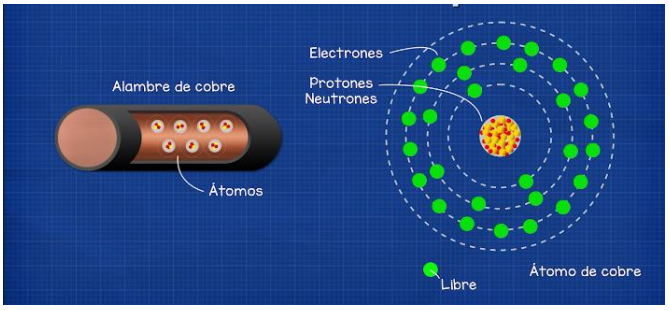
CAPACITACION 4.0

Índice del documento

Electricidad y electrónica básica

La corriente eléctrica

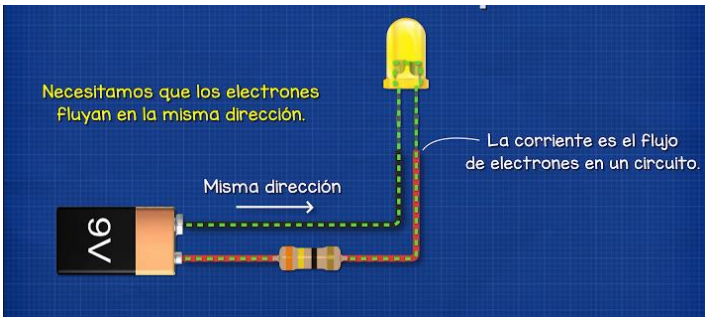
En este apartado vamos a analizar la corriente eléctrica para entender los diferentes tipos los símbolos utilizados para representarlas, cómo medir la corriente y como los dispositivos de seguridad se utilizan para protegernos a nosotros y a nuestros circuitos eléctricos.



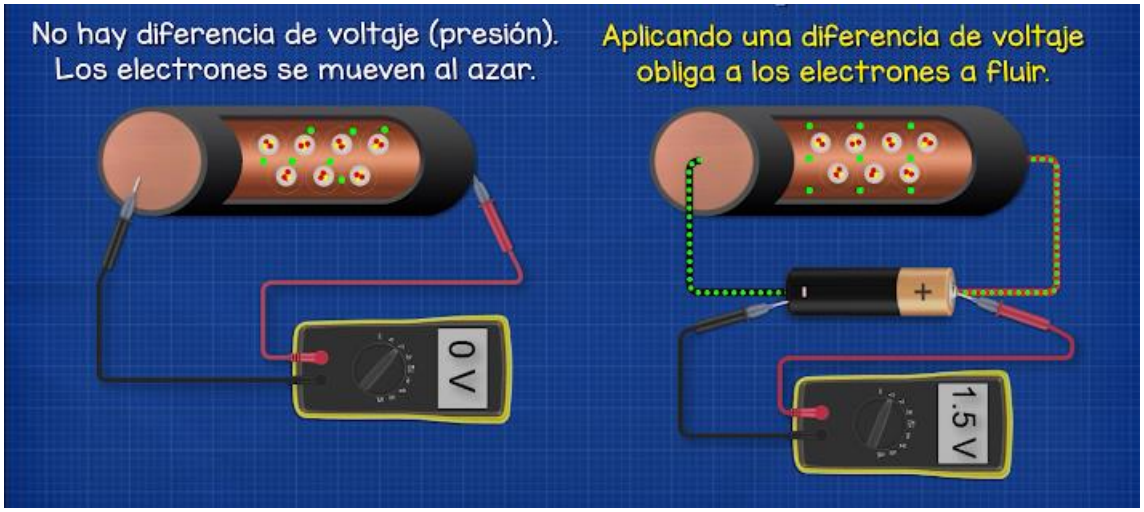
La corriente es el flujo de electrones en un circuito. Para utilizar la electricidad necesitamos que los electrones fluyan en la misma dirección alrededor de un circuito, normalmente utilizamos cables de cobre para formar el circuito porque el cobre es un muy buen conductor eléctrico, lo que significa que los átomos que componen el cobre tienen un electrón débilmente ligado en esa capa exterior o de valencia que es libre de moverse dentro del metal, este electrón libre se mueve muy fácilmente por eso el cobre es tan utilizado. Es tan fácil de mover que se moverán naturalmente a otros átomos de cobre por sí misma, pero esto ocurre al azar en todas las direcciones, lo que no nos sirve.

Envolvemos los cables de cobre en plástico porque el plástico es un aislante, lo que significa que no permite que los electrones libres pasen a través de él, eso proporciona una barrera y mantiene a la electricidad dentro de los cables y lejos de nosotros.

Para que podamos usar la electricidad para hacer funcionar nuestros dispositivos necesitamos que muchos electrones fluyen en la misma dirección a lo largo de un circuito, podemos entonces colocar cosas como lámparas en el camino de estos electrones para que tengan que fluir a través de él y generar luz y calor en el proceso.



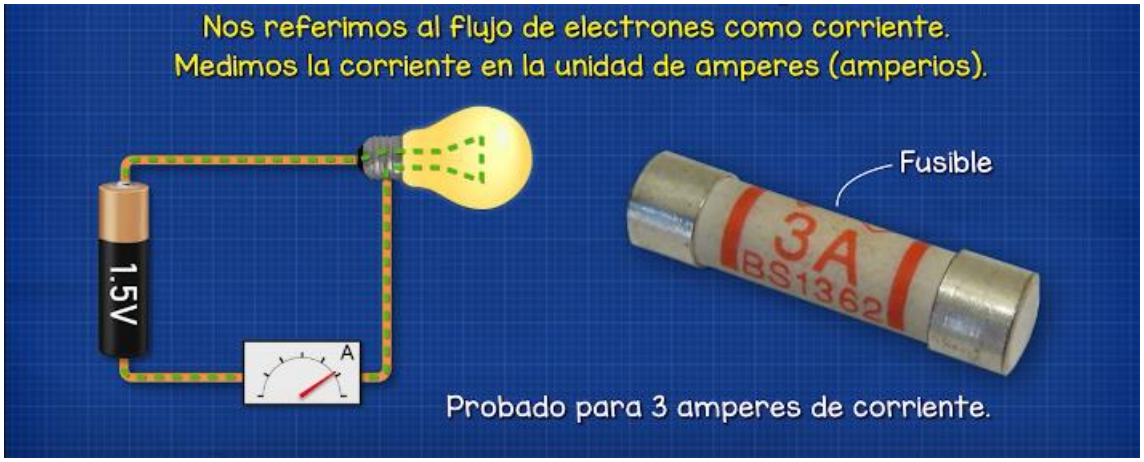
Para hacer esto necesitamos forzar a los electrones a moverse y podemos hacerlo aplicando un voltaje, el **voltaje** es la fuerza de empuje y se mide en **voltios**. Es como la presión en una tubería de agua, cuanta más presión tenemos más agua puede fluir, cuanto más voltaje tenemos más electrones pueden fluir. Podemos medir la presión sin que fluya el agua y podemos medir el voltaje sin que fluya la corriente, pero no podemos medir cuánta agua está fluyendo si no hay agua fluyendo y no podemos medir la corriente eléctrica sino hay electrones fluyendo.



Si tomamos un cable de cobre no hay **diferencia de voltaje** entre los dos extremos así que los electrones libres se mueven al azar, este movimiento aleatorio no se considera una corriente, pero si tomamos una batería de digamos 1.5 volts y conectamos el cable a través de los dos terminales entonces hay ahora una diferencia de 1.5 volts a través del cable y esta diferencia va a forzar a los electrones a fluir en la misma dirección.

Así que necesitamos que muchos electrones fluyan a lo largo de un circuito y a través de nuestras lámparas para que brillen, sin embargo los cables y la lámpara solo pueden manejar una cierta cantidad de electrones que pasan a través de ellos, al igual que una tubería está diseñada para manejar una cierta cantidad de agua que pasa a través de ella o una cierta presión, si se excediera la tubería explotaría, de la misma manera sí hay demasiados electrones pasando por el cable o la lámpara entonces estallará o se quemará.

