

Arduino Programming Notebook

Escrito y compilado por Brian W. Evans
Traducido al español por JarBot.

Arduino Programming Notebook
Written and compiled by Brian W. Evans
Traducido al español por **JarBot** (Noviembre 2009).

Publicado:
Primera Edición Agosto 2007
Segunda Edición Septiembre 2008

Prólogo

Este documento sirve como un fácil y conveniente compañero, para el principiante que quiere aprender a utilizar y programar el **microcontrolador Arduino**. Su objetivo es ser una referencia secundaria para ser utilizada junto con otros sitios web, libros, talleres, o clases. Para hacerlo simple, ciertas exclusiones han dado lugar a un ligero énfasis utilizando el microcontrolador para los propósitos independiente y, por ejemplo, deja fuera las matrices más complejas o las formas avanzadas de comunicación en serie. A partir de la estructura básica de Arduino derivado programación en lenguaje C, esta referencia sirve para describir los elementos más comunes e ilustrar su uso con ejemplos y fragmentos de código.. Por encima de todo, este documento no sería posible sin la gran comunidad de políticas y de masas de corte de material original que se encuentra en el Arduino página web, y un foro en <http://www.arduino.cc>.

ESTRUCTURA

structure
setup()
loop()
functions
{ } curly braces
; semicolon
/*... */ block comments
// line comments

VARIABLES

variables
variable declaration
variable scope

TIPOS DE DATOS

byte
int
long
float
arrays

ARITMETICA

arithmetic
compound assignments
comparison operators
logical operators

CONSTANTES

constants
true/false
high/low
input/output

CONTROL DE FLUJO

if
if... else
for
while
do... while

ENTRADA/SALIDA DIGITAL

pinMode(pin, mode)
digitalRead(pin)
digitalWrite(pin, value)

ENTRADA/ SALIDA ANALOGICA

analogRead(pin)
analogWrite(pin, value)

TIEMPO

delay(ms)
millis()

MATEMATICAS

min(x, y)
max(x, y)

ALEATORIO

randomSeed(seed)
random(min, max)

SERIAL

Serial.begin(rate)
Serial.println(data)

APENDICE

digital output
digital input
high current output
pwm output
potentiometer input
variable resistor input
servo output

Estructura

La estructura básica del lenguaje de programación de Arduino es bastante simple y corre en al menos dos partes. Estas dos piezas necesarias, o las funciones, los bloques para adjuntar las declaraciones.

```
void setup ()  
{  
  declaraciones;  
}
```

```
void loop ()
{
  declaraciones;
}
```

En `void setup ()` es la preparación, `void loop ()` es la ejecución. Ambas funciones son necesarias para el programa de trabajo.

La función de configuración deben seguir la declaración de las variables al principio del programa. Es la primera función para ejecutar en el programa, se ejecuta sólo una vez, y se utiliza para establecer `pinMode` o inicializar la comunicación serial.

La función de bucle sigue a continuación e incluye el código que se ejecutará de forma continua - lectura insumos, provocando salidas, etc Esta función es el núcleo de todos los programas de Arduino.

setup ()

El comando `void setup ()` es llamado una vez al iniciar el programa. Se utiliza para inicializar los modos de PIN, o empezar de serie. Debe incluirse en un programa, incluso si no hay declaraciones a correr.

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

loop ()

Después de llamar a `void setup()` la función `void loop ()` hace precisamente lo que sugiere su nombre, ejecuta los bucles de forma consecutiva, permitiendo que el programa responda a los cambios, y el control de la placa Arduino.

```
void loop ()
{
  digitalWrite (pin, HIGH); // se vuelve 'pin' on

  delay (1000);             // pausa por un segundo
  digitalWrite (pin, LOW);  // se vuelve 'pin' off
}
```

```
delay (1000);           // hace una pausa por un segundo
}
```

funciones

Una función es un bloque de código que tiene un nombre y un bloque de instrucciones que se ejecutan cuando se llama a la función. La configuración de las funciones void setup () y void loop () ya se han discutido y otras funciones incorporadas se discutirá más adelante.

Las funciones habituales pueden ser escritas para realizar tareas repetitivas y reducir el desorden en un programa. Las funciones son declaradas primero el tipo de variable de la función. Este es el tipo de valor a de ser devuelto por la función como "int" para una función de tipo entero. Si no hay ningún valor se va a devolver el tipo de función sería nula. Después de tipo, declarar el nombre dado a la función y entre paréntesis los parámetros que se pasan a la función.

```
type functionName (parámetros)
{
  declaraciones;
}
```

El siguiente tipo `int delayVal function ()` se utiliza para fijar un valor de retraso en un programa de lectura del valor de un potenciómetro. En primer lugar, declara una variable local `v`, `v` Establece el valor del potenciómetro que le da un número entre 0-1023, entonces ese valor se divide por 4 para un valor final entre 0-255, y finalmente regresa de nuevo a que el valor del programa principal .

```
int delayVal ()
{
  int v;           // v Crear variable temporal
  v = analogRead (pot); // leer el valor del potenciómetro
  v /= 4;         // convierte 0-1023 a 0-255
  return v;       // valor de retorno final
}
```

{ } Llaves

Llaves definen el comienzo y el final de los bloques de función y bloques de sentencias, como el void loop () y {} para las declaraciones.

```
type function ()
{
  declaraciones;
}
```

Una llave de apertura (siempre debe ser seguido por una llave de cierre). Esto se refiere a menudo como las llaves de ser equilibrado. Llaves no balanceado puede conducir a menudo crípticos, los errores del compilador impenetrable, que a veces puede ser difícil de rastrear en un programa de gran tamaño.

