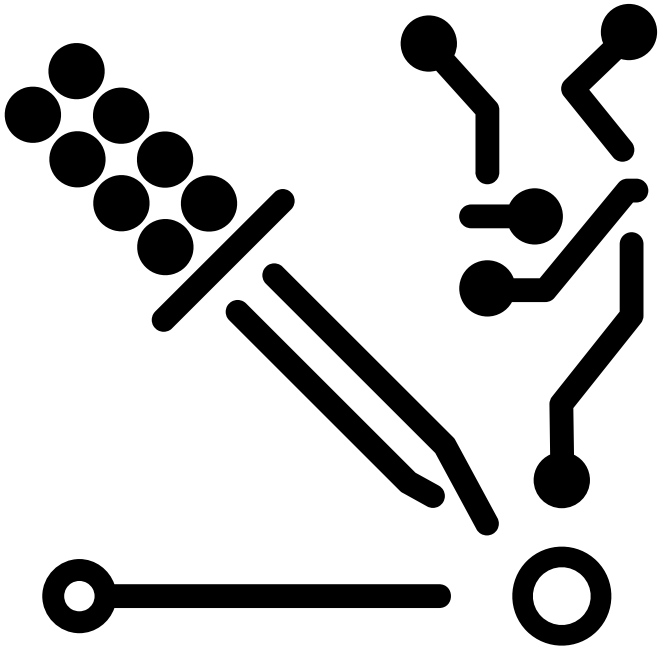
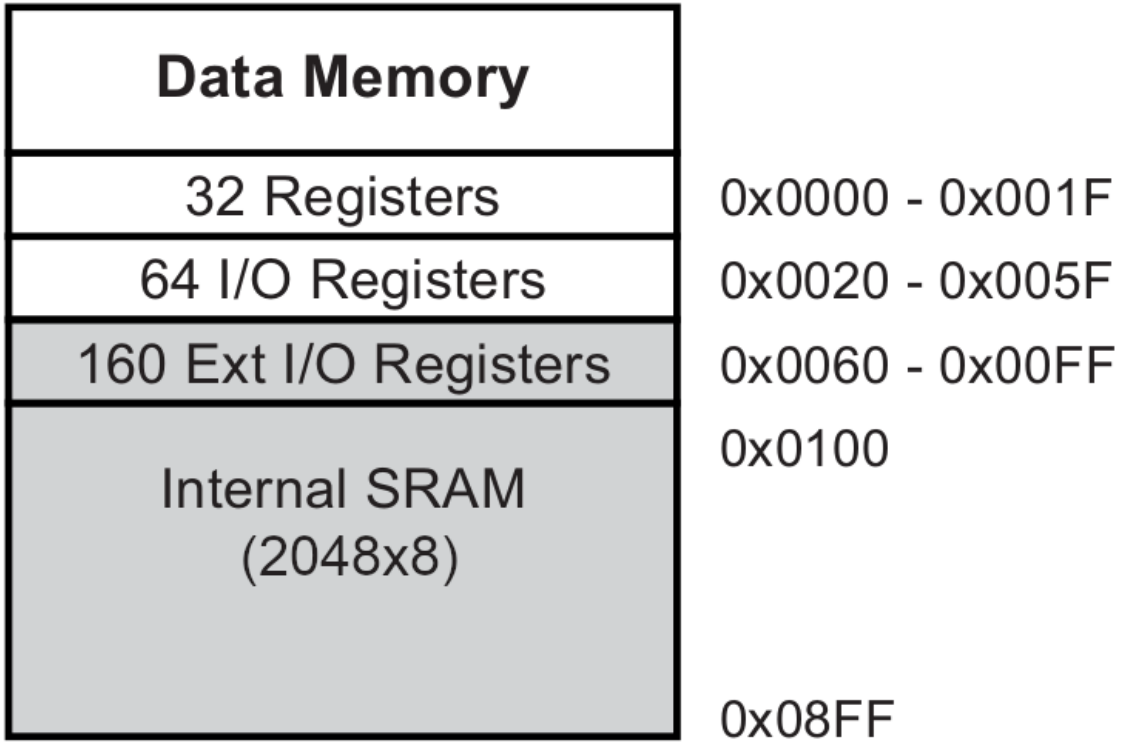


TALLER DE ROBÓTICA CON ARDUINO

MAPA DE MEMORIA



VARIABLES

Una variable es una forma de **nombrar y almacenar un valor** para su uso posterior por parte del programa, como los datos de un sensor o un valor intermedio utilizado en un cálculo.
Antes de que se utilicen, se deben **declarar** todas las variables. Declarar una variable significa definir su tipo y, opcionalmente, establecer un valor inicial (inicializar la variable). No es necesario inicializar las variables (asignarles un valor) cuando se declaran, pero suele ser útil.

```
int inputVariable1;
int inputVariable2 = 0; // ambos son correctos
```

Los programadores deben considerar el tamaño de los números que desean almacenar al elegir los tipos de variables. Las variables se renovarán cuando el valor almacenado exceda el espacio asignado para almacenarlo.

»»»» Variables

VARIABLES

Otra elección importante a la que se enfrentan los programadores es **dónde declarar** las variables. El lugar específico en el que se declaran las variables influye en cómo varias funciones de un programa verán la variable. A esto se le llama alcance o ámbito de la variable.

- **Ámbito global:** decimos que la variable es global porque se puede acceder a ella desde cualquier parte del programa, es decir, estamos accediendo a su valor desde todas las funciones que lo integran.
- **Ámbito local:** son variables que solo existen dentro del ámbito en el que han sido declaradas. Por entendernos rápidamente, un ámbito es lo que está entre llaves. Si las utilizamos fuera de su ámbito tendremos un error de compilación. Al existir solo dentro de su ámbito el programa podría repetir el mismo nombre de variable en distintos ámbitos.

