
        pinMode(boton, INPUT); // configura botón (pin7) como entrada
}

void loop(){
        valor = digitalRead(boton); //lee el estado de la entrada botón
        digitalWrite(led, valor); // envía a la salida ´led´ el valor leído
}
```

Bucles

A pesar que nuestra función loop() trabaja repitiendo la programación, muchas veces tendremos que programar repeticiones, ya sea una cantidad determinada de veces o hasta que se cumpla una condición. Para esto contamos con las estructuras de repetición while y for, como vemos en este ejemplo (EL DE ARRIBA). Existen cientos de aplicaciones con estas dos estructuras repetitivas, son muy necesarias para lograr el comportamiento que queremos en nuestro robot.

while

La estructura **while**, que significa "**mientras**" en inglés, nos **permite repetir las líneas de programación hasta que se deje de cumplir la condición especificada en su parámetro.** Esto significa que mientras se cumpla la condición detallada en el parámetro de la función esta estructura va a repetir el código que le ingresemos entre las llaves.

Su sintaxis correcta es:

```
while(CONDICION) {
    //Código que quiero repetir
    }
```

for

La estructura while es muy útil para todo tipo de programaciones, pero en **ocasiones necesitamos que alguna acción se ejecute sólo una determinada cantidad de veces**. La estructura **for** nos ayuda a lograr eso con mucha facilidad ya que va a repetirse la cantidad de veces que le especifiquemos y luego saldrá de la función.

Su sintaxis correcta es:

```
for (inicialización; condición; expresión) {
    //Código que quiero repetir
    }
```

La inicialización de una variable local se produce una sola vez y la condición se testea cada vez que se termina la ejecución de las instrucciones dentro del bucle. Si la condición sigue cumpliéndose, las instrucciones del bucle se vuelven a ejecutar. Cuando la condición no se cumple (o deja de cumplirse), el bucle termina.

Ejemplo