El medio ambiente Arduino incluye una práctica función para comprobar el saldo de llaves. Sólo tiene que seleccionar un par, o incluso, haga clic en el punto de inserción inmediatamente después de un par, y su compañero lógica será resaltada.

; Punto y coma

Un punto y coma se utiliza para poner fin a una declaración y los distintos elementos del programa. Un punto y coma se utiliza también para los distintos elementos en un bucle for.

```
int x = 13 ;           // declara la variable 'x' como el entero 13
```

Nota: El olvido de poner fin a una línea en un punto y coma dará lugar a un error del compilador. El texto de error puede ser obvia, y se refieren a un punto y coma que falta, o puede que no. Si un error del compilador impenetrable o aparentemente lógicos surge, una de las primeras cosas que hay que comprobar es un punto y coma que falta, cerca de la línea donde el compilador se quejó.

/*...*/ Comentarios en bloque

Los comentarios de bloque son de varios comentarios en línea, son áreas de texto ignorados por el programa y se utilizan para las descripciones de texto grandes de código o comentarios que ayuden a entender otras partes del programa. Ellos empiezan con /* y terminan con */, y puede abarcar varias líneas.

/* Este es un bloque cerrado comentario no olvide el comentario de cierre -
que tienen que ser equilibradas!

```
*/
```

Dado que los comentarios son ignorados por el programa y no tienen espacio en la memoria que se debe utilizar con generosidad y también se puede utilizar para "comentar" bloques de código para propósitos de depuración.

Nota: Si bien es posible incluir comentarios en línea único dentro de un bloque de comentario, adjuntando un comentario segundo bloque no está permitido.

// Comentarios en línea

Comentarios de una sola línea en empezar con // y terminar con la siguiente línea de código. Como comentarios en bloque, son ignorados por el programa y no tienen espacio en la memoria.

```
// Esto es un comentario de una sola línea
```

Comentarios de línea individual se utilizan a menudo después de una declaración válida para proporcionar más información acerca de lo que lleva a cabo la declaración o para proporcionar un recordatorio en el futuro.

variables

Una variable es una manera de nombrar y guardar un valor para su uso posterior por el programa. Como su nombre lo indica, las variables son valores que puede ser cambiado continuamente frente a las constantes cuyo valor no cambia nunca. Una variable debe ser declarado y, opcionalmente, el valor asignado a la necesidad de ser almacenado. El siguiente código declara una variable llamada `inputVariable` y luego le asigna el valor obtenido en la clavija de entrada analógica 2:

```
int inputVariable = 0; // declara una variable y
                      // Asigna el valor 0
inputVariable = analogRead (2); // conjunto de variables y el valor de
                               // Pin analógico 2
```

'`inputVariable`' es la variable. La primera línea declara que contenga un `int`, abreviatura de número entero. La segunda línea establece la variable con el valor en la terminal analógico 2. Esto hace que el valor de los pines 2 accesibles en otras partes del código.

Una vez que una variable se le ha asignado, o re-asignado, usted puede probar su valor a ver si cumple con ciertas condiciones, o usted puede utilizar su valor directamente. Como ejemplo para ilustrar tres operaciones útiles con las variables, el siguiente código comprueba si el `inputVariable` es inferior a 100, si es cierto que

asigna el valor 100 a inputVariable y, a continuación establece un retraso sobre la base de inputVariable que ahora es de un mínimo de 100:

```
if (inputVariable <100) // Pruebas de variable si menos de 100
{
inputVariable = 100; // si es cierto asigna valor de 100
}
delay (inputVariable); // utiliza como variable de retraso
```

Nota: Las variables deben tener nombres descriptivos, para hacer el código más legible. Los nombres de variables como tiltSensor o pulsador sirven de ayuda a los programadores y cualquier otra persona la lea el código para entender lo que representa la variable. Los nombres de variables como var o valor, por el contrario, hacen poco para hacer el código legible y sólo se utiliza aquí como ejemplos. Una variable puede ser nombrado con cualquier palabra que no es ya una de las palabras clave en el idioma Arduino.

declaración de variables

Todas las variables tienen que ser declaradas antes de que puedan ser utilizados. Declarar una variable mediante la definición de su tipo de valor, como en int, long, float, etc, el establecimiento de un nombre específico y, opcionalmente, asignar un valor inicial. Esto sólo debe hacerse una vez en un programa, pero el valor puede ser cambiado en cualquier momento usando la aritmética y diversas tareas. El ejemplo siguiente declara que inputVariable es un int, o de tipo entero, y que su valor inicial es igual a cero. Esto se llama una tarea simple.

```
int inputVariable = 0;
```

Una variable puede ser declarada en una serie de ubicaciones en todo el programa y en esta definición se lleva a cabo determina qué partes del programa puede utilizar la variable.

