

# Estructuras básicas para entendernos con Arduino

De momento ya hemos puesto en funcionamiento nuestro Arduino y hemos visto algunos de los sensores y actuadores que podemos emplear con él. Ahora ha llegado el momento de ver con más detalle cómo darle las órdenes para hacerlo funcionar. Ya hemos visto algo de esto en los ejemplos de apartados anteriores (Véase el apartado [Entender el código para encender el LED](#)) donde aparecían funciones y palabras reservadas.

Ahora ha llegado el momento de ir un poco más allá, para poder comunicarnos con Arduino, y para ello, va a ser necesario que conozcamos algunas estructuras básicas comunes a todos los lenguajes de programación imperativa.

## Un momento... ¿programación imperativa?

<https://giphy.com/embed/M452ElhGUd3byc4HYw>

A la hora de programar nuestros proyectos y dependiendo del propósito de nuestro proyecto, existen diferentes paradigmas de programación. No vamos a ahondar en este tema más allá de mencionarlo, pero al menos, es importante saber qué significa eso de programación imperativa, porque probablemente nos encontremos con ese concepto si continuamos desarrollando proyectos de programación.

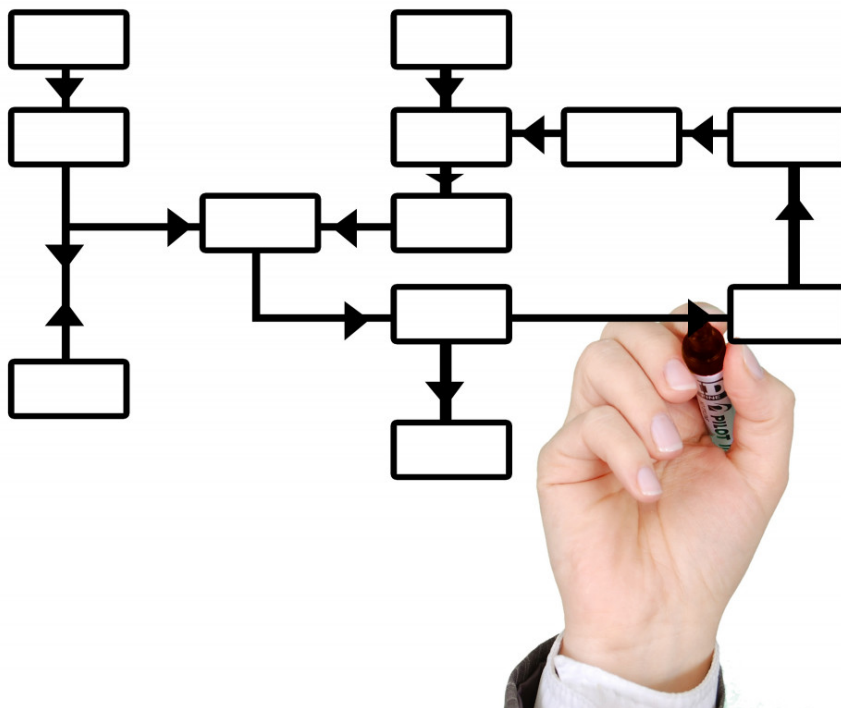
Básicamente, si programamos utilizando el paradigma imperativo lo que haremos será dar órdenes explícitamente. Un ejemplo de ello sería la sentencia: "Realiza la acción que yo quiera todo el rato". ¿Nos suena de algo? Eso es exactamente lo que hace la **función loop()** en Arduino.

Parece lógico que para programar tengamos que dar órdenes al ordenador o microcontrolador en cuestión, pero **existen otros paradigmas** como, por ejemplo, el **declarativo**, que no funcionan así. Un ejemplo de ello sería el caso en el que a nuestro ordenador le proporcionamos una conjunto de hechos y reglas y posteriormente le hacemos una consulta para que nos devuelva esa información. Por ejemplo, si creamos una base de hechos que contenga una serie de libros y su autor, podemos preguntar: ¿Quién ha escrito este libro? Y la aplicación nos devolverá esa información.

Si tienes interés en leer más sobre este tema, puedes echarle un vistazo [a estas páginas](#).

Existen problemas que podemos resolver empleando ambos paradigmas y cuando programamos es necesario decidir cuál es el más adecuado. No obstante, aquí no vamos a tener ese problema, porque con los proyectos que vamos a ver en este curso siempre vamos a emplear el paradigma imperativo.

La estructura de estos programas suele representarse con lo que se conoce como **diagrama de flujo**. Con él, se representan los diferentes caminos que puede tomar nuestro algoritmo dependiendo de las condiciones que se cumplan o no. Estas condiciones será lo primero que veremos, ya que son la base sobre la que se construye cualquier algoritmo.



Para más información sobre diagramas de flujo puedes consultar [esta página](#) de otro curso de Aularagón.

## Condicionales (if-else-else if)

Estas estructuras se encargan de controlar qué acciones de nuestro algoritmo van a ejecutarse y cuáles no. Solo hay dos opciones: o bien se cumplirán las condiciones, o no.

Las palabras clave de esta estructura son:

## IF - ELSE

- **IF:** A esta palabra le seguirá la condición que deberá cumplirse para ejecutar una serie de acciones.
- **ELSE:** irá seguida de las acciones que se ejecutarán en caso de que la condición no se cumpla.