```
for (int i = 0 ; i < CANTIDAD ; i++ ) {
    //Código que quiero repetir
    }
```

En este ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a la CANTIDAD que pongamos. Si es cierto, i se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves. Cuando la condición deja de cumplirse el robot sale del bucle y continúa con el programa.

Estructuras de control

A continuación veremos las estructuras de control que podemos usar en nuestros códigos.

Recordemos que en un programa las instrucciones se ejecutan secuencialmente, comenzando por las de arriba y ejecutando una por una hasta llegar abajo. **Las estructuras de control nos permitirán alterar ese flujo normal, pudiendo armar estructuras lógicas que alteren el comportamiento de nuestro programa y, por lo tanto, de nuestro robot.**

Las estructuras de control requieren mucha práctica y **siempre trabajan con comparaciones numéricas y operaciones lógicas para determinar qué camino deben tomar.** Tanto las operaciones lógicas como las comparaciones solo pueden arrojarnos dos posibles valores, verdadero (**true**) o falso (**false**).

Comparaciones numéricas

Son expresiones lógicas que nos permiten establecer la diferencia o igualdad entre dos valores numéricos. Tenemos seis operadores diferentes:

		Ejemplos	
x **==** y	x **es igual a** y	1 == 1	Resultado: true
x **!=** y	x **no es igual a** y	1 != 1	Resultado: false
x **<** y	x **es menor que** y	1 < 1	Resultado: false
x **>** y	x **es mayor que** y	2 > 1	Resultado: true
x **<=** y	x **es menor o igual que** y	6 <= 4	Resultado: false
x **>=** y	x **es mayor o igual que** y	3 >= 3	Resultado: true

¡IMPORTANTE!
Debes tener en cuenta que poner un doble igual (**==**) no es lo mismo que colocar uno solo (**=**). Recuerda que == es una comparación, donde el código comprueba si un valor es igual al otro, mientras que = simplemente describe una equivalencia. **Esto significa que si te equivocas y escribes dos iguales en vez de uno o uno en lugar de dos, el programa no te advertirá del error ya que ambas son instrucciones válidas aunque muy diferentes**.

Lo veamos en un ejemplo:

x == y Significa que comparará los valores de ambos términos. Resultado: true o false.
x = y Significa que x es igual a y, por lo que la variable x adquirirá el valor de y.

Operaciones lógicas

Nos permiten comparar dos valores o expresiones devolviendo un único resultado lógico. Para estas estructuras podemos usar tres operadores lógicos:

- AND (**&&**) Devuelve verdadero sólo si todas expresiones son verdadero.
- OR (**||**) Devuelve verdadero si cualquiera de sus expresiones es verdadera.
- NOT (**!**) Cambia el valor de una expresión.

Ejemplos

true && true	Resultado: true	1 == 1 && false	Resultado: false
1 != 2 || 1 == 5	Resultado: true	!true	Resultado: false
!true && true	Resultado: false		

Operaciones aritméticas

Dentro de las declaraciones de nuestros programas también podemos realizar operaciones aritméticas. **Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división.** Probablemente ya estés familiarizado con ellos, pero por si acaso vamos a repasar cómo se escriben.

Ejemplos

y = y **+** 3;	y es igual a y **más** 3
x = x **-** 7;	x es igual a x **menos** 7
i = j ***** 6;	i es igual a j **multiplicado** por 6
r = r **/** 5;	r es igual a r **dividido** 5

Asignaciones compuestas