Alcance de variables

Una variable se puede declarar en el comienzo del programa antes de void setup (), a nivel local dentro de las funciones, y, a veces dentro de un bloque de instrucciones como para los bucles. Cuando se declara la variable determina el alcance variable, o la capacidad de ciertas partes de un programa para hacer uso de la variable.

Una variable global es aquella que puede ser vista y utilizada por cada función y la

declaración en un programa. Esta variable se declara en el comienzo del programa, antes de la función void setup()

Una variable local es aquella que se define dentro de una función o como parte de un bucle for. Sólo es visible y sólo se puede utilizar dentro de la función en la que se hayan declarado. Por tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que contienen valores diferentes. Asegurarse de que sólo una función tiene acceso a las variables del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo muestra cómo declarar unos cuantos tipos diferentes de variables y demuestra la visibilidad de cada variable:

```
int value;          // 'valor' es visible Para cualquier función
void setup ()
{
  // No necesita instalación
}
void loop ()
{
  for (int i = 0; i < 20;) // 'i' sólo es visible
  {
    // Dentro del bucle for
    i ++;
  }
  float f;         // 'f' sólo es visible
}                  // Dentro del bucle loop
```

byte

Byte almacena un valor de 8 bits numérica sin decimales. Tienen un rango de 0-255.

```
byte someVariable = 180; // declara 'someVariable'
                        // Como un tipo de un byte
```

int

Los enteros son el tipo de datos primarios para el almacenamiento de números, sin decimales y almacenar un valor de 16-bits con un rango de -32.768 a 32.767.

```
int someVariable = 1500 // declara 'someVariable'
                       // Como un tipo entero
```

Nota: Las variables de tipo entero se acumularán si es forzado más allá de sus valores máximos y mínimos de una cesión o de comparación. Por ejemplo, si $x = 32767$ y una declaración posterior agrega 1 x , $x = x + 1$ o $++x$, x luego de vuelco y la igualdad de 32.768.

long

Tipo de datos de tamaño extendido para los números enteros de largo, sin decimales, almacenada en un valor de 32 bits con un rango de -2147483648 a 2147483647.

```
long someVariable = 90000; // declara 'someVariable'  
                        // Como un tipo largo
```

float

Un tipo de datos de números de punto flotante o números que tienen un punto decimal. Números de punto flotante tienen mayor resolución que los enteros y se almacenan como un valor de 32 bits con una gama de $3.4028235E +38$ a $-3.4028235E +38$.

```
float someVariable = 3.14; // declara 'someVariable'  
                          // Como un tipo de punto flotante
```

Nota: Los números de coma flotante no son exactos, y puede producir resultados extraños en comparación. Aritmética de punto flotante es también mucho más lento que el de matemáticas de enteros en realizar los cálculos, por lo que debe evitarse si es posible.

Array

Una matriz es un conjunto de valores que se accede con un número índice. Cualquier valor en la matriz puede ser llamado llamando al nombre de la matriz y el número de índice del valor. Las matrices son iguales a cero el índice, con el primer valor de la matriz que comienza en el número de índice 0. Una matriz debe ser declarado y, opcionalmente, asignar valores antes de que puedan ser utilizados.

```
int myArray [] = {value0, valor1, valor2 ...}
```

Del mismo modo, es posible declarar una matriz, al declarar el tipo de matriz y el tamaño y luego asignar valores a una posición de índice:

```
int myArray [5]; // declara matriz de enteros w / 6 posiciones  
myArray [3] = 10; // asigna el 4º índice valor 10
```

Para recuperar un valor de una matriz, asignar una variable a la matriz y la posición de índice:

```
x = myArray [3]; // x ahora es igual a 10
```

Las matrices se utilizan a menudo en los bucles, donde la lucha contra el incremento también se utiliza como la posición de índice para cada valor de la matriz. El siguiente ejemplo utiliza una matriz a parpadear un LED. Uso de un bucle, el contador empieza en 0, escribe el valor que figura en la posición de índice 0 en el parpadeo de matriz [], en este caso 180, el pin PWM 10, hace una pausa para 200 ms, a continuación, pasa a la siguiente posición de índice.

```

int ledPin = 10;          // led en el pin 10
byte flicker[] = {180, 30, 255, 200, 10, 90, 150, 60};
                        // Por encima de matriz, de 8 de
void setup ()           // valores diferentes
{
pinMode (ledPin, OUTPUT); // establece pin de salida
}
void loop ()
{
for (int i = 0; i < 7; i++) // bucle es igual a número
{
                        // De los valores en la matriz de
analogWrite (ledPin, flicker [i]); // escribir el valor de índice
delay (200);           // pausa de 200 ms
}
}

```

Aritmética

Los operadores aritméticos son la suma, resta, multiplicación y división. Que devolver la suma, diferencia, producto, o cociente (respectivamente) de dos operandos.

```

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

```

La operación se lleva a cabo utilizando el tipo de datos de los operandos, por lo que, por ejemplo, $9 / 4$ resultados en 2 en lugar de 2,25 desde el 9 y 4 son enteros, y son incapaces de usar decimales. Esto también significa que la operación puede desbordar si el resultado es mayor que lo que puede ser almacenado en el tipo de datos.

Si los operandos son de tipos diferentes, el tipo más grande se utiliza para el cálculo. Por ejemplo, si uno de los números (operandos) son del tipo float y el otro de tipo entero, las matemáticas de punto flotante se utilizará para el cálculo.