Nos referimos al flujo de electrones como corriente y lo medimos en unidades de **amperes** aunque también se conocen como amperios, esto se representa con una A mayúscula, por ejemplo este fusible tiene un tres y una A mayúscula, lo que significa que está diseñado para tres amperes de corriente. Veremos cómo funcionan los posibles un poco más adelante en este capítulo.



Corriente convencional vs flujo de electrones

Algo que te va a causar gran confusión cuando aprendas sobre la electricidad es la diferencia entre la corriente convencional y el flujo de electrones, ambas son teorías de cómo funciona la electricidad.

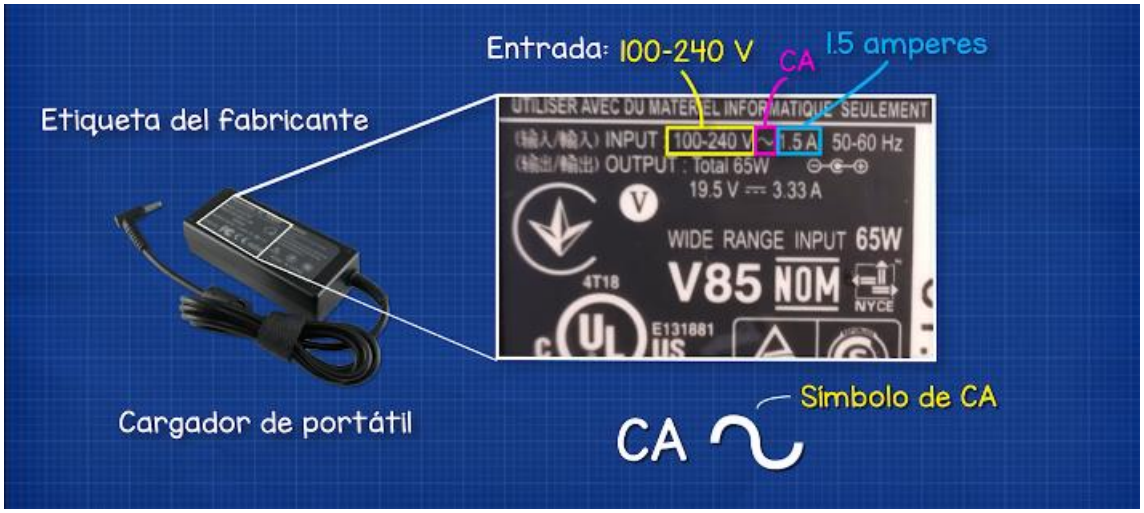
cuando benjamín franklin experimentó por primera vez con la electricidad tuvo la idea de que algo debía fluir dentro de los materiales, para ello le dieron un tubo de vidrio y cuando éste se frotaba con un paño parecía acumular este extraño e invisible fluido porque cuando alguien más tocaba el tubo recibió una pequeña descarga. Ahora lo conocemos como electricidad estática pero en ese momento benjamín franklin asumió que el tubo estaba acumulando una cantidad excesiva de este fluido invisible, así que consideró que esto era positivo y que la persona que lo tocaba debía tener menos de este fluido por lo que se consideraron negativos.

Así que dijo bueno, supongo que este extremo es **positivo** y este extremo es **negativo** y la electricidad fluye de positivo a negativo y esto tiene sentido porque el agua es un fluido y fluye de un nivel alto a un nivel bajo.

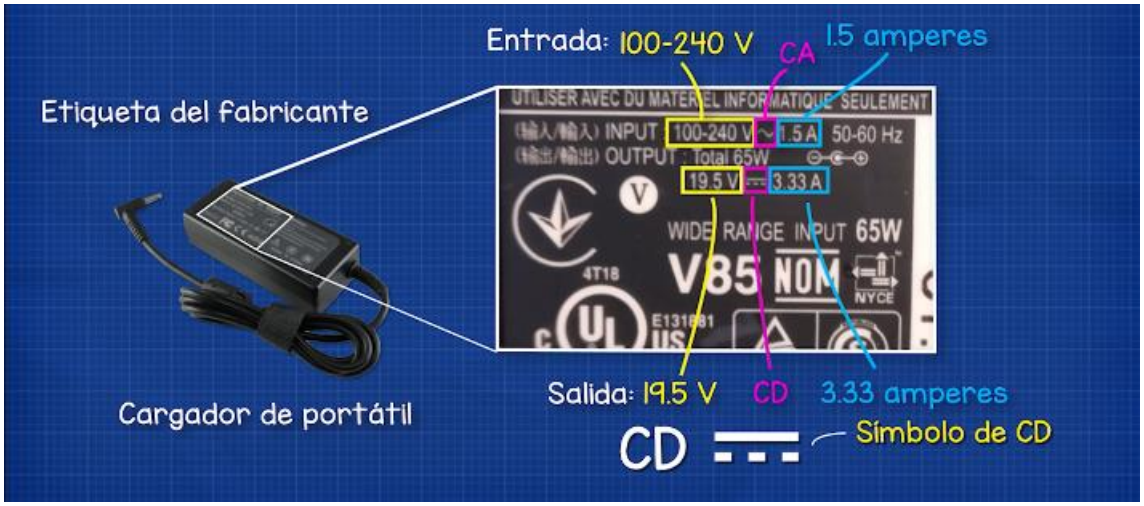
pronto los fabricantes comenzaron a producir baterías basadas en su trabajo y también dijeron bueno este extremo es positivo y este extremo es negativo y todavía usamos esta convención de nombres hasta el día de hoy, esto se conoció como **corriente convencional, porque es la teoría convencional de cómo fluye la electricidad** sin embargo a medida que la ciencia evoluciona y los experimentos se hicieron más precisos alguien llamado Joseph Thompson descubrió que esta cosa invisible que se movía dentro del cable era una partícula y llamó a esta partícula un **electrón**. También descubrió que estos electrones en realidad fluían en dirección opuesta de la negativa a la positiva. Benjamín franklin no se dio cuenta de que la tela de seda estaba removiendo electrones del vidrio, así que en realidad estaban fluyendo desde la persona y hacia el tubo de vidrio. La teoría de Joseph Thompson se conoció como **flujo de electrones,** porque es el flujo de electrones en sí, **y lo que realmente está ocurriendo es que estos electrones están fluyendo en un circuito de la terminal negativa a la positiva y no de la positiva a la negativa,** sin embargo no importa realmente en lo que se mueve dentro del cable o en qué dirección porque las fórmulas de ingeniería eléctrica que utilizamos no lo tienen en cuenta.

Hay que recordar que siempre que hablamos de electricidad o cuando diseñamos un circuito o incluso miramos un dibujo de un circuito eléctrico siempre nos salimos de la norma y asumimos que se está produciendo una corriente convencional pero los ingenieros y científicos saben que los electrones están fluyendo en realidad en sentido contrario.

CA o CC (CD)



si miras en los enchufes de tus aparatos eléctricos deberías encontrar etiquetas de los fabricantes que te digan para qué está diseñado el producto, por ejemplo este cargador de portátil nos dice que para que el dispositivo funcione necesita una entrada de entre 100 y 240 volts y 15 amperes de CA o corriente alterna lo que se representa aquí con este símbolo, el cargador entonces lo convertirá para dar una salida de alrededor de 19.5 voltios y 3.33 amperes de CD o corriente directa, lo cual está representado por este.



En la corriente alterna este tipo los electrones no fluyen en un ciclo continua, alternan entre moverse hacia adelante y hacia atrás como la marea del mar sus dispositivos eléctricos como los ordenadores portátiles y los teléfonos móviles utilizarán electricidad de corriente directa, en este tipo los electrones fluyen en una sola dirección directamente de un terminal a otro, puedes pensar en esto como el flujo de agua de un río.

