Funciones interesantes en Arduino

- Para indicar si un pin emite (salida) o recibe (entrada) electricidad: **`pinMode(PIN, MODO)`**, con modo *OUTPUT* (salida) o *INPUT* (entrada).
- En un **pin digital de salida**, para hacer que emita electricidad o no: **`digitalWrite(PIN, MODO)`**, con modo *HIGH* cuando queremos encenderlo o *LOW* si queremos apagarlo.
- En un **pin digital de salida PWM (con el símbolo ~)**, para indicar cuánta electricidad debe pasar: **`analogWrite(PIN, VALOR)`**, con un número entre 0 (nada de electricidad) y 255 (el máximo que Arduino da, 5 VOLTIOS).
- En un **pin digital de entrada**, para saber si está recibiendo electricidad o no: **`digitalRead(PIN)`**, nos devolverá 1 si recibe y 0 si no recibe.
- En un **pin analógico de entrada**, para indicar cuánta electricidad recibe: **`analogRead(PIN)`**, nos devolverá un número entre 0 (nada de electricidad) y 1023 (el máximo que Arduino puede recibir, 5 VOLTIOS).
- Para que después de las demás funciones el Arduino espere el tiempo que nosotros queramos: **`delay(TIEMPO)`**, el tiempo lo daremos en milisegundos. 1 segundo = 1000 milisegundos.

Recuerda que después de escribir una de estas funciones debes poner el **símbolo del punto y coma** (**;**). Si las hemos escrito bien, se volverán naranjas.

Variables en Arduino: Tipos y cómo usarlas

Ya dijimos que las variables son “datos”, es decir, tienen un nombre y un valor que van siempre juntos. También son de un tipo concreto: pueden ser números, letras, textos enteros…

¿Para qué queremos variables?

Imagínate que quieres que Arduino encienda y apague una bombilla muchas veces. Has puesto en cada sitio el número del pin al que está conectada, pero de repente cambias la bombilla de pin. ¿Qué puedes hacer? Puedes cambiar numerito por numerito en los 200 sitios en los que lo has tenido que poner… O puedes hacer una variable cuyo valor sea ese número para cambiar todo de una sola vez. Maravillosa jugada.

¿Cómo le pasamos a Arduino una variable? En una línea, le tendremos que decir el tipo de variable, el nombre que le ponemos y el valor que queramos que tenga: **`tipo nombre = valor;`** (acordaos del punto y coma). A continuación veremos qué tipos hay y qué valores tienen.

Tipos de variables y sus valores posibles

- **Los datos VERDADERO/FALSO:** usaremos el tipo **`bool`** , y su valor será **`true`** (verdadero) o **`false`** (falso).
- **Los números**
    - Si son números sin decimales, positivos o negativos, **usaremos el tipo `int`** . También existe el **`long`** , que es “más grande”. Como valor escribiremos el número sin más.
    - Si son números con decimales, positivos o negativos, **usaremos el tipo `float`** . Como valor escribiremos el número, y en vez de una coma usaremos un punto.
- **Las letras:** usaremos el tipo **`char`** , que es cuando solo tenemos una letra. La letra la escribiremos entre comillas simples (**`'X'`**), y nos aparecerá en un azul algo verdoso.

- **Las palabras, frases y textos:** usaremos el tipo **`String`** (sí, con mayúscula al principio, es el rarito). Todo el texto lo escribiremos entre comillas dobles (**`"Palabra o FRASE. O un texto."`**), y nos aparecerá en un azul más oscuro que el **`char`** .

Con todo esto, podemos coger la estructura **`tipo nombre = valor;`** y cambiar el tipo y el valor por una de las opciones anteriores.

El nombre que le pongamos da igual y lo podemos elegir libremente, **pero nunca puede tener espacios ni guiones (-)**. Podemos usar **barras bajas (_)** o poner mayúscula en cada palabra en vez de espacios si queremos.

Los tipos aparecerán en azul si los escribimos correctamente.

Podemos “declarar” las variables (darles un tipo, nombre y valor) tanto dentro como fuera de las llaves, pero siempre en una línea que esté vacía.

Vamos a poner algunos ejemplos:

```
bool meGustanLosRobots = true;
int numeroDePlacasArduino = 5;
float cuantosMetrosMidePauGasol = 2.13;
char primeraLetraDeMiNombre = 'A';
String nombre_del_taller = "Curso de Arduino";
```

Matemáticas en Arduino (con variables, por supuesto)

Arduino puede sumar, restar, multiplicar, dividir y hacer un montón de cosas más muy rápidamente, de hecho puede hacerlo alrededor de un millón de veces cada segundo.