Elija tamaño variable que son lo suficientemente grandes como para mantener el mayor resultado de sus cálculos. Saber en qué punto de su variable de vuelco y también lo que sucede en la otra dirección, por ejemplo $(0 - 1)$ O $(0 - - 32768)$.

Para matemáticas que requiere fracciones, las variables float uso, pero sea consciente de sus desventajas: gran tamaño y baja velocidad del cálculo.

Nota: Utilice el operador de conversión por ejemplo, `(int) myFloat` para convertir un tipo variable a otro sobre la marcha. Por ejemplo, `i = (int) 3,6` establecerá `i` igual a 3.

asignaciones compuestas

Asignaciones compuestas combinar una operación aritmética con una asignación de variables. Estos se encuentran comúnmente en los bucles como se describe más adelante. Las asignaciones de compuestos más comunes incluyen:

```
x ++      // igual que x = x + 1, o por incrementos de 1 x
x --      // igual que x = x - 1, o decrementos x por -1
x += y    // igual que x = x + y, o por incrementos x + y
x -= y    // lo mismo que x = x - y, o decrementos x por y
x *= y    // lo mismo que x = x * y, o multiplica x por y
x /= y    // lo mismo que x = x / y, o se divide x por y
```

Nota: Por ejemplo, `x *= 3` triplicaría el valor anterior de X y volver a asignar el valor resultante a x.

Los operadores de comparación

Las comparaciones de una variable o constante en contra de otro son a menudo utilizados en caso de declaraciones para comprobar si una condición especificada es verdadera. En los ejemplos se encuentran en las páginas siguientes, se utiliza para indicar cualquiera de las siguientes condiciones:

```
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 a Y
x >= y   // x es mayor que o igual a Y
```

operadores lógicos

Los operadores lógicos son generalmente una manera de comparar dos expresiones y devuelven un verdadero o falso dependiendo del operador. Existen tres operadores lógicos AND, OR y NOT, que se utilizan a menudo en las sentencias if

Y lógico:

```
if (x0 && x < 5)      // cierto sólo si ambas
                      // Expresiones son verdaderas
```

O lógico:

```
if (x > 0 || y > 0) //verdadero si cualquier
```

```
                // Expresión es verdadera
NO lógico:
if (!x > 0)      // sólo si es cierto
                // Expresión es falsa
```

Constantes

El lenguaje Arduino tiene algunos valores predefinidos, que se llaman constantes. Se utilizan para hacer los programas más fácil de leer. Las constantes se clasifican en grupos.

TRUE/FALSE //verdadero / falso

Estas son las constantes que definen los niveles de Boole la lógica. FALSO es fácil de definir como 0 (cero), mientras que cierta es a menudo definida como 1, pero también puede ser otra cosa excepto cero. Así, en un sentido de Boole, -1, 2, y -200 son todos también se define como TRUE.

```
if (b == TRUE);
{
  lo a ejecutar;
}
```

HIGH/LOW // encendido / apagado

Estas constantes definir los niveles de pin como alta o baja y se utilizan al leer o escribir a los pines digitales. Alta se define como el nivel lógico 1, ON, o 5 voltios mientras que los bajos es el nivel lógico 0, OFF, o 0 voltios.

```
digitalWrite (13, HIGH);
```

INPUT /OUTPUT //entrada / salida

Constantes se utiliza con el pinMode () para definir el modo de un alfiler digital, ya sea como entrada o salida.

```
pinMode (13, OUTPUT);
```

if

si las declaraciones probar si una determinada condición se ha alcanzado, como un valor analógico por encima de un cierto número, y ejecuta las sentencias dentro de los corchetes, si el enunciado es verdadero. Si es falso el programa pasa por la declaración. El formato de un caso de prueba es:

```
if (Variable ?? valor)
{
o a ejecutar;
}
```

El ejemplo anterior se compara someVariable a otro valor, que puede ser una variable o constante. Si la comparación, o la condición entre paréntesis es verdadera, se ejecutan las sentencias dentro de los corchetes. Si no, el programa pasa por ellos y continúa después de los corchetes.

Nota: Tenga cuidado con el uso accidental '=', como en el caso de (x = 10), mientras que técnicamente válida, define la variable x al valor de 10 y es el resultado siempre es cierto. Utilizar en su lugar '==', como en if (x == 10), que sólo se evalúa si x pasa a ser igual al valor de 10 o no. Piensa en '=' como "iguales" y no "==" ser "es igual a".

if ... else

si ... else permite 'bien-o' que se tomen decisiones. Por ejemplo, si desea probar una entrada digital, y hacer una cosa si la entrada fue ALTA o en lugar de hacer otra cosa si la entrada fue baja, podría escribir que de esta manera:

```
if (inputPin == HIGH)
{
doThingA;
}
else
{
doThingB;
}
```

los demás también puede preceder a otra si la prueba, de manera que múltiples pruebas mutuamente excluyentes se pueden ejecutar al mismo tiempo. Es incluso posible tener un número ilimitado de estas ramas más. Recuerde, sin embargo, sólo un conjunto de estados se llevará a cabo en función de las pruebas de condición:

```
if (inputPin <500)
{
doThingA;
}
else if (inputPin >= 1000)
{
doThingB;
}
```

```
else
{
doThingC;
}
```