»»»» scope

VARIABLES GLOBALES

```
int a = 5; // "a" es una variable global

void setup(){

}

void loop(){
    a = a + 1;
}
```

VARIABLES LOCALES

```
void setup(){
    int a = 5;
    // la variable "a" solo existe dentro de la función setup
}

void loop() {
    a = a + 1;
    // al compilar daría un error de compilación
}
```

STATIC

La palabra clave **static** se usa para crear variables que son visibles para una sola función. Sin embargo, a diferencia de las variables locales que se crean y destruyen cada vez que se llama a una función, las variables estáticas persisten más allá de la llamada a la función, conservando sus datos entre las llamadas a la función.
Las variables declaradas como estáticas solo se crearán e inicializarán la primera vez que se llame a una función.

```
if (bot == true) {
    static int var = 0;
    Serial.println(var);
    var++;
}
```

»»»» static

VOLATILE

Declarar una variable volátil es una directiva para el compilador. Específicamente, ordena al compilador que cargue la variable desde la RAM y no desde un registro de almacenamiento, que es una ubicación de memoria temporal donde se almacenan y manipulan las variables del programa. En determinadas condiciones, el valor de una variable almacenada en los registros puede ser inexacto.
Una variable debe declararse volátil siempre que su valor pueda cambiarse por algo más allá del control de la sección de código en la que aparece, como un hilo que se ejecuta simultáneamente. En Arduino, el único lugar donde es probable que esto ocurra es en las secciones de código asociadas con las interrupciones, llamadas rutina de servicio de interrupciones.

»»» volatile

TIPOS DE DATOS

Tipos de Datos	Memoria que ocupa	Rango de valores
boolean	1 byte	0 o 1 (True o False)
byte/ unsigned char	1 byte	0 – 255
char	1 byte	-128 – 127
int	2 bytes	-32768 – 32767
word / unsigned int	2 bytes	0 – 65535
long	4 bytes	-2147483648 – 2147483647
unsigned long	4 bytes	0 - 4294967295
float / double	4 bytes	-3,4028235E+38 - 3,4028235E+38
string	1 byte + x	Array de caracteres
array	1 byte + x	Colección de variables

TIPOS DE DATOS

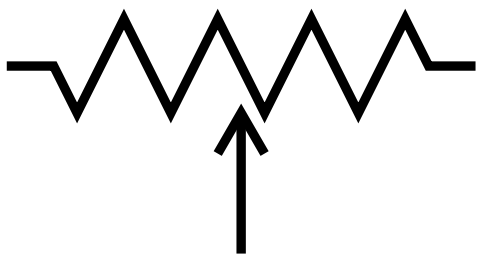
Falla en el cohete **Ariane 5G** debido a que usaba el mismo sistema de navegación inercial que el Ariane 4 pero nadie había tenido en cuenta que la mayor potencia del Ariane 5 iba a hacer que al poco de despegar adquiriera mucha más velocidad horizontal que el 4 y que esto podía afectar al funcionamiento del INS.

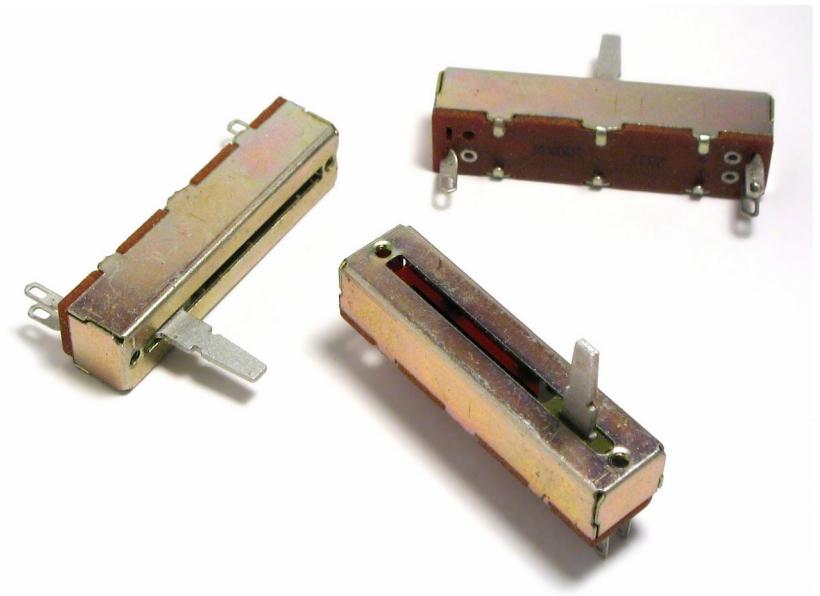
»»»» Veinte años de la explosión del primer Ariane 5 por un fallo de software

TIPOS DE DATOS

POTENCIÓMETRO

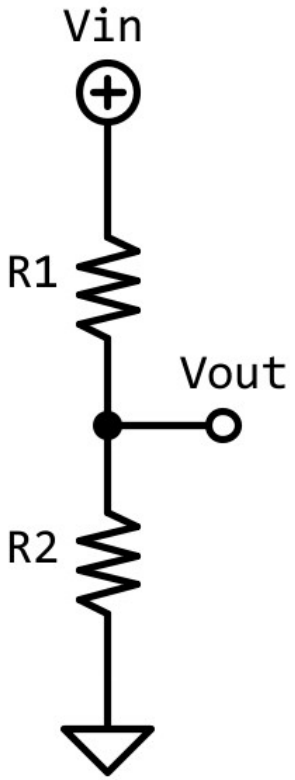
Un **potenciómetro** es uno de los usos que posee la resistencia o resistor variable mecánica (con cursor y de al menos tres terminales). El usuario al manipularlo, obtiene entre el terminal central (cursor) y uno de los extremos una fracción de la diferencia de potencial total, se comporta como un divisor de tensión o voltaje.