En la mayoría de los casos transportamos la electricidad de una central eléctrica a los pueblos y ciudades usando electricidad de CA porque es fácil aumentar y disminuir el voltaje usando transformadores y también es muy eficiente transportar electricidad a largas distancias usando este método.

Utilizamos principalmente la corriente directa o CD para los circuitos de pequeños dispositivos electrónicos como ordenadores portátiles, teléfonos móviles y televisores. Eso es porque la CD es más fácil de controlar y permite que los circuitos sean más pequeños y compactos.

Podemos convertir la CA en CD usando un dispositivo conocido como rectificador, esto es extremadamente común en la electrónica.

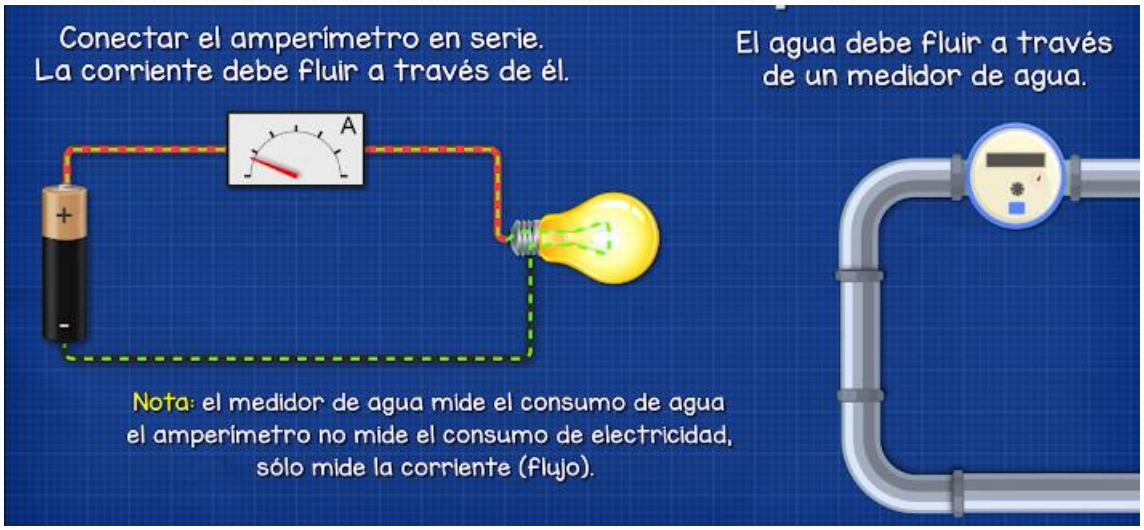
Corriente

La gente a menudo se refiere a un río o a la marea del mar como si tuviera una corriente fuerte, es muy similar a la electricidad. Se dice que un río con mucha agua que fluye rápidamente tiene una corriente fuerte, lo mismo que la electricidad. Un cable con muchos electrones fluyendo también tiene una corriente alta. La tubería es capaz de manejar una cierta cantidad de agua que fluye a través de él, pero si entra más agua de la que puede manejar el río se desbordara, lo mismo ocurre con la electricidad, el cable se reventara y se quemará, por lo tanto los fabricantes deben ser capaces de probar los cables y lámparas para saber cuenta corriente pueden manejar. También queremos ser capaces de ver cuánta corriente fluye a través de nuestros circuitos además de poder calcularla, podemos medirla con un amperímetro y medimos el **flujo de corriente** en unidades de **amperes** pero también se oye a la gente decir amperios.

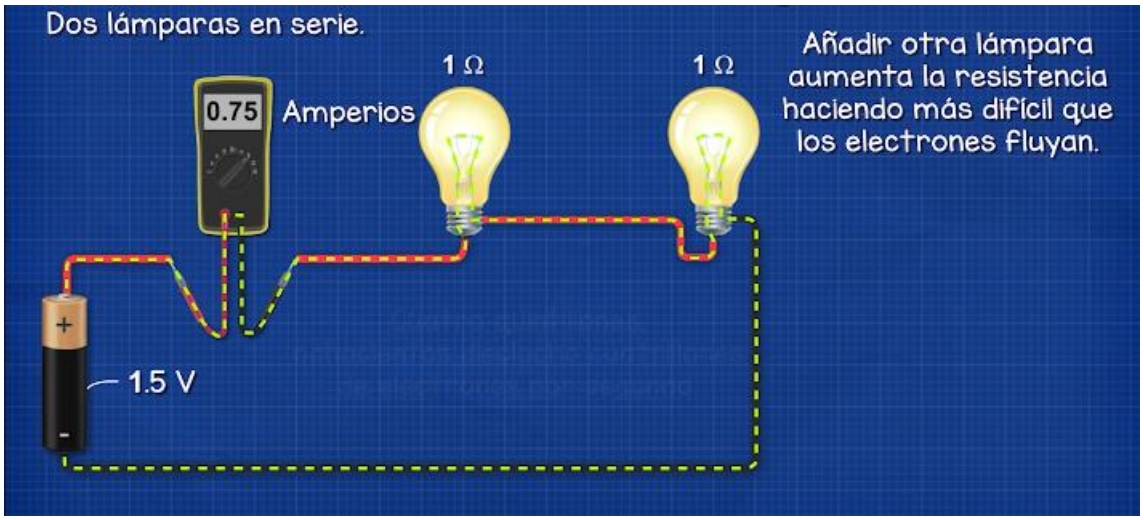
Entonces que es una amperio? un amperio es igual a un colombio por segundo y un colombio es igual a aproximadamente 6 quintillones 242 cuatrillones de electrones por segundo.

El brillo de la lámpara variará con el voltaje, a medida que disminuimos el voltaje hay menos presión que empujan los electrones así que menos electrones fluyan, a medida que aumentamos el voltaje fluyen más electrones y la lámpara brilla más, pero no olvides que a un cierto voltaje y corriente una lámpara se quemará.

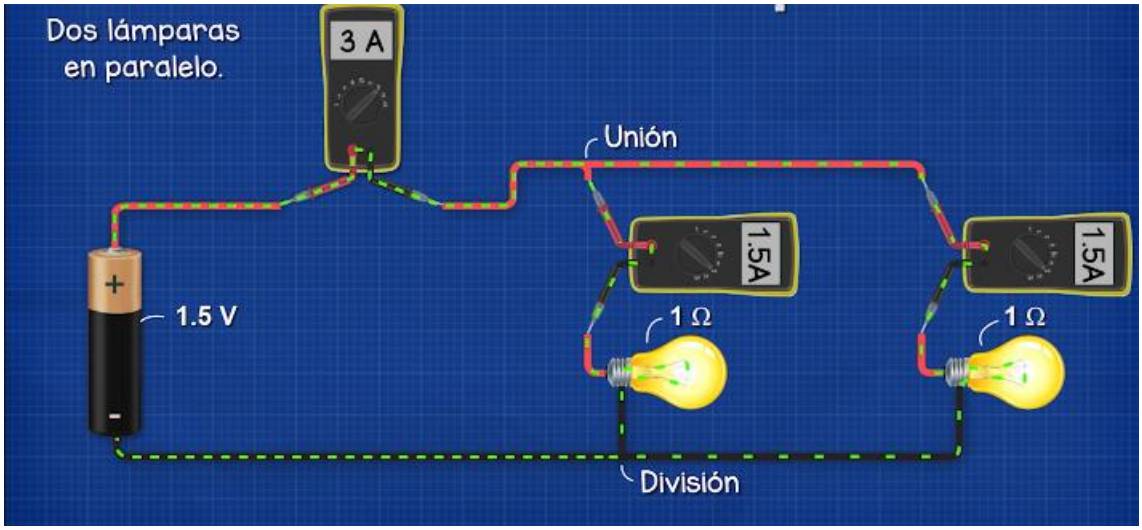
Para medir la corriente en un circuito necesitamos conectar un **amperímetro en serie** para que la corriente fluya a través de él, piensa en ello como un medidor de agua, el agua en una tubería, necesita fluir a través del medidor de agua para que sepamos cuánta agua está fluyendo, del mismo modo necesitamos que los electrones fluyan a través de nuestro amperímetro. En lugar de usar un amperímetro vamos a usar un multímetro ya que podemos hacer mucho más con este dispositivo.