Nota: Una declaración de si las pruebas sólo si la condición dentro de los paréntesis es verdadera o falsa. Esta declaración puede ser cualquier sentencia válida de C como en el primer ejemplo, si (inputPin == HIGH). En este ejemplo, la declaración sólo si comprueba si efectivamente la entrada seleccionada es a nivel lógico alto o +5 V.

for

La instrucción for es usado para repetir un bloque de sentencias encerradas en llaves de un número determinado de veces. Un incremento contador se utiliza a menudo para aumentar y terminar el bucle. Hay tres partes, separadas por punto y coma (;), a la cabecera de bucle:

```
for (inicialización; condición; expresión)
{
lo a ejecutar;
}
```

La inicialización de una variable local, o la lucha contra el incremento, que ocurre la primera y única vez. Cada vez que a través del bucle se comprueba con la siguiente condición. Si la condición sigue siendo cierto, las siguientes declaraciones y de expresión se ejecutan y la condición es la prueba de nuevo. Cuando la condición se convierte en falso, el bucle finaliza.

El ejemplo siguiente inicia el entero i en 0, pruebas para ver si sigue siendo menos de 20 y si es cierto, incrementa i en 1 y ejecuta las instrucciones adjunta:

```
for (int i = 0; i <20; i ++) // declara que, si las pruebas menos
{
// De 20, los incrementos de i en 1
digitalWrite (13, HIGH); // convierte pin 13 en
delay (250); // pausa durante 1 / 4 de segundo
digitalWrite (13, LOW); // pin 13 vueltas fuera
delay (250); // pausa durante 1 / 4 de segundo
}
```

Nota: El C de bucle es mucho más flexible que para los bucles se encuentran en algunos otros lenguajes de programación, incluyendo BASIC. Cualquiera o todos

de los tres elementos de encabezado se puede omitir, si bien se requiere el punto y coma. También las declaraciones de la inicialización, la condición y de expresión pueden ser válidas las declaraciones C con variables independientes. Estos tipos de raro que los estados pueden ofrecer soluciones a algunos problemas de programación rara.

while

mientras que los bucles se repetirá continuamente, y el infinito, hasta que la expresión dentro del paréntesis se convierte en falsa. Algo debe cambiar la variable de prueba, o el bucle mientras que nunca se cerrará. Esto podría ser en su código, como una variable se incrementa, o una condición externa, como las pruebas de un sensor.

```
while (someVariable ?? valor)
{
doSomething;
}
```

El ejemplo siguiente prueba si "someVariable 'es menor que 200 y si es cierto ejecuta las sentencias dentro de los corchetes, y continuará hasta el bucle' someVariable 'ya no es inferior a 200.

```
while (someVariable <200) // Pruebas de si es menos de 200
{
doSomething;           // ejecuta adjunta declaraciones
someVariable ++;      // incrementos variables del 1 de
}
```

do ... while

El ciclo do es un ciclo conducido fondo que trabaja en la misma forma que el bucle while, con la excepción de que la condición se prueba al final del bucle, de modo que el bucle no se ejecutará siempre al menos una vez.

```
do
{
doSomething;
} While (someVariable ?? valor);
```

El ejemplo siguiente asigna readSensors () a la variable 'x', hace una pausa de 50 milisegundos, a continuación, recorre los bucles de forma indefinida hasta que 'x' ya no es inferior a 100:

```

do
{
x = readSensors ();          // asigna el valor de
                             // ReadSensors () para x
delay (50);                  // hace una pausa de 50 milisegundos
}
While (x <100);              // recorre bucles hasta que x es menor de 100

```

pinMode (pin, mode)

Utilizado en void setup () para configurar un pin especificado para comportarse bien como una entrada o una salida.

```
pinMode (pin, OUTPUT);      // pin conjuntos a la salida
```

Arduino digital alfileres por defecto a los insumos, por lo que no es necesario declarar explícitamente como insumos con pinMode (). Pins configurado como ENTRADA se dice que están en un estado de alta impedancia. También hay resistencias de 20KΩ conveniente pullup integrado en el chip atmega que se puede acceder desde el software. Incorporados en estas resistencias pullup se accede de la siguiente manera:

```
pinMode (pin, INPUT);      // Set 'pin' a la entrada de
digitalWrite (pin, HIGH);  // activar resistencias pullup
```

Resistencias pullup normalmente se utiliza para la conexión de los insumos, como los interruptores. Observe en el ejemplo anterior no convierte a un pin de salida, es simplemente un método para la activación de la fuerza interna de empresas. Pins configurado como SALIDA se dice que están en un estado de baja impedancia y puede proporcionar 40 mA (miliamperios) de la corriente a otros dispositivos / circuitos. Esta es la corriente suficiente para un brillante luz LED (no se olvide de la resistencia en serie), pero no suficiente corriente para ejecutar la mayoría de los relés, solenoides, o motores. Los cortocircuitos en las patillas Arduino y el exceso de corriente puede dañar o destruir el pin de salida, o dañar el chip atmega entero. A menudo es una buena idea para conectar un pin de salida a un dispositivo externo en serie con una resistencia de 470Ω o 1KΩ.