POTENCIÓMETRO

Un **divisor de tensión** es una configuración de un circuito eléctrico que reparte la tensión de una fuente entre una o más resistencias conectadas en serie.



$$V_{\text{out}} = \frac{R_2}{R_1 + R_2} \cdot V_{\text{in}}$$

ENTRADA ANALÓGICA

No se **define** en el *setup*.

Se **lee** con la instrucción analogRead(pin) y retorna valores enteros de 0 a 1023 (10 bits). Trabajando con 5V vamos a poder medir 4,88mV por cada paso (5V/1024). Por ejemplo:

```
analogRead(A0);
```

»»»» analogRead()

ENTRADA ANALÓGICA

Con la función **analogReference()** se configura la referencia interna del ADC para obtener un rango acorde a la señal recibida. En placas Arduino AVR (Uno, Mega, Leonardo, etc.) tenemos a disposición:

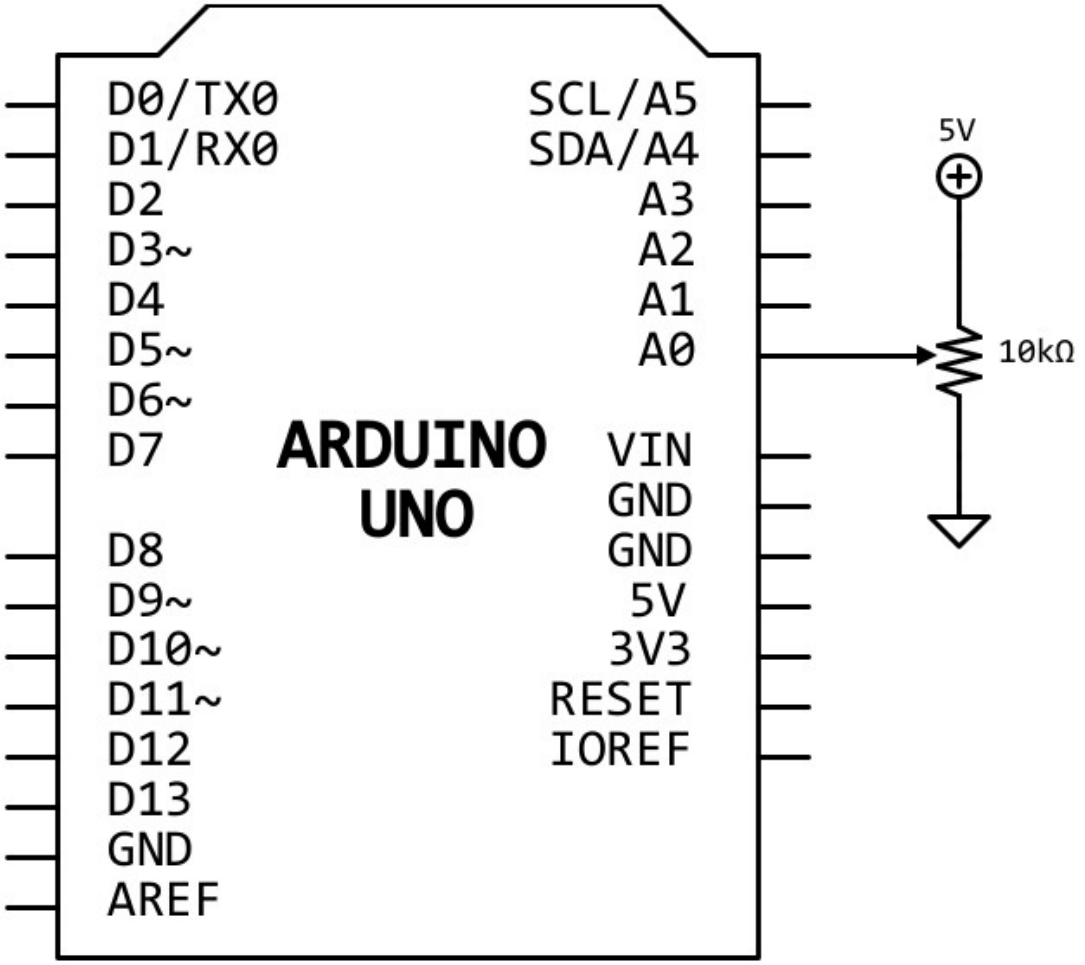
- **DEFAULT:** referencia de 5V o 3.3V dependiendo del tipo de placa.
- **INTERNAL:** referencia interna igual a 1.1V en ATmega168 o ATmega328P, y 2.56V en ATmega32U4 y ATmega8 (no disponible en Arduino Mega).
- **INTERNAL1V1:** referencia interna de 1.1V solo para Arduino Mega.
- **INTERNAL2V56:** referencia interna de 2.56V solo para Arduino Mega.
- **EXTERNAL:** se utilizará como referencia la tensión aplicada al pin AREF (0 a 5V).

»»»» analogReference()