Si conectamos esta batería de 1.5 volts y esta lámpara que tiene una resistencia de un ohm obtenemos una lectura de corriente de 1.5 amperes. Debido a que los componentes de este circuito están conectados en serie la corriente es la misma en cualquier parte del circuito por lo que podemos tomar una medición en cualquier lugar y es el mismo valor. si añadimos otra lámpara al circuito conectado de nuevo en serie y la lámpara también tiene una resistencia de un ohm entonces estamos añadiendo más resistencia al circuito, así que ahora es más difícil que los electrones fluyan a través de él y así vemos una reducción de la corriente, en este caso obtenemos una lectura de 0 75 amperes.



si ahora conectamos el circuito con dos lámparas en paralelo ambas con una resistencia de 1 ohm y conectamos este circuito a una batería de 1,5 volts, entonces en el cable principal desde y hacia la batería obtenemos 3 amperes, pero por la división en cada lámpara obtenemos 1,5 amperes porque la circulación de electrones se divide, así que se comparten entre las dos lámparas, el camino se une de nuevo así que obtenemos la corriente total del circuito combinado de 3 amperes porque ambas lámparas tienen la misma resistencia, tienen la misma corriente pero por ejemplo si una tiene una resistencia de 1 ohm y la otra tiene una resistencia de 3 oms, entonces en el cable principal obtenemos una lectura de dos amperes, en la división para la lámpara obtenemos 1.5 amperes y la división para la lámpara b obtenemos 0.5, sin embargo noten que la lámpara b es más tenue, eso es porque la resistencia es mayor, lo que dificulta que los electrones fluyan a través de ella. en ambos casos los amperes y las divisiones se suman y son iguales a la corriente total que fluye en el cable principal hacia y desde la batería, por lo tanto podemos añadir resistencias a nuestros circuitos para restringir cuenta corriente puede fluir.



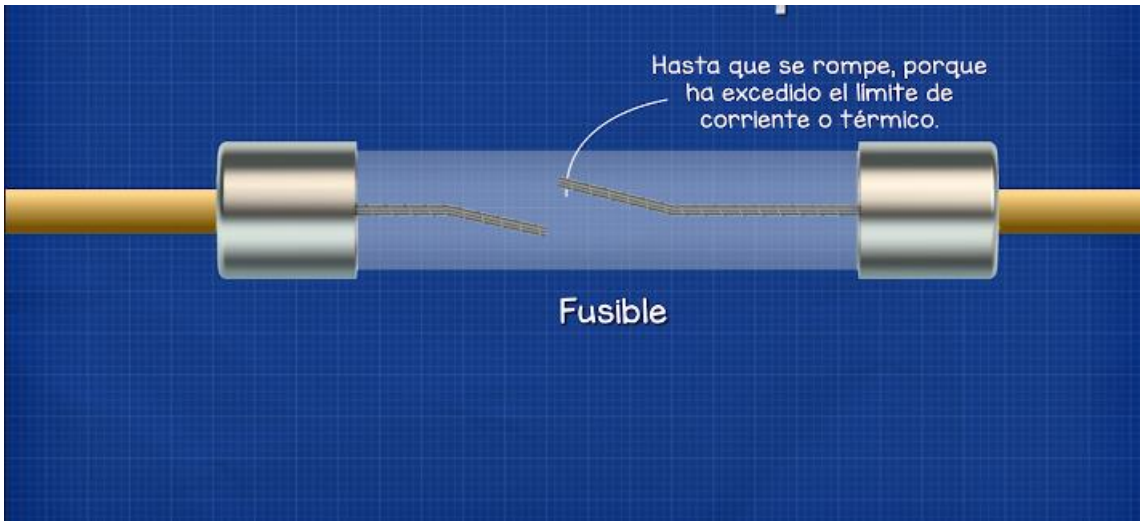
Resistencias

Las resistencias hacen más difícil que los electrones fluyan a través de un circuito y por eso **añadimos resistencias a los circuitos porque reducen la corriente,** es como tener una torcedura en una tubería, esto añadirá resistencia al flujo de agua lo que reduce la cantidad de agua que puede fluir a través de ella y como el agua está chocando con la pared de la tubería va a perder energía, así que tenemos una caída de presión. Lo mismo ocurre con una resistencia, el material dificulta el paso de los electrones, los electrones van a colisionar y desperdiciar energía, **así que tenemos una caída de voltaje.** Esta energía desperdiciada necesita ir a algún lugar, así que se escapa en forma de calor.

Por ejemplo este Led tiene una capacidad máxima de 22 miliamperios o 0.022 amperes, queremos conectar esto a una fuente de alimentación de 9 volts, si usamos una resistencia de 100 ohms entonces la corriente sería de 0,09 amperes lo cual es demasiado y el Led se quemara. Si usamos una resistencia de 450 ohms la corriente es de 0.02 amperes lo que está por debajo del límite así que eso debería estar bien, si usamos una resistencia de 900 ohms entonces la corriente es de 0,01 amperes lo que es demasiado bajo, por lo que el Led no brillará con mucha intensidad.

Fusibles

Los fusibles básicamente tienen un fino trozo de alambre en su interior que está diseñado para manejar una cierta cantidad de corriente que fluye a través de él, en este caso este está diseñado para manejar 3 amperes, si fluye demasiada corriente en el circuito entonces el fusible se quemará y esto abrirá o romperá el circuito para proteger los costosos componentes eléctricos. El fusible actúa como un punto débil y es muy barato de reemplazar así que puedes encontrarlos instalados en paneles de circuitos.



La batería (o pila)

Una batería es un dispositivo que se utiliza para almacenar energía para un futuro cuando la necesitemos. Utilizamos baterías para alimentar pequeños dispositivos electrónicos como una linterna, la energía se almacena como energía química y ésta puede convertirse en energía eléctrica cuando la necesitemos. Si observamos un simple circuito de batería y lámpara para iluminar, en la lámpara necesitamos que muchos electrones fluyan a través de ella para iluminarse, la batería va a proporcionar la fuerza de empuje que permite que los electrones fluyan a través de la lámpara, solo tenemos que conectar la lámpara a través de las terminales positivas y negativas de la batería para completar el circuito.

La batería solo puede empujar los electrones durante una cierta cantidad de tiempo aunque este tiempo depende de cuánta energía se almacena en el interior de la batería y de cuánta demanda la carga tenemos. Cuando hablamos de carga en un circuito eléctrico nos referimos a cualquier componente que requiera electricidad para funcionar, podrían ser cosas como resistencias Led motores de corriente directa o incluso tarjetas de circuito completas.

Algunas baterías pueden ser recargadas y esto se indicará claramente en la parte lateral pero la típica batería alcalina doméstica no se puede, así que simplemente se desecha cuando se agota la energía, éstas pueden ser recicladas así que asegúrate de deshacerte de ellas de forma responsable.

Una típica batería alcalina de un voltio y medio se parece a esto, pero los colores varían según el fabricante. Al observar la batería normalmente tenemos un envoltorio de plástico bien ajustado en el exterior, esto va a aislar a la batería pero también nos dirá información importante como la capacidad y el voltaje así que como que extremo es el positivo y el negativo.

El extremo positivo se conoce como cátodo y tendrá esta superficie extendida que sobresale hacia el exterior, el extremo negativo será plano y el negativo se conoce como ánodo. Estas dos terminales están eléctricamente aisladas una de la otra.