digitalRead (pin)

Lee el valor de un pin digital especificado con el resultado de alta o baja. El pin se puede especificar ya sea como una variable o constante (0-13).

```
value = digitalRead (pin); // establece el valor "igual a
```

// El pin de entrada de

digitalWrite (pin,value)

Productos de alto nivel ya sea lógica o BAJA al (se enciende o apaga) un pin digital especificado. El pin se puede especificar ya sea como una variable o constante (0-13).

```
digitalWrite (pin, HIGH); // pin conjuntos "al más alto
```

El siguiente ejemplo lee un pulsador conectado a una entrada digital y se enciende un LED conectado a una salida digital cuando el botón se ha pulsado:

```
int led = 13; // connect LED a pin 13
int pin = 7; // connect boton a pin 7
int value = 0; // variable to store the read value
void setup()
{
  pinMode(led, OUTPUT); // configura pin 13 como salida
  pinMode(pin, INPUT); // configura pin 7 como entrada
}
void loop()
{
  value = digitalRead(pin); // valor establece "igual a

                          // El pin de entrada de

  digitalWrite(led, value); // configure el led
}                          //y valor de los botones
```

analogRead (pin)

Lee el valor de un pin analógico se especifica con una resolución de 10 bits. Esta función sólo funciona en la analógica en las patillas (0-5). Los valores entero resultante de 0 a 1023.

```
value = analogRead (pin); // establece el valor ', ' igual a 'pin'
```

Nota: los pernos análogos a diferencia de los digitales, no necesitan ser declaradas como primera entrada ni SALIDA.

analogWrite (pin, valor)

Escribe un pseudo-valor analógico utilizando hardware activado de modulación por ancho de pulso (PWM) a un pin de salida marcada PWM. En los nuevos Arduinos con el chip ATmega168, esta función funciona en los pines 3, 5, 6, 9, 10 y 11.

Mayores Arduinos con un sólo apoyo ATmega8 pines 9, 10 y 11. El valor puede especificarse como una variable o constante con un valor de 0-255.

```
analogWrite (pin, valor); // escribe "valor" al pin analógico '
```

Un valor de 0 genera un constante 0 voltios de salida en el pin especificado; un valor de 255 genera una constante de 5 voltios de salida en el pin especificado. Para los valores de entre 0 y 255, el pin rápidamente alterna entre 0 y 5 voltios - cuanto mayor sea el valor, más a menudo el pin es alto (5 voltios). Por ejemplo, un valor de 64 será de 0 voltios, tres cuartas partes del tiempo, y 5 voltios cuarta parte del tiempo, un valor de 128 será a 0 la mitad del tiempo y 255 la mitad del tiempo, y un valor de 192 se ser 0 voltios cuarta parte del tiempo y de 5 voltios de tres cuartas partes del tiempo.

Debido a que esta es una función del hardware, el pin que generará una ola constante después de una llamada a analogWrite en el fondo hasta la siguiente llamada a analogWrite (o una llamada a digitalWrite o en el mismo PIN).

Nota: los pin análogos a diferencia de los digitales, no necesitan ser declaradas como primera entrada ni SALIDA.

El siguiente ejemplo lee un valor analógico de un pin de entrada analógica, convierte el valor de dividir por 4, y emite una señal PWM en un pin PWM:

```
int led = 10; // LED con resistencia de 220 en el pin 10
int pin = 0; // potenciómetro analógico 0 en el pin
int value; // valor de la lectura
void setup (){} // no necesita instalación
void loop ()
{
value = analogRead (pin); // establece el valor 'value' igual a 'pin'
value /= 4; // convierte 0-1023 de 0-255
analogWrite (led, value); // salidas de señal PWM a LED
}
```

delay (ms)

Detiene un programa para la cantidad de tiempo como se especifica en milisegundos, en 1000 es igual a 1 segundo.

```
delay (1000); // espera por un segundo
```

millis ()

Devuelve el número de milisegundos desde la placa Arduino empezó a correr el programa actual como un valor de largo sin signo.

```
value = millis (); // establece el valor "igual a millis ()
```

Nota: Este número de desbordamiento (reiniciado de nuevo a cero), después de aproximadamente 9 horas.

min (x, y)

Calcula el mínimo de dos números de cualquier tipo de datos y devuelve el número más pequeño.

```
value = min (value, 100); // establece el valor "a la menor de  
// 'Valor' o 100, asegurándose de que  
// Nunca se pone por encima de 100.
```

max (x, y)

Calcula el máximo de dos números de cualquier tipo de datos y devuelve el número mayor.

```
value = max (value, 100); // establece el valor "a la mayor de  
// 'Valor' o 100, asegurándose de que  
// Es por lo menos 100.
```

randomSeed (seed) //semilla aleatoria

Establece un valor, o de las semillas, como punto de partida para la función random ().

```
randomSeed (valor); // establece el valor "como la semilla aleatoria
```

Debido a que el Arduino es incapaz de crear un número verdaderamente aleatorio, randomSeed le permite colocar una función variable, constante o en la función aleatoria, lo que ayuda a generar más al azar "aleatorio" números. Hay una variedad de semillas diferentes, o funciones, que pueden ser utilizados en esta función, incluyendo millis () o incluso analogRead () para leer el ruido eléctrico a través de un pin analógico.

random (máx.)**random(min, max)**

La función aleatoria le permite devolver números pseudo-aleatorios dentro de un rango especificado por los valores mínimo y máximo.

```
value = random (100, 200); // establece el valor "de una muestra aleatoria  
// Número entre 100-200
```

Nota: Utilice este después de usar el randomSeed () la función.