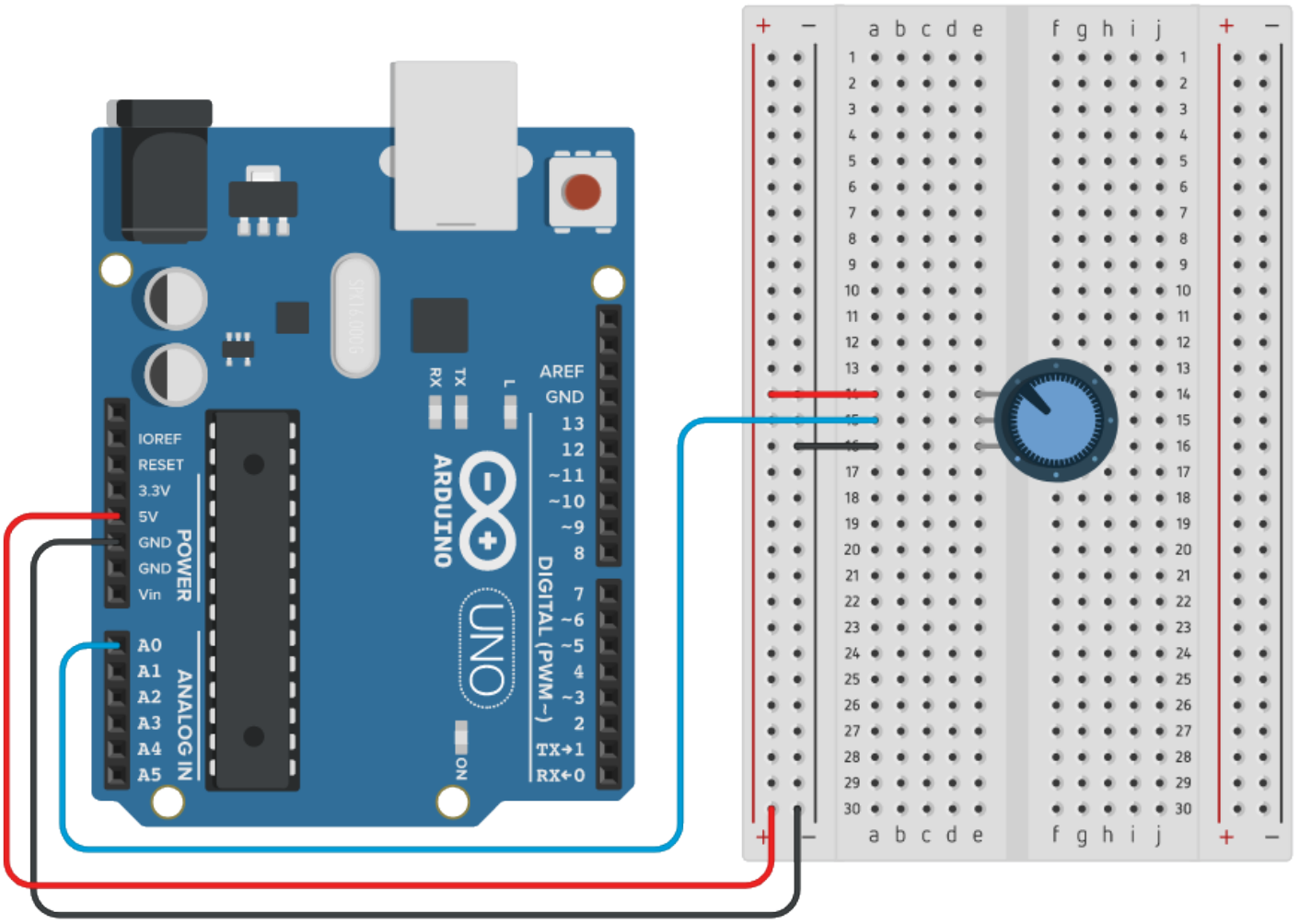
ENTRADA ANALÓGICA

```
void setup() {
    analogReference(INTERNAL);
}

void loop() {

    int sensor = analogRead(A0);

}
```