Combinar operaciones aritméticas con variables asignadas nos permite crear asignaciones compuestas.

Estas asignaciones compuestas son comúnmente utilizadas en los bucles y pueden ser:

x **++**	igual que x = x + 1, o incrementar x en + 1
x **--**	igual que x = x - 1, o decrementar x en -1
x **+=** y	igual que x = x + y, o incrementar x en +y
x **-=** y	igual que x = x - y, o decrementar x en -y
x ***=** y	igual que x = x * y, o multiplicar x por y
x **/=** y	igual que x = x / y, o dividir x por y

Ejemplos

a ++
Equivale a escribir **a = a + 1**
Cada vez que se ejecute esta asignación, el valor de **a** crecerá en **1**.

a --
Equivale a escribir **a = a - 1**
Cada vez que se ejecute esta asignación, el valor de **a** disminuirá en **1**.

c += 4
Equivale a escribir **c = c + 4**
Tomando como referencia el valor de la variable **c**, esta asignación le sumará **4** a ese valor y el resultado de esta suma se convertirá en el nuevo valor de **c**.
Nota: se diferencia de un **c++** porque nos permite poner un valor de incremento diferente a 1.

g -= 2
Equivale a escribir **g = g - 2**
Esta asignación usará el valor de la variable **g** para restarle **2** y el resultado de esa resta se convertirá en el nuevo valor de **g**.
Nota: se diferencia de un **g--** porque nos permite poner un valor de disminución diferente a 1.

x * = 3
Equivale a escribir **x = x * 3**
Esto significa que la variable **x** multiplicará por **3** su valor y el resultado se convertirá en el nuevo valor de **x**.

j / = 5
Equivale a escribir **j = j / 5**
Tomando como referencia el valor de la variable **j**, esta asignación dividirá **j** en **5** y el resultado de esa división será el nuevo valor de **j**.

Errores comunes en la programación

Es muy común cometer errores de tipeo, olvidos o distracciones al escribir un programa. En la programación estos errores se pueden clasificar en dos tipos:

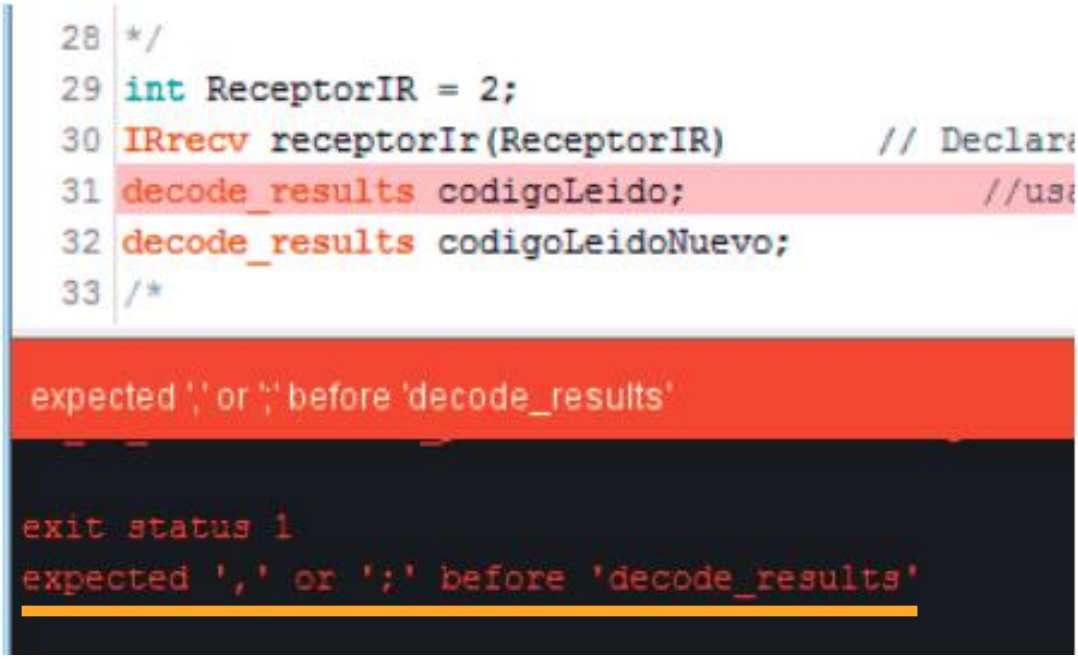
1. el **error de sintaxis**, cuando escribimos algo mal, o nos olvidamos de escribir algo, y
2. el **error lógico**, que sucede cuando el programa no hace lo que debería porque estamos haciendo los pasos equivocados.

Afortunadamente, el IDE de arduino puede darse cuenta de que cometimos un error de sintaxis, decirnos cuál es, y orientarnos para encontrarlo dentro del código. Cada vez que nos salte un error en la compilación del código, lo más importante es tratar de entender qué nos está queriendo decir el mensaje de error para luego intentar solucionarlo.

A continuación veremos algunos de los errores más comunes al programar y la forma de identificarlos:

Faltó un ";"

Recuerda que todo comando debe terminar con **";"**. Cuando falta el punto y coma, recibiremos el error que vemos en la siguiente ventana de mensajes, así como veremos una línea resaltada en la pantalla.

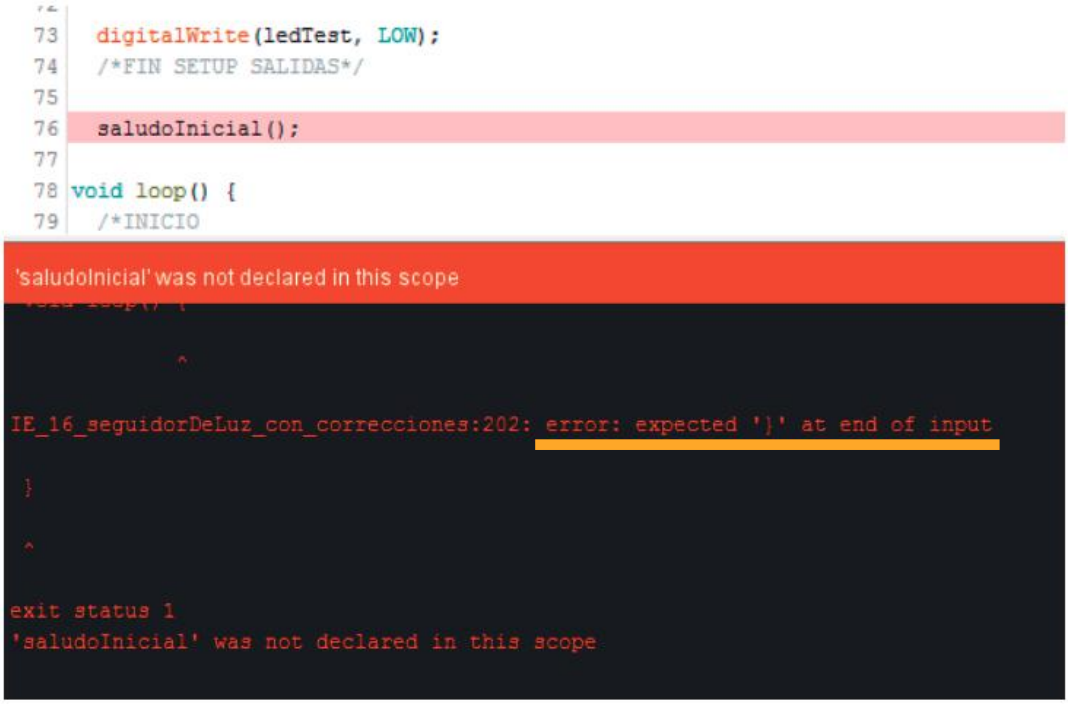


Por lo general la línea resaltada es la que contiene el error, pero si prestas atención, vas a ver que en este ejemplo el IDE resalta la línea siguiente. Por lo tanto, cuando recibas este error, fíjate en la línea anterior a la resaltada y agrega el **";"** al final.

Faltó una llave "{" o "}"

Este error es un poco más difícil de identificar por su mensaje, y lamentablemente el IDE tampoco nos acerca mucho a la línea donde está el problema.

Si nos olvidamos de cerrar una llave el mensaje será similar a:

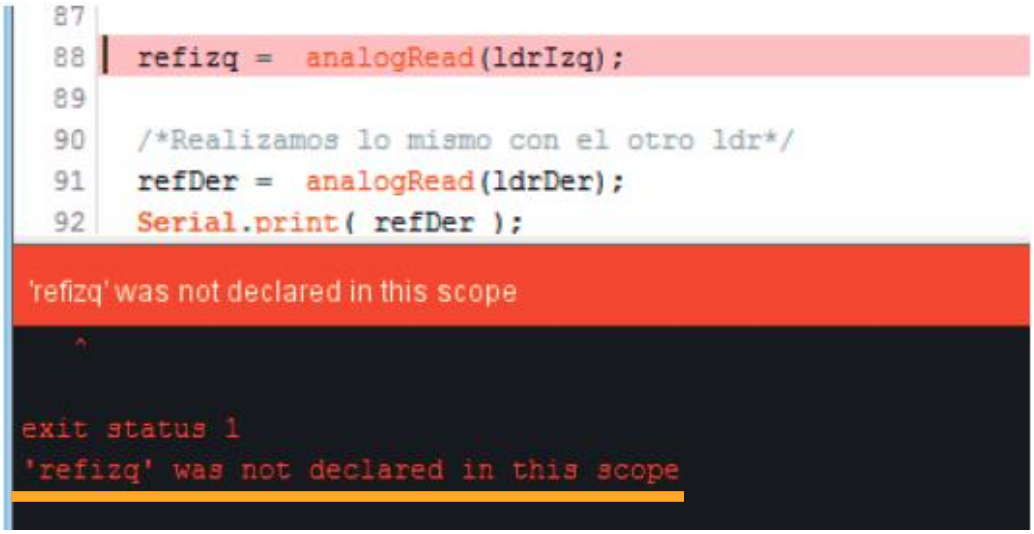