El ejemplo siguiente crea un valor aleatorio entre 0-255 y entrega una señal PWM en un pin PWM igual al valor aleatorio:

```
int randomNumber;           // variable para almacenar el valor aleatorio
int led = 10;              // led con Resistencia 220 om sobre pin 10
void setup () {}          // no necesita instalación
void loop ()
{
  randomSeed (millis ());  // establece millis () como semillas
  randomNumber = random (255); // número aleatorio 0-255
  analogWrite (led, randomNumber); // salidas de señal PWM
  delay (500);            // pausa durante medio segundo
}
```

Serial.begin (rate)

Abre el puerto de serie y conjuntos de la velocidad de transmisión para la transmisión de datos en serie. La velocidad de transmisión típica para comunicarse con el ordenador es de 9600, aunque se admiten otras velocidades.

```
void setup ()
{
  Serial.begin (9600); // abre el puerto serie
}                       // Tipo de conjuntos de datos a 9600 bps
```

Nota: Cuando se utiliza la comunicación serial, pins digital 0 (RX) y 1 (TX) no pueden utilizarse al mismo tiempo.

Serial.println (data)

Imprime los datos al puerto serie, seguido por un retorno de carro automático y avance de línea. Este comando tiene la misma forma que Serial.print (), pero es más fácil para la lectura de datos en el monitor de serie.

```
Serial.println (analogValue); // envía el valor de
                               // 'AnalogValue'
```

Nota: Para más información sobre las varias permutaciones de la Serial.println () y Serial.print () funciones, por favor vaya al sitio web de Arduino.

El ejemplo siguiente toma una lectura de pin0 analógico y envía estos datos a la computadora cada 1 segundo.

```
void setup ()
```

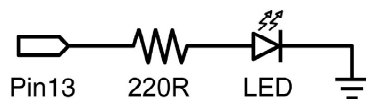
```

{
Serial.begin (9600);           // juegos de serie a 9600bps
}
void loop ()
{
Serial.println (analogRead (0)) // envía valor analógico
delay (1000);                 // pausa durante 1 segundo
}

```

digital output

salida digital



Esta es la base 'hola mundo' programa utilizado para convertir algo simplemente encendido o apagado. En este ejemplo, un LED está conectado a pin13, y está parpadeando cada segundo. La resistencia se puede omitir en este pin desde el Arduino tiene una in construido

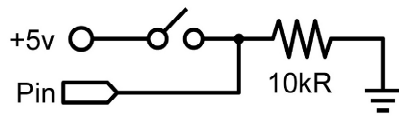
```

int ledPin = 13;           // LED en el pin digital 13
void setup ()              // ejecutar una vez
{
pinMode (ledPin, OUTPUT); // sets de 13, como de salida
}
void loop ()               // ejecutar una y otra vez
{
digitalWrite (ledPin, HIGH); // convierte el LED
delay (1000);               // pausa durante 1 segundo
digitalWrite (ledPin, LOW) // convierte el LED apagado
delay (1000);              // pausa durante 1 segundo
}

```

digital input

entrada digital



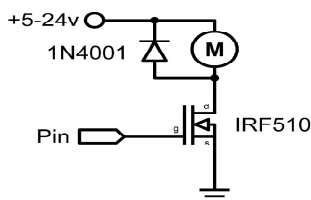
Esta es la forma más simple de entrada con sólo dos estados posibles: encendido o apagado. En este ejemplo se lee un simple interruptor o pulsador conectado a PIN2. Cuando el interruptor está cerrado el pin de entrada a leer ALTA y encender un LED.

```
int ledPin = 13;           // pin de salida para el LED de
int inPin = 2;            // pin de entrada (para un interruptor)
void setup ()
{
  pinMode (ledPin, OUTPUT); // declara LED como salida
  pinMode (inPin, INPUT);   // declare como entrada de interruptor de
}

void loop ()
{
  if (digitalRead (inPin) == HIGH) // comprueba si la entrada es ALTA
  {
    digitalWrite (ledPin, HIGH); // convierte el LED encendido
    delay (1000);                // pausa durante 1 segundo
    digitalWrite (ledPin, LOW);  // convierte el LED apagado
    delay (1000);                // pausa durante 1 segundo
  }
}
```

high current output

salida de corriente de alta



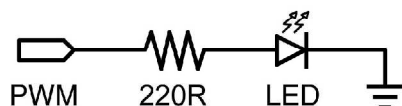
A veces es necesario controlar más de 40 mA de la Arduino. En este caso un

transistor MOSFET o se podría utilizar para conmutar cargas de corriente más elevada. El siguiente ejemplo rápidamente se enciende y apaga el MOSFET de 5 veces cada segundo.