ENTRADA ANALÓGICA

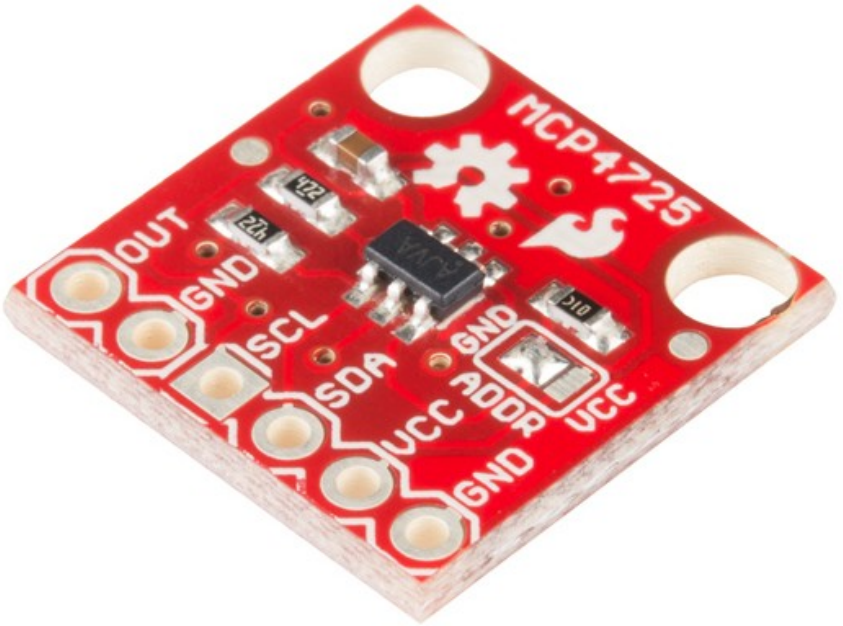


ENTRADA ANALÓGICA



SALIDA PWM

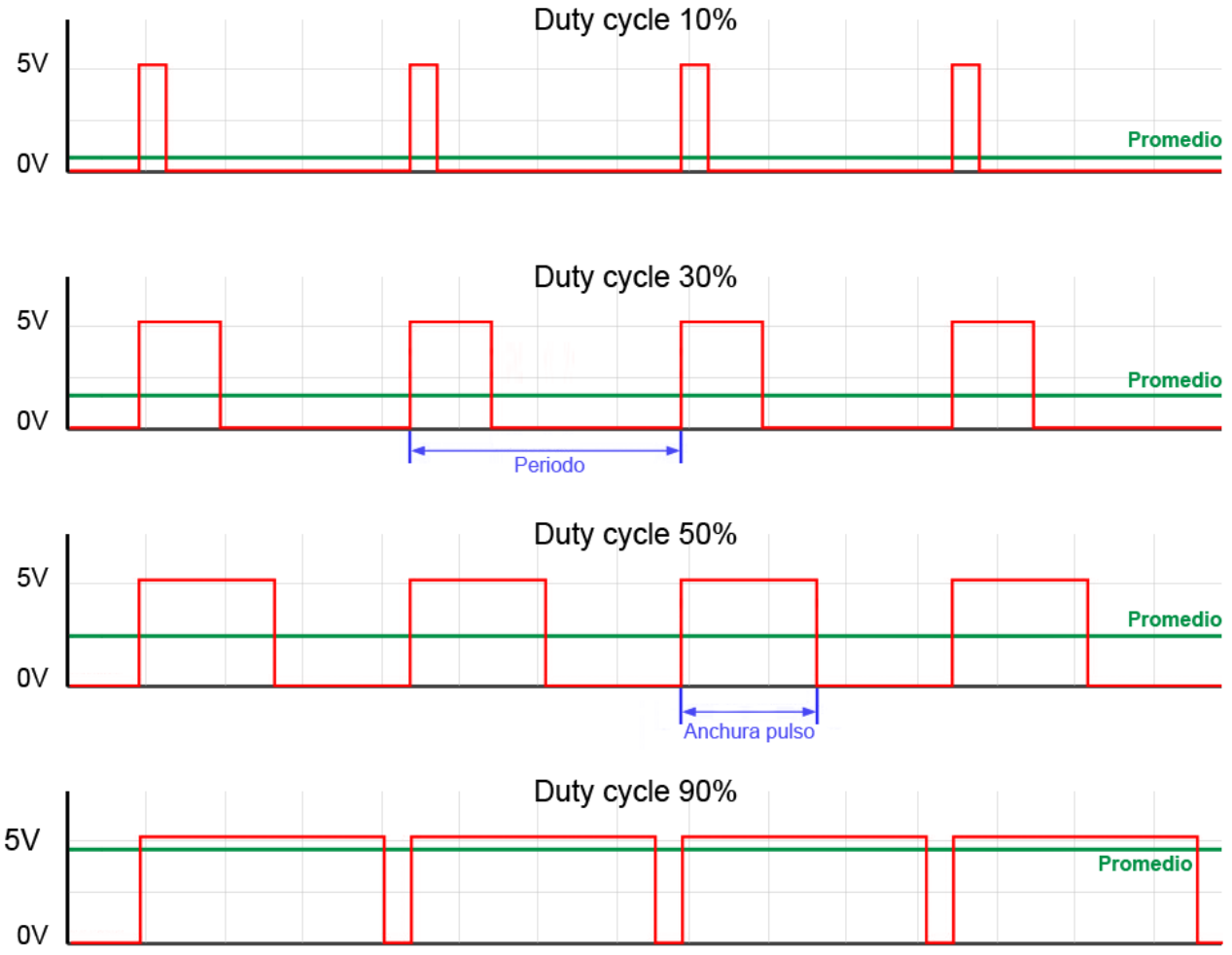
Arduino **no es capaz** de proporcionar una auténtica salida analógica. Para salvar esta limitación y simular una salida analógica emplea una señal PWM, que consiste en activar una salida digital durante un tiempo y mantenerla apagada durante el resto. El promedio de la tensión de salida, a lo largo del tiempo, será igual al valor analógico deseado.



SALIDA PWM

Es importante recordar en todo momento que en una salida PWM el **valor de tensión realmente es VCC**. Por ejemplo, si estamos alimentando un dispositivo que necesita 3V, y usamos una señal PWM, en realidad estaremos suministrando 5V durante un 60% del tiempo y 0V durante el 40%. Pero si el dispositivo, por ejemplo, soporta como máximo 3V, podemos dañarlo si lo alimentamos mediante un PWM.
En la modulación de ancho de pulso (PWM) se mantiene constante la frecuencia (es decir, el tiempo entre disparo de pulsos), mientras que se hace variar la anchura del pulso.

SALIDA PWM


Duty cycle 10%
5V
0V
Promedio
Duty cycle 30%
5V
0V
Promedio
Periodo
Duty cycle 50%
5V
0V
Promedio
Anchura pulso
Duty cycle 90%
5V
0V
Promedio

SALIDA PWM

BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev2, Nano Every	3, 5, 6, 9, 10	976 Hz
MKR boards *	0 - 8, 10, A3, A4	732 Hz
MKR1000 WiFi *	0 - 8, 10, 11, A3, A4	732 Hz
Zero *	3 - 13, A0, A1	732 Hz
Nano 33 IoT *	2, 3, 5, 6, 9 - 12, A2, A3, A5	732 Hz
Nano 33 BLE/BLE Sense	1 - 13, A0 - A7	500 Hz
Due **	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz

SALIDA PWM

Las funciones PWM por hardware emplean los **timers** para generar la onda de salida. Cada timer da servicio a 2 ó 3 salidas PWM. Para ello dispone de un registro de comparación por cada salida. Cuando el tiempo alcanza el valor del registro de comparación, la salida invierte su valor.

Cada salida conectada a un mismo temporizador comparte la misma frecuencia, aunque pueden tener distintos ciclos de trabajo, dependiendo del valor de su registro de comparación.

SALIDA PWM

Asociación de timers y PWM, en el caso de Arduino Uno, Mini y Nano:

- El Timer0 controla las salidas PWM 5 y 6.
- El Timer1 controla las salidas PWM 9 y 10.
- El Timer2 controla las salidas PWM 3 y 11.

Mientras que en Arduino Mega

- El Timer0 controla las salidas PWM 4 y 13.
- El Timer1 controla las salidas PWM 11 y 12.
- El Timer2 controla las salidas PWM 9 y 10.
- El Timer3 controla las salidas PWM 2, 3 y 5.
- El Timer4 controla las salidas PWM 6, 7 y 8.
- El Timer5 controla las salidas PWM 44, 45 y 46.

SALIDA PWM

La **frecuencia** de cada PWM depende de las características del temporizador al que está conectado, y de un registro de preescalado, que divide el tiempo por un número entero.
La frecuencia de los PWM se puede modificar cambiando el preescalado de los Timer correspondientes.