Bajo la envoltura encontramos la carcasa principal que suele ser de acero con un recubrimiento de níquel esto mantiene todos los componentes internos en su lugar y evita que interactúen con elementos de la atmósfera como el aire y el agua. Dentro de la carcasa tenemos múltiples capas de diferentes materiales estos materiales son especialmente seleccionados porque sus reacciones químicas crean ciertos niveles de voltajes y corrientes

La electricidad es el flujo de electrones en un circuito. Las baterías pueden proporcionar la fuerza de empuje que mueve los electrones a través del circuito, los electrones quieren volver a su fuente e inmediatamente tomarán cualquier camino que sea posible para lograrlo. Colocando cosas como lámparas en el camino de los electrones podemos forzarlos a hacer un trabajo para nosotros como iluminar la lámpara. Las baterías producen electricidad CD o corriente directa esto significa que los electrones fluyen en una sola dirección desde el negativo al positivo como un río que fluye en una sola dirección. El flujo de electrones va de negativo a positivo, pero puede que estés acostumbrado a ver la corriente convencional que va de positivo a negativo.

El flujo de electrones es lo que realmente ocurre pero la corriente convencional fue la teoría original que aún se utiliza y enseña ampliamente hasta hoy, solo hay que ser consciente de las dos y de la que estamos usando.

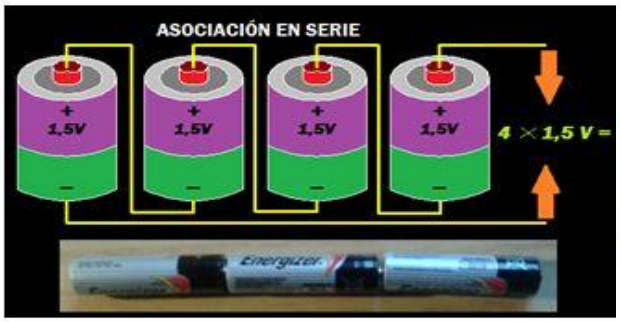
La electricidad que se obtiene de las tomas de corriente en sus casas proporciona electricidad CA o corriente alterna, esto es diferente a la electricidad proporcionada por una batería. Con la corriente alterna los electrones fluyen hacia adelante y hacia atrás continuamente como la marea del mar que fluye hacia dentro y hacia afuera entre la marea alta y la baja.

Voltaje

El voltaje es como la presión en un tanque de agua, para saber cuánta presión tenemos debemos comparar la presión dentro de la tubería con la presión exterior y utilizamos un manómetro para hacerlo. En cuanto al voltaje usamos un **voltímetro** para medir la diferencia de voltaje entre dos puntos diferentes. Si medimos la diferencia a través de la batería obtenemos 1.5 volts, pero si medimos el mismo extremo entonces obtenemos 0 volts, porque es el mismo extremo así que no hay diferencia. Algunos materiales permiten que los electrones pasen fácilmente, estos se conocen como **conductores** el cobre y la mayoría de los metales son ejemplos de esto. Otros materiales no permiten que los electrones pasen a través de ellos estos se conocen como **aislantes,** el caucho y la mayoría de los plásticos son ejemplos, por eso usamos cables de cobre con aislantes de plástico. El cobre transporta la electricidad a donde la necesitamos y el plástico nos mantiene a salvo.

Podemos usar una batería para alimentar algunos componentes pero normalmente una sola batería no es suficiente para alimentar nuestros dispositivos, para eso necesitamos combinar las baterías.

Podemos conectar las baterías de dos maneras diferentes, en **serie** o en **paralelo** hemos cubierto estos tipos de circuitos con gran detalle más adelante.



Cuando conectamos las baterías en serie el voltaje de cada batería se suma así que dos baterías de 1.5 volts nos dan 3 y 3 baterías nos dan 4.5 volts. El voltaje real puede ser ligeramente diferente. En el mundo real el voltaje aumenta porque cada batería está potenciando los electrones que entran en ella por lo que obtenemos un voltaje más alto si, conectamos las baterías en paralelo sólo obtendremos 1.5 volts independientemente de cuántas conectemos, eso es porque el camino se fusionan el suministro pero se divide en el retorno, por lo que los electrones no serán impulsados. Sin embargo este tipo de configuración será capaz de proporcionar más corriente y también tendrá una mayor capacidad para que podamos alimentar algo durante más tiempo. Por ejemplo si la batería tuviera una capacidad de 1.200 mA/hora y pusiéramos dos en paralelo tendríamos una capacidad de 2.400mA/h, pero un voltaje de un volt y medio. Las que están en serie tenemos una capacidad de 1.200mA/H, pero un voltaje de 3 volts.

Usamos baterías para alimentar nuestros circuitos pero ¿cuánto tiempo puede una batería alimentarlo? cuando miramos el empaquetado o la hoja de datos de una batería vemos un valor con las letras mA por un lado, este es el valor en miliamperios.

Por ejemplo esta tiene un voltaje de 1.2V y una capacidad de 1900ma/h, eso significa que teóricamente podría proporcionar una corriente de 1900 miliamperios por una hora. Sin embargo en la vida real probablemente no durará tanto tiempo porque la reacción química se reduce con el tiempo, por lo que la resistencia interna de la batería cambia a medida que se vacía y acorta su vida útil. Hay muchas otras cosas que afectan a esto como la antigüedad o la temperatura en que se encuentran al momento de usarlas.

No hay una forma real de calcular con precisión la duración de la batería, la mejor forma es simplemente probarla. Sin embargo podemos hacer una estimación de la vida útil con la siguiente fórmula:

Duración de la batería = capacidad mA/h dividida por la corriente del circuito en mA

Así por ejemplo en un circuito calculamos una demanda de 19 miliamperios y la batería tiene una capacidad de 1.900 mA/h por lo que 1.900 dividido entre 19 da 100 horas. Sin embargo este es realmente el mejor escenario posible y en realidad es casi seguro que no llegue ahí.

Capítulo 2: Placa Arduino

Conociendo la placa

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso.

Arduino es una placa basada en un microcontrolador ATMEL. Los microcontroladores son circuitos integrados en los que se pueden grabar instrucciones, las cuales las escribes con el lenguaje de programación que puedes utilizar en el entorno Arduino IDE. Estas instrucciones permiten crear programas que interactúan con los circuitos de la placa.

El microcontrolador de Arduino posee lo que se llama una interfaz de entrada, que es una conexión en la que podemos conectar en la placa diferentes tipos de periféricos. La información de estos periféricos que conectes se trasladará al microcontrolador, el cual se encargará de procesar los datos que le lleguen a través de ellos.

El tipo de periféricos que puedas utilizar para enviar datos al microcontrolador depende en gran medida de qué uso le estés pensando dar. Pueden ser cámaras para obtener imágenes, teclados para introducir datos, o diferentes tipos de sensores.

También cuenta con una interfaz de salida, que es la que se encarga de llevar la información que se ha procesado en el Arduino a otros periféricos. Estos periféricos pueden ser pantallas o altavoces en los que reproducir los datos procesados, pero también pueden ser otras placas o controladores.

Tipos de señales

Analógica

Digital

PWM

Capítulo 3: Manual de programación

C++/Arduino

En este capítulo veremos una de las partes fundamentales de Arduino, su programación, lenguaje y funciones básicas para adaptar el controlador de la placa a nuestras necesidades, por lo general el código entre diferentes placas de Arduino son casi idénticos, salvo leves diferencias de disposición de pines o limitantes de capacidad y procesamiento.

Estructura de un programa en Arduino

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones.

```
void setup()
{
  estamentos;
}
  void loop()
{
  estamentos;
}
```

En donde setup () es la parte encargada de recoger la configuración y loop () es la que contienen el programa que se ejecutará cíclicamente (de ahí el termino loop –bucle-). Ambas funciones son necesarias para que el programa trabaje. La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras. La función bucle (loop) siguiente contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc.) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

Setup

La función setup () se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar.

```
void setup()
    {
        pinMode(pin, OUTPUT); // configura el 'pin' como salida
    }
```

Loop

Después de llamar a setup (), la función loop () hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la tarjeta

```
void loop()
      {
            digitalWrite(pin, HIGH);   // pone en uno (on, 5v) el ´pin´
            delay(1000);               // espera un segundo (1000 ms)
            digitalWrite(pin, LOW);    // pone en cero (off, 0v.) el ´pin´
            delay(1000);
      }
```

Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de estamentos que son ejecutados cuando se llama a la función. Son funciones setup () y loop () de las que ya se ha hablado. Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor “type”. Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “función vacía”. Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán, si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
type  nombreFunción(parámetros)
      {
          estamentos;
      }
```

La función siguiente devuelve un número entero, delayVal () se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino.