Nota: El esquema muestra un motor y un diodo de protección, pero otras cargas inductivas puede ser utilizado sin el diodo.

```
int outPin = 5;           // pin de salida para el IC
void setup ()
{
  pinMode (outPin, OUTPUT); // establece pin5 como salida
}
void loop ()
{
  for (int i = 0; i <= 5; i ++) // loops 5 veces
  {
    digitalWrite (outPin, HIGH); // RES se convierte en
    delay (250);                 // pausas 1 / 4 de segundo
    digitalWrite (outPin, LOW); // convierte IC fuera
    delay (250);                 // pausas 1 / 4 de segundo
  }
  delay (1000);               // se detiene 1 segundo
}
```

pwm output



Modulación de amplitud de pulso (PWM) es una forma de salida analógica por fingir un pulso a la salida. Esto podría servir para atenuar e iluminar un LED o posterior para controlar un servomotor. El siguiente ejemplo lentamente ilumina y se oscurece un LED utilizando para bucles.

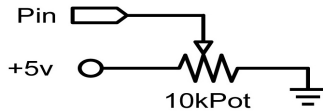
```
int ledPin = 9;           // pin PWM para el LED
void setup () {}         // no necesita instalación
void loop ()
{
  for (int i = 0; i <= 255; i ++) // ascendente valor para i
  {
    analogWrite (ledPin, i); // Establece el nivel de brillo a i
    delay (100);           // pausa durante 100 ms
  }
  for (int i = 255; i >= 0; i --) // descendente de valor para i
  {
```

```

analogWrite (ledPin, i); // Establece el nivel de brightness a i
delay (100);           // pausa durante 100 ms
}
}

```

potentiometer input



Uso de un potenciómetro y un análogo de la Arduino a la conversión digital (ADC) Pins es posible leer los valores analógicos 0-1024. El siguiente ejemplo utiliza un potenciómetro para controlar un tipo de LED's de parpadeo.

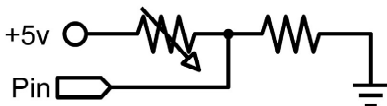
```

int potPin = 0;           // pin de entrada para el potenciómetro de
int ledPin = 13;         // pin de salida para el LED de
void setup ()
{
  pinMode (ledPin, OUTPUT); // declara ledPin como SALIDA
}
void loop ()
{
  digitalWrite (ledPin, HIGH); // convierte en ledPin
  delay(analogRead (potPin)); // Programa de pausa
  digitalWrite (ledPin, LOW) // convierte ledPin fuera
  delay (analogRead (potPin)); // Programa de pausa
}

```

variable resistor input

entrada de resistencia variable



Resistencias variables incluyen sensores de luz CdS, termistores, sensores de flexión, y así sucesivamente. Este ejemplo hace uso de una función para leer el valor analógico y establecer un tiempo de retardo. Esto controla la velocidad a la que un LED se ilumina y se oscurece.

```

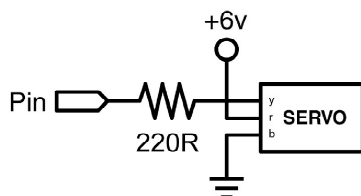
int ledPin = 9; // pin PWM para el LED
int analogPin = 0; // resistencia variable en el pin analógicas 0
void setup () {} // no necesita instalación

void loop ()
{
for (int i = 0; i <= 255; i ++)// ascendente valor para i
{
analogWrite (ledPin, i); // Establece el nivel de brillo a i
delay (delayVal ()); // obtiene el valor del tiempo y las pausas
}
for (int i = 255; i >= 0; i --) // descendente de valor para i
{
analogWrite (ledPin, i); // Establece el nivel de brightness a i
delay(delayVal ()); // obtiene el valor del tiempo y las pausas
}
}
int delayVal ()
{
int v; // Crear la variable temporal
v = analogRead (analogPin); // leer el valor analógico
v /= 8; // convertir 0-1024 a 0-128
return v ; // devuelve el valor final
}

```

servo output

salida de servo



Hobby servos son un tipo de auto-contenido de motor que puede mover en un giro de 180 ° de arco. Todo lo que se necesita es enviar un pulso cada 20 ms. En este

ejemplo se utiliza una función de servoPulse para mover el servo de 10 ° -170 ° y viceversa.

```
int servoPin = 2;      // servo conectado a la clavija digital 2
int myAngle           // ángulo de la servo aproximadamente 0-180
int pulseWidth        // variable de función servoPulse
void setup ()
{
pinMode (servoPin, OUTPUT); // establece el pin 2 como salida
}

void servoPulse (int servoPin, int myAngle)
{
pulseWidth = (myAngle * 10) + 600    // determina la demora
digitalWrite (servoPin, HIGH);       // set servo de alta
delayMicroseconds (pulseWidth);     // pausa de microsegundo
digitalWrite (servoPin, LOW);        // set servo de baja
}
void loop ()
{
// Servo comienza a los 10 grados y gira a
170 grados
for (myAngle = 10; myAngle <= 170; myAngle + +)
{
servoPulse (servoPin, myAngle); // enviar pin y el ángulo
delay (20)                       // ciclo de actualización
}
// Empieza servo a 170 grados y gira a 10
grados
for (myAngle = 170; myAngle >= 10; myAngle --)
{
servoPulse (servoPin, myAngle); // enviar pin y el ángulo
delay (20) ;                     // ciclo de actualización
}
}
```