SALIDA PWM

Arduino Uno, Mini y Nano disponen de tres temporizadores:

- **Timer0**, con una frecuencia de 62500Hz, y preescalados de 1, 8, 64, 256 y 1024.
- **Timer1**, con una frecuencia de 31250Hz, y preescalados de 1, 8, 64, 256, y 1024.
- **Timer2**, con una frecuencia de 31250Hz, y preescalados de 1, 8, 32, 64, 128, 256, y 1024.

Arduino Mega añade tres temporizadores más

- **Timer3, 4 y 5**, con una frecuencia de 31250Hz, y preescalados de 1, 8, 64, 256, y 1024.

La **frecuencia estándar** para las salidas PWM en Arduino Uno, Mini y Nano es de **490Hz** para todos los pines, excepto para el 5 y 6 cuya frecuencia es de **980Hz**.
En cuanto a Arduino Mega, la frecuencia estándar es de **490Hz** para todos los pines, excepto para el 4 y 13 cuya frecuencia es de **980Hz**.

SALIDA PWM

El uso de los timers no es exclusivo de las salidas PWM, si no que es compartido con otras funciones. Emplear funciones que requieren el uso de estos timers supondrá que no podremos emplear de forma simultánea alguno de los pines PWM.
A continuación alguna de las incompatibilidades más frecuentes en Arduino Uno, Mini y Nano:

- La **librería servo** hace un uso intensivo de temporizadores por lo que, mientras la estemos usando, no podremos usar algunas de las salidas PWM. La librería usa el Timer 1, por lo que no podremos usar los pines 9 y 10 mientras usemos un servo.
- El pin 11 se emplea también para la **función MOSI de la comunicación SPI**. Por lo tanto, no podremos usar ambas funciones de forma simultánea en este pin.
- La **función Tone** emplea el Timer 2, por lo que no podremos usar los pines 3 y 11.

SALIDA PWM

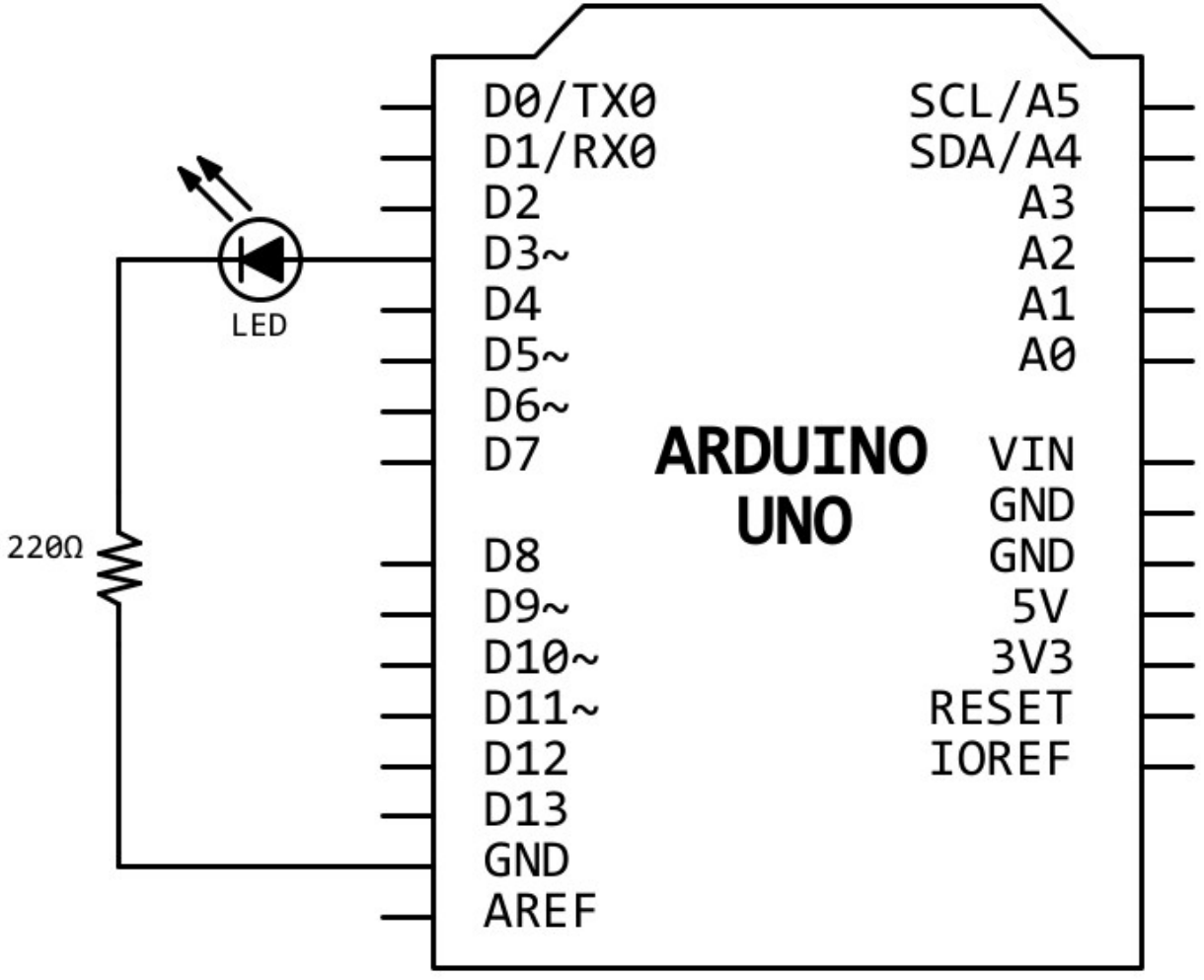
No se **define** en el *setup*.