Operadores matemáticos:

- Suma: +
- Resta: -

- Multiplicación: * (estrella)
- División: / (una sola barra)

Estos operadores los escribiremos entre dos números cualquiera. ¡Esto también incluye a las variables de números! El resultado lo tendremos que “guardar” en otra variable. ¿Cómo? Declarando una variable como antes, pero el valor en vez de ser un número será la operación.

Condicionales: controlando aún más

Los condicionales nos sirven para decidir automáticamente si queremos que el Arduino haga una parte de nuestro código o no.
Podemos comparar dos números, resultados o variables fácilmente escribiendo los siguientes símbolos entre dos números.

- Igual que: ==
- Distinto a: !=
- Mayor que: >
- Mayor o igual que: >=
- Menor que: <
- Menor o igual que: <=

En el caso de **`char`** o **`String`** , solo podemos usar:

- == (son iguales)
- != (son diferentes)

Todas estas comparaciones nos devolverán un **`bool`**, es decir, **`true`** o **`false`** . Ahora es cuando podemos usarlas para controlar el programa de Arduino.
El condicional **`if`** necesita que le pasemos un **`bool`** entre sus paréntesis para que decida si el código debe ser ejecutado o no. Si es verdadero, lo hará, y si es falso, no lo hará. Tiene la siguiente estructura:

```
if (true) { //CONDICIÓN VERDADERA O FALSA
    hacer_algo();
}
```

Bucles, para repetir muchas veces sin cansarnos

Si queremos hacer algo varias veces no vamos a escribir lo mismo cientas de veces, podemos utilizar los bucles. Hay distintos tipos, cada uno tiene un uso determinado, aunque nos las podemos apañar para intercambiarlos.

El Bucle FOR

El más famoso es el bucle **`for`**, que repite lo que pongamos dentro de él tantas veces como le digamos. Además, nos deja usar la variable que nos indica el número de la repetición que se está ejecutando en ese momento.
Por ejemplo, de forma básica si queremos *hacer algo* (lo mismo) 100 veces con un bucle **`for`** podemos escribir:

```
for (int i = 0; i < 100; i++) { //De X a Y cada Z
    hacer_algo();
}
```

Donde la variable **`i`** es un número sin decimales. Cuando se haga por primera vez, **`i`** tendrá un valor de **0**. Cuando Arduino acabe de hacer todo lo que hay dentro, cambiará el valor de la variable **`i`** a **1**, es decir, sumará 1 al valor que tenía antes, y volverá a hacer todo lo que hay dentro. Esto se repetirá hasta que **`i`** sea 99, que es el último número menor que 100 y por tanto el último que cumplirá la condición que hemos puesto.
Como hemos empezado en el 0 y hemos llegado a 99 de 1 en 1, lo que hayamos puesto dentro se habrá hecho **100 veces**, pero solo hemos escrito la función una vez.

Comunicar el Arduino con el ordenador

Para hacer que la placa Arduino hable con el ordenador al que está conectado, utilizaremos la herramienta **`Serial`** . Nos servirá para pasar datos del PC al Arduino, pero la mayor parte de las veces lo usaremos al contrario. Así podremos leer las variables de nuestro Arduino rápidamente y, de paso, controlar la parte del código en la que está.

Las funciones clave son las siguientes:

- **`Serial.begin()`** es la que "enciende" la comunicación, debemos llamarla en el **`setup()`** ya que solo es necesario hacerlo una vez y siempre al principio.
- **`Serial.print("Texto")`** envía un mensaje de texto desde el Arduino hacia el ordenador en la misma línea que el anterior mensaje (si es que no es el primero).
- **`Serial.println("Texto")`** hace lo mismo que la anterior pero esta vez escribe el texto en una nueva línea y no en la misma.
- **`Serial.available()`** nos dice si hay algún mensaje del ordenador "esperando" a ser leído. Es de tipo **`bool`** (verdadero o falso).
- **`Serial.read()`** nos devuelve un byte cada vez (puede ser un número o una letra, no hace falta que nos preocupemos) si es que hay un mensaje del ordenador "esperando" a ser leído. Lo tendremos que guardar en una variable de tipo **`int`** (número) o **`char`** (letra).
- **`Serial.readString()`** nos devuelve siempre un **`String`** (un texto entero) de golpe, así que no tenemos que preocuparnos de repetir **`Serial.read()`** muchas veces, ya que espera a que lleguen todos los datos para devolvernos todo a la vez.