Para solucionarlo tendremos que repasar nuestro código hasta encontrar qué llave se encuentra sin su compañera para que la declaración que contienen resulte válida en el programa y arduino pueda llevarla a cabo.

Intercambiar mayúsculas con minúsculas

Otro error común de sintaxis es olvidar que el lenguaje de arduino es ***"case sensitive"***. Esto significa que el código es sensible a la diferencia entre un carácter escrito en mayúsculas y otro en minúsculas.

Por ejemplo, si en la declaración escribimos el nombre de una variable de una forma en particular, "valorLuz" no será lo mismo que "ValorLuz" ni que "valorluz" ni que "VALOR LUZ". **Si nuestra variable está declarada de determinada manera, al utilizarla en el código debe estar**



Afortunadamente, en este caso el IDE es claro en el mensaje (***"La variable no está declarada en este ámbito"***) y nos lleva directo hacia donde usamos la variable en cuestión, por lo que sólo nos resta buscar la declaración de la variable (si es que no nos olvidamos de hacerla) y ver bien cómo la llamamos para escribir su nombre exactamente igual en el lugar del error.

Recomendaciones

Como siempre que estamos aprendiendo algo, todo es cuestión de práctica y paciencia, incluso el entender los mensajes de error del IDE. Nuestra sugerencia es que cada vez que encuentres un error que no entiendas de qué se trata, te animes a googlear para tratar de encontrar la solución. De esta forma no sólo vas a poder solucionar tus problemas de código si no que seguramente también vas a profundizar tus conocimientos.

Si después de revisar todo aún no encontrás la solución, recordá que podés contactarnos a través de la página y hacernos tu consulta.

ACTIVIDADES Y EJERCITACIÓN

Cap. 4: Actividades complementarias

En base a los contenidos vistos en el **"Capítulo 4: Luces para mi robot"** realiza las siguientes actividades complementarias.

Actividad 1

Responde a las siguientes preguntas.

a. ¿Para qué nos sirven las funciones a la hora de programar?

b. ¿Para qué sirven los parámetros?

c. ¿Cuáles son las funciones estructurales?

d. ¿Con qué función definimos si un pin es de entrada o salida?

e. ¿Por qué es conveniente declarar una variable o constante asociada a un pin al inicio del programa si este pin no cambia nunca?

f. ¿Para qué sirve la estructura FOR?.

Actividad 2

Teniendo en cuenta las conexiones internas de la protoboard, modifica el circuito para que los dos leds blancos trabajen con un solo pin de arduino, y los dos leds rojos con otro, usando así sólo dos pines de la placa en lugar de cuatro. En base a estos cambios, adecuar la programación según corresponda.

Cap. 4: Respuestas

Actividad 1

Responde a las siguientes preguntas.

a. ¿Para qué nos sirven las funciones a la hora de programar?
Las funciones tienen dos utilidades principales:
- La primera, es **ahorrar líneas de código**. Cuando tenemos un conjunto de instrucciones que se repite más de una vez en nuestro programa es conveniente separar esas líneas en una función y después llamarla desde nuestro programa principal.

- La segunda utilidad es la de permitirnos **clarificar el código al poder ponerle un "nombre"** a un grupo de instrucciones y usar solamente ese nombre en nuestro programa principal.