Se **escribe** con la instrucción analogWrite(pin, valor) y acepta valores enteros de 0 a 255 (8 bits). Por ejemplo:

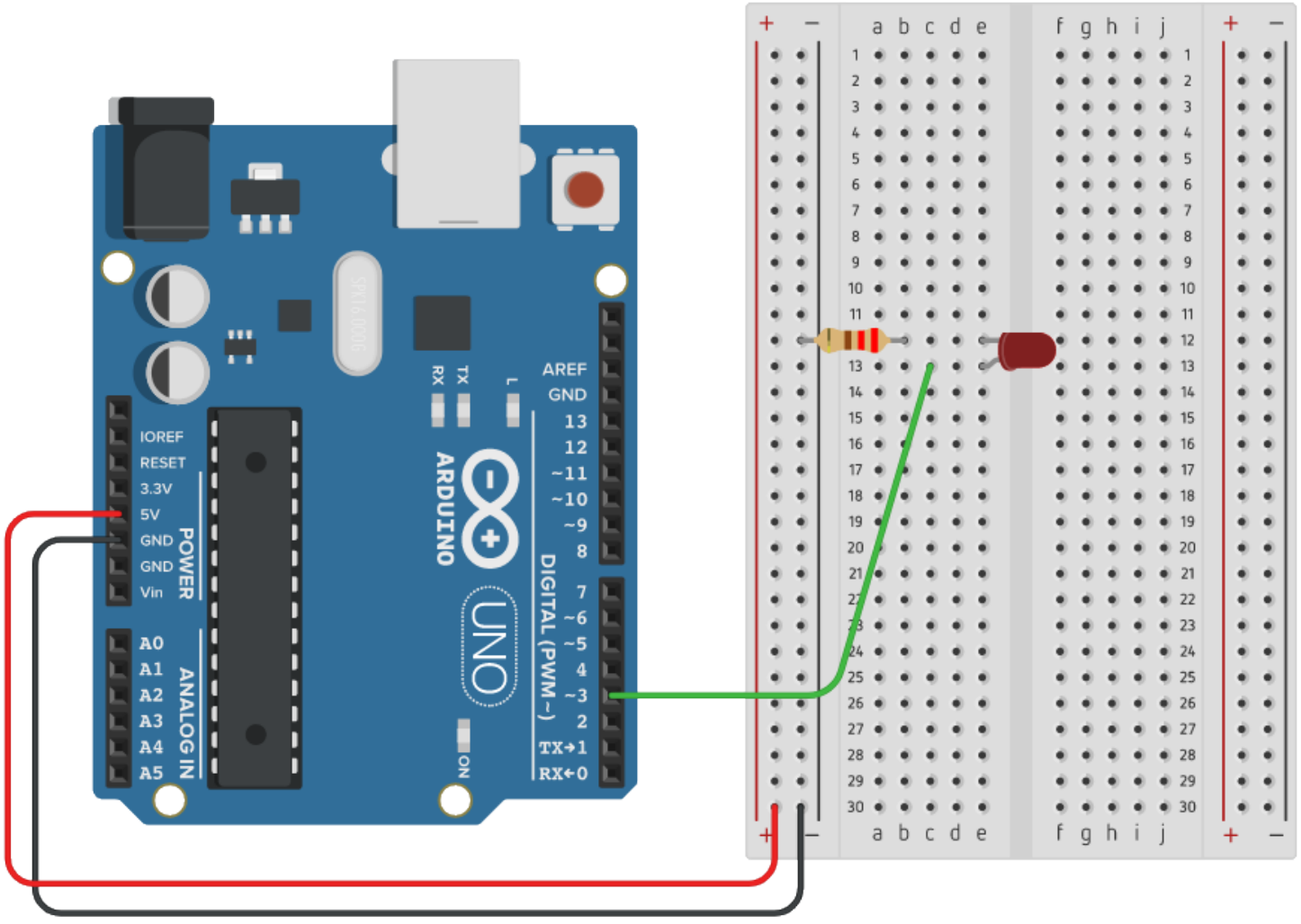
```
analogWrite(3, 100);
```

»»» analogWrite()

SALIDA PWM


D0/TX0
D1/RX0
D2
D3~
D4
D5~
D6~
D7
D8
D9~
D10~
D11~
D12
D13
GND
AREF
SCL/A5
SDA/A4
A3
A2
A1
A0
VIN
GND
GND
5V
3V3
RESET
IOREF
ARDUINO
UNO
LED
220Ω