```
int  delayVal()
   {
      int  v;                    // crea una variable temporal  'v'
      v= analogRead(pot);        // lee el valor del potenciómetro
      v  /=  4;                  // convierte  0-1023 a  0-255
      return  v;                 // devuelve el valor final
   }
```

Al principio se declara como una variable local,´v́ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor´v́ y se retornaría al programa principal. Esta función cuando se ejecuta devuelve el valor de tipo entero "v ".

{ } Entre llaves

Las llaves sirven para definir el principio y el final de un bloque de instrucciones. Se utilizan para los bloques de programación setup (), loop (), if... etc.

```
type funcion()
      {
        estamentos;
      }
```

Una llave de apertura "{"siempre debe ir seguida de una llave de cierre "}", si no es así el programa dará errores. El entorno de programación de Arduino incluye una herramienta de gran utilidad para comprobar el total de llaves. Sólo tienes que hacer click en el punto de inserción de una llave abierta e inmediatamente se marca el correspondiente cierre de ese bloque (llave cerrada).

; Punto y coma

El punto y coma ";" se utiliza para separar instrucciones en el lenguaje de programación de Arduino. También se utiliza para separar elementos en una instrucción de tipo "bucle for".

```
int x = 13;      // declara la variable 'x' como tipo entero de valor 13
```

Comentarios

Una línea de comentario empieza con / / y terminan con la siguiente línea de código. Al igual que los comentarios de bloque, los de línea son ignoradas por el programa y no ocupan espacio en la memoria.

```
// Soy un comentario

/* Soy un comentario multilinea
   Puedo escribir aquí
   Y terminar aquí /*
```

Variables

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada variableEntrada y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int variableEntrada = 0;          // declara una variable y le asigna el valor 0
variableEntrada = analogRead(2);// la variable recoge el valor analógico del PIN2
```

'variableEntrada' es la variable en sí. La primera línea declara que será de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código. Una vez que una variable ha sido asignada, o re-asignada, usted puede probar su valor para ver si cumple ciertas condiciones (instrucciones if...), o puede utilizar directamente su valor. Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (delay) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100) // pregunta si la variable es menor de 100
  {
    entradaVariable = 100;// si es cierto asigna el valor 100 a esta
  }
delay(entradaVariable);     // usa el valor como retardo
```

Nota: Las variables deben tomar nombres descriptivos, para hacer el código más legible. Nombres de variables pueden ser "contactoSensor" o "pulsador", para ayudar al programador y a cualquier otra persona a leer el código y entender lo que representa la variable. Nombres de variables como "var" o "valor", facilitan muy poco que el código sea inteligible. Una variable puede ser cualquier nombre o palabra que no sea una palabra reservada en el entorno de Arduino.

Declaración de variables

Todas las variables tienen que declararse antes de que puedan 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. Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas.

El siguiente ejemplo declara la variable entradaVariable como una variable de tipo entero "int", y asignándole un valor inicial igual a cero. Esto se llama una asignación.

```
int entradaVariable = 0;
```

Una variable puede ser declarada en una serie de lugares del programa y en función del lugar en donde se lleve a cabo la definición esto determinará en que partes del programa se podrá hacer uso de ella

Utilización de una variable

Una variable puede ser declarada al inicio del programa antes de la parte de configuración setup (), a nivel local dentro de las funciones, y, a veces, dentro de un bloque, como para los bucles del tipo if... for..., etc. En función del lugar de declaración de la variable así se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella. Una variable global es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de setup (). Una 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. Declaración de variables Utilización de una variable

```
int value; // 'value' es visible para cualquier función
void setup()
      {
            // no es necesario configurar
      }

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

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.

Tipos de variables

Byte

Byte almacena un valor numérico de 8 bits sin decimales. Tienen un rango entre 0 y 255

```
byte unaVariable = 180;  // declara 'unaVariable' como tipo byte
```

Int

Enteros son un tipo de datos primarios que almacenan valores numéricos de 16 bits sin decimales comprendidos en el rango 32,767 a -32,768.

```
int unaVariable = 1500;  // declara 'unaVariable' como una variable
                         de tipo entero
```

Nota: Las variables de tipo entero "int" pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor se x pasará a ser -32.768. (Algo así como que el valor da la vuelta)

Long

El formato de variable numérica de tipo extendido "long" se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

```
long unaVariable = 90000; // declara 'unaVariable' como tipo long
```

Float

El formato de dato del tipo "punto flotante" "float" se aplica a los números con decimales. Los números de punto flotante tienen una mayor resolución que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

```
float unaVariable = 3.14; // declara 'unaVariable' como tipo flotante
```

Nota: Los números de punto flotante no son exactos, y pueden producir resultados extraños en las comparaciones. Los cálculos matemáticos de punto flotante son también mucho más lentos que los del tipo de números enteros, por lo que debe evitarse su uso si es posible.

Array

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado

```
int miArray[] = {valor0, valor1, valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

```
int miArray[5];      // declara un array de enteros de 6 posiciones
miArray[3] = 10;     // asigna l valor 10 a la posición 4
```

Para leer de un array basta con escribir el nombre y la posición a leer:

```
x = miArray[3];      // x ahora es igual a 10 que está en la posición 3
                     del array
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED. Utilizando un bucle tipo for, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que hemos escrito dentro del array parpadeo [], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i".

```
int ledPin = 10;       // Salida LED en el PIN 10
byte parpadeo[] = {180, 30, 255, 200, 10, 90, 150, 60}; // array de 8 valores
                                                          diferentes
    void setup()
    {
      pinMode(ledPin, OUTPUT); //configura la salida PIN 10
    }

      void loop()       // bucle del programa

    {
      for(int i=0; i<8; i++) // crea un bucle tipo for utilizando la variable i de 0 a 7
    {

      analogWrite(ledPin, parpadeo[i]); // escribe en la salida PIN 10 el valor al
                                           que apunta i dentro del array
                                           parpadeo[]
    delay(200);  // espera 200ms

    }
}
```

Aritmética

Los operadores aritméticos que se incluyen en el entorno de programación son suma, resta, multiplicación y división. Estos devuelven la suma, diferencia, producto, o cociente (respectivamente) de dos operandos

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

La operaciones se efectúa teniendo en cuanta el tipo de datos que hemos definido para los operandos (int, double, float, etc...), por lo que, por ejemplo, si definimos 9 y 4 como enteros "int", 9 / 4 devuelve de resultado 2 en lugar de 2,25 ya que el 9 y 4 se valores de tipo entero "int" (enteros) y no se reconocen los decimales con este tipo de datos.

Esto también significa que la operación puede sufrir un desbordamiento si el resultado es más grande que lo que puede ser almacenada en el tipo de datos. Recordemos el alcance de los tipos de datos numéricos que ya hemos explicado anteriormente. Si los operandos son de diferentes tipos, para el cálculo se utilizará el tipo más grande de los operandos en juego. Por ejemplo, si

uno de los números (operandos) es del tipo float y otra de tipo integer, para el cálculo se utilizará el método de float es decir el método de coma flotante. Elija el tamaño de las variables de tal manera que sea lo suficientemente grande como para que los resultados sean lo precisos que usted desea. Para las operaciones que requieran decimales utilice variables tipo float, pero sea consciente de que las operaciones con este tipo de variables son más lentas a la hora de realizarse el cómputo... Nota: Utilice el operador (int) myFloat para convertir un tipo de variable a otro sobre la marcha. Por ejemplo, i = (int) 3,6 establecerá i igual a 3.