Por ejemplo, para hacer parpadear varios leds necesitamos tantas instrucciones digitalWrite en LOW como leds tengamos, un delay, y después otras tantas digitalWrite en HIGH, más otro delay, dentro de un FOR. Todo esto en medio de nuestro programa puede hacernos perder el foco de lo que realmente queremos lograr, por lo que es más claro, *"sacar del medio"* todas esas instrucciones y dejar solo una llamada a una función "parpadeo".

Además, las funciones también tienen la **capacidad de devolver un valor al programa principal**, por lo que podríamos, por ejemplo, generar una función que se llame "Promedio" y le pasemos una serie de valores por parámetro. La función realizaría el cálculo del valor, y nos devolvería la media.

b. ¿Para qué sirven los parámetros?
Los parámetros nos sirven para dar más flexibilidad a nuestras funciones y hacerlas más reutilizables. Por ejemplo, si quisiéramos hacer funciones para poder encender o apagar un grupo de leds, sin parámetros, deberíamos crear dos funciones, (una para encenderlos, y otra para apagarlos) y estas funciones serían prácticamente iguales. En cambio, podemos crear una sola función y usar un parámetro para pasar el valor HIGH o LOW, dependiendo de si necesitamos encender o apagar los leds.

c. ¿Cuáles son las funciones estructurales?
Las funciones estructurales son Setup y Loop. Estas no pueden faltar nunca en nuestro programa ya que son indispensables para su funcionamiento.

d. ¿Con qué función definimos si un pin es de entrada o salida?
Definimos si un pin es de entrada o salida con la función pinMode()

e. ¿Por qué es conveniente declarar una variable o constante asociada a un pin al inicio del programa si este pin no cambia nunca?
Es conveniente declarar la variable o constante al inicio porque si necesitamos cambiar ese pin por cualquier razón, sólo debemos cambiar el número de pin asociado a la variable y listo, sin necesidad de rastrear por todo el programa dónde utilizamos ese pin para modificarlo las veces que sea necesario.

f. ¿Para qué sirve la estructura FOR?
La estructura FOR nos sirve para repetir una o más instrucciones un determinado número de veces. Una vez que se alcanza ese número de repeticiones, el bucle se corta y el programa continúa.

Actividad 2

Teniendo en cuenta las conexiones internas de la protoboard, modifica el circuito para que los dos leds blancos trabajen con un solo pin de arduino, y los dos leds rojos con otro, usando así sólo dos pines de la placa en lugar de cuatro. En base a estos cambios, adecuar la programación según corresponda.

Esta es la programación modificada. La modificación consistió en eliminar todas las referencias a leds derechos porque los conectamos en serie con los izquierdos.

```
int ledBlancoIzquierda = 4;
int ledRojoIzquierda = 8;

void setup() {
  pinMode(ledBlancoIzquierda, OUTPUT);
  pinMode(ledRojoIzquierda, OUTPUT);

  digitalWrite(ledBlancoIzquierda, LOW);
  digitalWrite(ledRojoIzquierda, LOW);
 }

void loop() {
  digitalWrite(ledBlancoIzquierda, HIGH);
  digitalWrite(ledRojoIzquierda, HIGH);

  delay(1000);

  parpadeo(ledRojoIzquierda, 5, 1000);
  parpadeo(ledBlancoIzquierda, 10, 750);

  digitalWrite(ledBlancoIzquierda, LOW);
  digitalWrite(ledRojoIzquierda, LOW);
  delay(2000);
}

void parpadeo(int ledParpadea, int veces, int tiempo) {
  for (int j=0; j <= veces ; j++) {
    digitalWrite(ledParpadea, LOW);
    delay(tiempo);
    digitalWrite(ledParpadea, HIGH);
    delay(tiempo);
  }
}
```