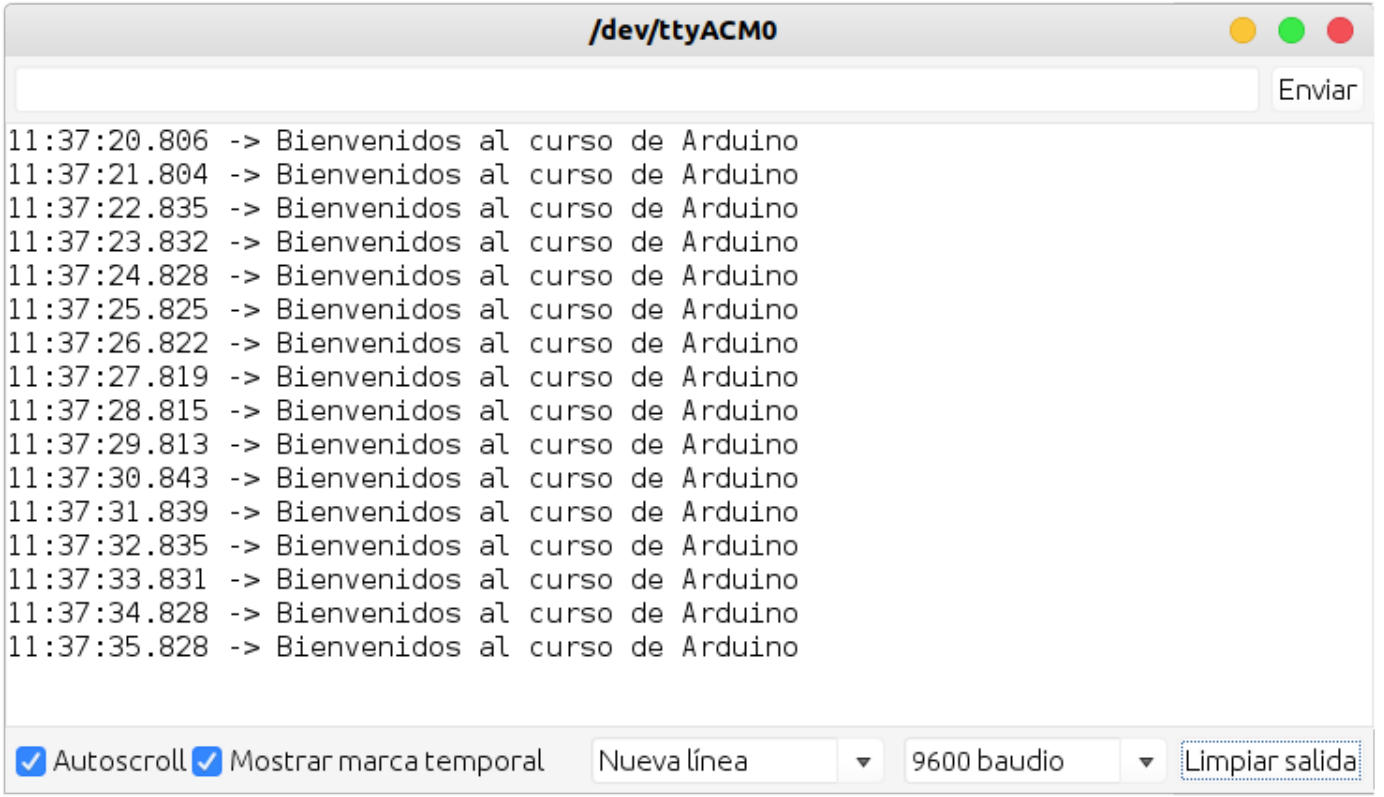
SALIDA PWM


IOREF
RESET
3.3V
5V
GND
GND
Vin
POWER
A0
A1
A2
A3
A4
A5
ANALOG IN
AREF
GND
13
12
~11
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
TX
RX
L
ON
ARDUINO
UNO
a b c d e
f g h i j
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30

MONITOR SERIE

El monitor serial es un conexión entre la computadora y la placa Arduino. Permite **enviar y recibir mensajes** de texto, útiles para la **depuración** y también **control** de Arduino.
El monitor de puerto serie es una pequeña utilidad integrada dentro del IDE que nos permite enviar y recibir fácilmente información a través del puerto serie. Su uso es muy sencillo, y dispone de dos zonas, una que muestra los datos recibidos, y otra para enviarlos.

MONITOR SERIE



MONITOR SERIE

Vuelca o envía un dato (variable, número o cadena), al puerto serie; lo hace mediante la representación textual de su valor. Dicho comando puede tomar diferentes formas, dependiendo de los parámetros que usemos para definir el formato de volcado de los datos.

```
Serial.print("texto a mostrar");
Serial.print(variable [,formato]);
Serial.print(variable [,dígitos]);
```

- **variable:** puede ser de cualquier tipo.
- **formato:** por defecto es DEC, pero se puede usar HEX, OCT o BIN.
- **dígitos:** si la variable es tipo float o double se puede especificar el numero de decimales.

MONITOR SERIE

Se **define** con la instrucción Serial.begin(velocidad), donde velocidad se mide en bits por segundo (baud). Por ejemplo:

```
Serial.begin(9600);
```

Se **escribe** en pantalla con la instrucción Serial.print(valor). Por ejemplo:

```
Serial.print(variable);
Serial.print(“Mensaje”);
Serial.println(“Mensaje”);
```

»»»» Serial
»»»» Serial.begin()

MONITOR SERIE

Es posible que **print()** muestre varios valores al mismo tiempo pero estos deben ser **concatenados** en una variable **String** previamente. Por ejemplo:

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    String sensor = "Sensor: ";
    sensor = sensor + analogRead(A0);
    Serial.println(sensor);
}
```

»»»» string

MONITOR SERIE

Los literales de caracteres y cadenas también pueden representar caracteres especiales que son difíciles o imposibles de expresar de otra manera en el código fuente de un programa, como "nueva linea" (\n) o "tab" (\t). Estos caracteres especiales son todos precedidos por un carácter de barra diagonal inversa (\).

Codigos de escape

Código	Descripción	ASCII	DEC
\n	nueva línea	LF	10
\r	retorno de carro	CR	13
\t	tabulacion horizontal	HT	9
\v	tabulacion vertical	VT	11
\b	retroceso	BS	8
\f	salto de página	FF	12
\a	alerta (pitido)	BELL	7
\'	comilla simple	'	39
\"	comillas dobles	"	34
\?	signo de interrogación	?	63
\\	barra invertida	\	47

MONITOR SERIE

Serial.write() escribe datos binario en el puerto serie. Estos datos son enviados como un byte o serie de bytes; para enviar caracteres representando los dígitos de un número usar la función print().

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.write(45);
}
```

»»»» Serial.write()

MONITOR SERIE

Serial.read() lee datos que ingresan por el puerto serie. Retorna el primer byte entrante si está disponible ó -1 si no hay datos disponibles.

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    // envía datos solo cuando recibe datos:
    if (Serial.available() > 0) {
        // leer el byte ingresante:
        char incomingByte = Serial.read();
        // mostrar:
        Serial.print("Recibido: ");
        Serial.println(incomingByte);
    }
}
```

ESCALADO

Mapea un número dentro de un rango a otro. Se realiza con la instrucción map(valor, desde mínimo, desde máximo, hacia mínimo, hacia máximo). Por ejemplo:

```
var = map(var, 0, 1023, 0, 255);
```

»»»» map()

DESAFÍO

¡A experimentar!

CRÉDITOS

Lucas Martín Treser
lmtreser@gmail.com – www.automatismos-mdq.com.ar