Asignaciones compuestas

Las asignaciones compuestas combinan una operación aritmética con una variable asignada. Estas son comúnmente utilizadas en los bucles tal como se describe más adelante. Estas asignaciones compuestas 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 incrementra 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
```

Nota: Por ejemplo, x * = 3 hace que x se convierta en el triple del antiguo valor x y por lo tanto x es reasignada al nuevo valor.

Operadores de comparación

Las comparaciones de una variable o constante con otra se utilizan con frecuencia en las estructuras condicionales del tipo if... para testear si una condición es verdadera. En los ejemplos que siguen en las próximas páginas se verá su utilización práctica usando los siguientes tipo de condicionales:

```
x == y     // x es igual a y
x != y     // x no es igual a y
x <  y     // x es menor que y
x >  y     // x es mayor que y

x <= y     // x es menor o igual que y
x >= y     // x es mayor o igual que y
```

Operadores lógicos

Los operadores lógicos son usualmente una forma de comparar dos expresiones y devolver un VERDADERO o FALSO dependiendo del operador. Existen tres operadores lógicos, AND (&&), OR (||) y NOT (!), que a menudo se utilizan en estamentos de tipo if...:

```
Logical AND:
if (x > 0 && x < 5)        // cierto sólo si las dos expresiones son ciertas

Logical OR:
if (x > 0 || y > 0)        // cierto si una cualquiera de las expresiones
                           es cierta

Logical NOT:
if (!x > 0)                //  cierto solo si la expresión es falsa
```

Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos.

Cierto-falso (true-false)

Estas son constantes booleanas que definen los niveles HIGH (alto) y LOW (bajo) cuando estos se refieren al estado de las salidas digitales. FALSE se asocia con 0 (cero), mientras que TRUE se asocia con 1, pero TRUE también puede ser cualquier otra cosa excepto cero. Por lo tanto, en sentido booleano, -1, 2 y -200 son todos también se definen como TRUE. (Esto es importante tenerlo en cuanta)

```
if (b == TRUE);
{
ejecutar las instrucciones;
}
```

Alto-Bajo (High-Low)

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. ALTO se define como en la lógica de nivel 1, ON, o 5 voltios, mientras que BAJO es lógica nivel 0, OFF, o 0 voltios.

```
digitalWrite(13, HIGH); // activa la salida 13 con un nivel alto (5v.)
```

Entrada-Salida (Input-Output)

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.

```
pinMode(13, OUTPUT); // designamos que el PIN 13 es una salida
```

Condicionales

If (si)

If es un estamento que se utiliza para probar si una determinada condición se ha alcanzado, como por ejemplo averiguar si un valor analógico está por encima de un cierto número, y ejecutar una serie de declaraciones (operaciones) que se escriben dentro de llaves, si es verdad. Si es falso (la condición no se cumple) el programa salta y no ejecuta las operaciones que están dentro de las llaves, El formato para if es el siguiente:

```
if (unaVariable ?? valor)
{
ejecutaInstrucciones;
}
```

En el ejemplo anterior se compara una variable con un valor, el cual puede ser una variable o constante. Si la comparación, o la condición entre paréntesis se cumple (es cierta), las declaraciones dentro de los corchetes se ejecutan. Si no es así, el programa salta sobre ellas y sigue.

Nota: Tenga en cuenta el uso especial del símbolo '=', poner dentro de if (x = 10), podría parecer que es válido pero sin embargo no lo es ya que esa expresión asigna el valor 10 a la variable x, por eso dentro de la estructura if se utilizaría X==10 que en este caso lo que hace el programa es comprobar si el valor de x es 10... Ambas cosas son distintas por lo tanto dentro de las estructuras if, cuando se pregunte por un valor se debe poner el signo doble de igual "=="

If-else (sí... sino)

If... else viene a ser un estructura que se ejecuta en respuesta a la idea "si esto no se cumple haz esto otro". Por ejemplo, si se desea probar una entrada digital, y hacer una cosa si la entrada fue alto o hacer otra cosa si la entrada es baja, usted escribiría que de esta manera:

```
if (inputPin == HIGH) // si el valor de la entrada inputPin es alto
      {
            instruccionesA; //ejecuta si se cumple la condición
      }
else
      {
            instruccionesB; //ejecuta si no se cumple la condición
      }
```

Else puede ir precedido de otra condición de manera que se pueden establecer varias estructuras condicionales de tipo unas dentro de las otras (anidamiento) de forma que sean mutuamente excluyentes pudiéndose ejecutar a la vez. Es incluso posible tener un número ilimitado de estos condicionales. Recuerde sin embargo qué sólo un conjunto de declaraciones se llevará a cabo dependiendo de la condición probada:

```
if (inputPin < 500)
      {
            instruccionesA; // ejecuta las operaciones A
      }
else if (inputPin >= 1000)
      {
            instruccionesB; // ejecuta las operacione B
      }
else
      {
            instruccionesC; // ejecuta las operaciones C
      }
```

Nota: Un estamento de tipo if prueba simplemente si la condición dentro del paréntesis es verdadera o falsa. Esta declaración puede ser cualquier declaración válida. En el anterior ejemplo, si cambiamos y ponemos (inputPin == HIGH). En este caso, el estamento if sólo chequearía si la entrada especificado está en nivel alto (HIGH), o +5 v.

For (para...)

La declaración for se usa para repetir un bloque de sentencias encerradas entre llaves un número 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:

```
for (inicialización; condición; expresión)
{
      ejecutaInstrucciones;
}
```

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, el bucle termina. El siguiente ejemplo inicia el entero i en el 0, y la condición es probar que el valor es inferior a 20 y si es cierta i se incrementa en 1 y se vuelven a ejecutar las instrucciones que hay dentro de las llaves:

```
for (int i=0; i<20; i++)          // declara i, prueba que es menor que
                                  20, incrementa i en 1
{
      digitalWrite(13, HIGH);     // envía un 1 al pin 13
      delay(250);                 // espera ¼ seg.
      digitalWrite(13, LOW);      // envía un 0 al pin 13
      delay(250);                 // espera ¼ de seg.
}
```

While

Un bucle del tipo While es un bucle de ejecución continua mientras se cumpla la expresión colocada entre paréntesis en la cabecera del bucle. La variable de prueba tendrá que cambiar

para salir del bucle. La situación podrá cambiar a expensas de una expresión dentro el código del bucle o también por el cambio de un valor en una entrada de un sensor

```
while (unaVariable ?? valor)
{

      ejecutarSentencias;
}
```

El siguiente ejemplo testea si la variable "unaVariable" es inferior a 200 y, si es verdad, ejecuta las declaraciones dentro de los corchetes y continuará ejecutando el bucle hasta que 'unaVariable' no sea inferior a 200.

```
While (unaVariable < 200)      // testea si es menor que 200
{
      instrucciones;           // ejecuta las instrucciones entre llaves
      unaVariable++;           // incrementa la variable en 1
}
```

Do...while

El bucle do While funciona de la misma manera que el bucle While, con la salvedad de que la condición se prueba al final del bucle, por lo que el bucle siempre se ejecutará al menos una vez.

```
do
{
      Instrucciones;
} while (unaVariable ?? valor);
```

El siguiente ejemplo asigna el valor leído leeSensor () a la variable 'x', espera 50 milisegundos, y luego continúa mientras que el valor de la 'x' sea inferior a 100:

```
do
{
      x = leeSensor();
      delay(50);
} while (x < 100);
```

Funciones especiales de Arduino

PinMode (pin, mode)

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).

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

Los terminales de Arduino, por defecto, están configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines configurados como entrada quedan, bajo el punto de vista eléctrico, como entradas en estado de alta

impedancia. Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias se acceden de la siguiente manera:

```
pinMode(pin, INPUT);      // configura el 'pin' como entrada
digitalWrite(pin, HIGH);  // activa las resistencias internas
```

Las resistencias internas normalmente se utilizan para conectar las entradas a interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es simplemente un método para activar las resistencias interiores. Los pins configurado como OUTPUT (salida) se dice que están en un estado de baja impedancia estado y pueden proporcionar 40 mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores. Un cortocircuito en las patillas Arduino provocará una corriente elevada que puede dañar o destruir el chip Atmega. A menudo es una buena idea conectar en la OUTUPT (salida) una resistencia externa de 470 o de 1000 Ω.

digitalRead(pin)

Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

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

digitalWrite (pin, value)

Envía al´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

```
digitalWrite(pin, HIGH);  // deposita en el 'pin' un valor HIGH (alto o 1)
```

El siguiente ejemplo lee el estado de un pulsador conectado a una entrada digital y lo escribe en el´pin´de salida LED:

```
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
}
```

analogRead (pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023

```
valor = analogRead(pin); // asigna a valor lo que lee en la entrada 'pin'
```

Nota: Los pins analógicos (0-5) a diferencia de los pines digitales, no necesitan ser declarados como INPUT u OUPUT ya que son siempre INPUT´s.

analogWrite(pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pin´s de Arduino marcados como "pin PWM". El más reciente Arduino, que implementa el chip ATmega168, permite habilitar como salidas analógicas tipo PWM los pines 3, 5, 6, 9, 10 y 11. Los modelos de Arduino más antiguos que implementan el chip ATmega8, solo tiene habilitadas para esta función los pines 9, 10 y 11. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

```
analogWrite(pin, valor); // escribe 'valor' en el 'pin' definido como
                         analógico
```

Si enviamos el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado. Para valores de entre 0 y 255, el pin saca tensiones entre 0 y 5 voltios - el valor HIGH de salida equivale a 5v (5 voltios). Teniendo en cuenta el concepto de señal PWM, por ejemplo, un valor de 64 equivaldrá a mantener 0 voltios de tres cuartas partes del tiempo y 5 voltios a una cuarta parte del tiempo; un valor de 128 equivaldrá a mantener la salida en 0 la mitad del tiempo y 5 voltios la otra mitad del tiempo, y un valor de 192 equivaldrá a mantener en la salida 0 voltios una cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo restante.

Debido a que esta es una función de hardware, en el pin de salida analógica (PWN) se generará una onda constante después de ejecutada la instrucción analogWrite hasta que se llegue a ejecutar otra instrucción analogWrite (o una llamada a digitalRead o digitalWrite en el mismo pin).

Nota: Las salidas analógicas a diferencia de las digitales, no necesitan ser declaradas como INPUT u OUTPUT... El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor dividiéndolo por 4, y envía el nuevo valor convertido a una salida del tipo PWM o salida analógica:

```
int led = 10;          // define el pin 10 como ´led´
int analog =  0;       // define el pin 0 como ´analog´
int valor;             // define la variable ´valor´

void setup(){}         // no es necesario configurar entradas y salidas

void loop()
{
      valor = analogRead(analog);     // lee el pin 0 y lo asocia a la
                                      variable valor
      valor /= 4; //divide valor entre 4 y lo reasigna a valor
      analogWrite(led, value); // escribe en el pin10  valor
}
```

delay(ms)

Detiene la ejecución del programa la cantidad de tiempo en ms que se indica en la propia instrucción. De tal manera que 1000 equivale a 1seg.

```
delay(1000);      // espera 1 segundo
```

millis()

Devuelve el número de milisegundos transcurrido desde el inicio del programa en Arduino hasta el momento actual. Normalmente será un valor grande (dependiendo del tiempo que esté en marcha la aplicación después de cargada o después de la última vez que se pulsó el botón “reset” de la tarjeta)..

```
valor = millis();       // valor recoge el número de milisegundos
```

Nota: Este número se desbordará (si no se resetea de nuevo a cero), después de aproximadamente 9 horas.

min(x, y)

Calcula el mínimo de dos números para cualquier tipo de datos devolviendo el número más pequeño.

```
valor = min(valor, 100); // asigna a valor el más pequeños de los dos números
especificados.
```

Si 'valor' es menor que 100 valor recogerá su propio valor si ´valor´ es mayor que 100 valor pasara a valer 100.

max(x, y)

Calcula el máximo de dos números para cualquier tipo de datos devolviendo el número mayor de los dos.

```
valor = max(valor, 100); // asigna  a valor el mayor de los dos números  'valor' y
100.
```

randomSeed(seed)

Establece un valor, o semilla, como punto de partida para la función random()

```
randomSeed(valor);   // hace que valor sea la semilla del random
```

Debido a que Arduino es incapaz de crear un verdadero número aleatorio, randomSeed le permite colocar una variable, constante, u otra función de control dentro de la función random, lo que permite generar números aleatorios "al azar". Hay una variedad de semillas, o funciones, que pueden ser utilizados en esta función, incluido millis () o incluso analogRead () que permite leer ruido eléctrico a través de un pin analógico.

random(max)

random(min, max)

La función random devuelve un número aleatorio entero de un intervalo de valores especificado entre los valores min y max.

```
valor = random(100, 200);      // asigna a la variable 'valor' un numero aleatorio
                               comprendido entre 100-200
```

Nota: Use esta función después de usar el randomSeed(). El siguiente ejemplo genera un valor aleatorio entre 0-255 y lo envía a una salida analógica PWM :

```
int randNumber;      // variable que almacena el valor aleatorio
int led = 10;        // define led como 10

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

void loop()
{
randomSeed(millis());            // genera una semilla para aleatorio a partir
                                 de la función millis()
randNumber = random(255);        // genera número aleatorio entre 0-255
analogWrite(led, randNumber);    // envía a la salida led de tipo PWM el valor
delay(500);                      // espera 0,5 seg.
}
```

Serial.begin(rate)

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de velocidad para comunicarse con el ordenador es 9600, aunque otras velocidades pueden ser soportadas.

```
void setup()
    {
        Serial.begin(9600);    // abre el Puerto serie
    }                          // configurando la velocidad en 9600 bps
```

Nota: Cuando se utiliza la comunicación serie los pins digital 0 (RX) y 1 (TX) no puede utilizarse al mismo tiempo.

Serial.println(data)

Imprime los datos en el puerto serie, seguido por un retorno de carro automático y salto de línea. Este comando toma la misma forma que Serial.print (), pero es más fácil para la lectura de los datos en el Monitor Serie del software.

```
Serial.println(analogValue);        // envía el valor 'analogValue' al puerto
```

El siguiente ejemplo toma de una lectura analógica pin0 y envía estos datos al ordenador cada 1 segundo.

```
void setup()
    {
        Serial.begin(9600);     // configura el puerto serie a 9600bps
    }

void loop()
    {
        Serial.println(analogRead(0)); // envía valor analógico
        delay(1000);                   // espera 1 segundo
    }
```

Serial.avaible()

int Serial.available() Obtiene un número entero con el número de bytes (caracteres) diponibles para leer o capturar desde el puerto serie

Devuelve Un entero con el número de bytes disponibles para leer desde el buffer serie, o 0 si no hay ninguno. Si hay algún dato disponible, SerialAvailable() será mayor que 0. El buffer serie puede almacenar como máximo 64 bytes.

Ejemplo:

```
int incomingByte = 0; // almacena el dato serie
void setup() {
    Serial.begin(9600); // abre el puerto serie, y le asigna la velocidad de 9600
bps
}
void loop() {
    // envía datos sólo si los recibe:
    if (Serial.available() > 0) {
            // lee el byte de entrada:
            incomingByte = Serial.read();
        //lo vuelca a pantalla
            Serial.print("I received: ");
            Serial.println(incomingByte, DEC);
    }
}
```

Serial.Read()

int Serial.Read() Lee o captura un byte (un caracter) desde el puerto serie.

Devuelve :El siguiente byte (carácter) desde el puerto serie, o -1 si no hay ninguno.

Ejemplo

```
int incomingByte = 0; // almacenar el dato serie
void setup() {
    Serial.begin(9600); // abre el puerto serie,y le asigna la velocidad de 9600
bps
}
void loop() {
    // envía datos sólo si los recibe:
    if (Serial.available() > 0) {
                // lee el byte de entrada:
                incomingByte = Serial.read();
            //lo vuelca a pantalla
                Serial.print("I received: ");
                Serial.println(incomingByte, DEC);
    }
}
```