## Un ejemplo para verlo en acción

No hay mejor manera para entender algo que ver una demostración de su uso. Para ello, vamos a recurrir a uno de los ejemplos que nos brinda la IDE de Arduino. Este ejemplo podemos encontrarlo dentro de:

### Archivo > Ejemplos > 01. Basics > Fade



```
/*
  Fade

  This example shows how to fade an LED on pin 9 using the analogWrite()
  function.

  The analogWrite() function uses PWM, so if you want to change the pin you're
  using, be sure to use another PWM capable pin. On most Arduino, the PWM pins
  are identified with a "~" sign, like ~3, ~5, ~6, ~9, ~10 and ~11.

  This example code is in the public domain.

  https://www.arduino.cc/en/Tutorial/BuiltInExamples/Fade
*/

int led = 9;          // the PWM pin the LED is attached to
int brightness = 0;   // how bright the LED is
int fadeAmount = 5;   // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

En este ejemplo, lo que hacemos es aumentar y disminuir la luminosidad de un LED de manera progresiva, en lugar de encenderlo o apagarlo del todo, como hacíamos en la práctica del bloque anterior.

Para ello, haremos uso de unos **pines especiales (PWM)**, los cuáles pueden ser configurados para funcionar con modulación por ancho de pulsos. Y, ¿esto qué significa? pues que no solamente tienen dos posiciones (encendido o apagado), sino que podemos controlar cómo de apagado y encendido está controlando la cantidad de energía que recibe.

Regular esta cantidad de energía puede hacerse con la función **analogWrite()**. Ya habíamos visto la función **digitalWrite()**. Si no te acuerdas, puedes echarle un vistazo [aquí](#). `analogWrite()` nos permite darle al pin elegido un valor de 0 a 255, siendo 0 totalmente apagado y 255 totalmente encendido. ¿Cuál es la particularidad de esta función? que, a diferencia de `digitalWrite()`, que puede ser configurada con cualquier pin, `analogWrite()` no.

Una lista de los pines que admiten PWM dependiendo del Arduino que estemos utilizando la encontramos [aquí](#).

De este ejemplo, por el momento, vamos a ver concretamente la parte relacionada con el condicional, pero volveremos a él después para utilizarlo como ejemplo de uso de **palabras reservadas** y **funciones**. En este algoritmo, el condicional aparece en las siguientes líneas.

```
if (brightness <= 0 || brightness >= 255) {  
    fadeAmount = -fadeAmount;  
}
```

En estas líneas encontramos la palabra reservada **if** seguida de dos condiciones encerradas entre paréntesis. Lo que traducido a español significaría: "Si el brillo es menor o igual que cero **o** el brillo es mayor o igual que 255". La disyunción nos viene dada por el operador lógico **OR** que para Arduino se traduce como `||`. Otros operadores lógicos muy comunes son **AND**, que se traduce por `&&` y **NOT**, que se escribe `!`.

Entre corchetes `{ }` encontramos la acción a realizar. En este caso, revertir la cantidad de brillo que emitirá nuestro LED: si lo hemos apagado, lo comenzaremos a encender y al revés, si lo hemos encendido completamente, comenzaremos a apagarlo.

En este caso, no existe una acción que deba ejecutarse en caso de que no se de alguna de las dos circunstancias encerradas en el condicional, pero si la hubiese aparecería detrás de la palabra **else**.

Por tanto, un esquema de la estructura de los condicionales sería:

```
if (Condición) {  
    Acción A  
} else {  
    Acción B  
}
```

Siendo posible también:

```
if (Condición) {  
    Acción  
}
```

## Una última palabra sobre los condicionales: **else if**

Existe también la posibilidad de que tengamos que realizar selecciones que den lugar a más de dos posibilidades, como por ejemplo en el caso que estemos leyendo ciertos valores de un sensor y queramos que un actuador realice diferentes acciones dependiendo de ellos.

Si el sensor recibe valores entre 0 y 255 --> Acción A  
Si el sensor recibe valores entre 256 y 511 --> Acción B  
Si el sensor recibe valores entre 512 y 1024 --> Acción C

Una forma de conseguir esto es anidar varias condicionales. Para ello necesitaremos las palabras **else if**, las cuales indican una segunda condición, o sucesivas.

La estructura sería la siguiente:

```
if (Condición 1) {  
    Acción A  
} else if (Condición 2) {  
    Acción B  
} else {  
    Acción C  
}
```

## Iteraciones (for y while)

Definiremos como iteración a la ejecución sistemática de una serie de acciones determinadas mientras se dé una condición específica. Existe una condición que se analiza, normalmente, cada vez que se repite dicha iteración. Si esta condición se sigue cumpliendo, volveremos a repetir el proceso; si, por el contrario, ya no se cumple, pasaremos a la siguiente línea de código y ejecutaremos la acción correspondiente.

Las dos más comunes son **for** y **while**. En la sentencia for, existe un índice cuyo valor va aumentando o disminuyendo conforme se ejecuta el algoritmo y la condición de la iteración va comprobando si se ha alcanzado el límite o no.

Un esquema de esta estructura sería:

```
for (int indice= inicial; índice<= final; índice++){  
    Acción  
}
```

Es importante no olvidar colocar los paréntesis, puntos y comas y los corchetes donde corresponda, o Arduino se quejará y no nos hará caso...

## Un ejemplo para verlo en acción

Un ejemplo sencillo de esta estructura, que te aconsejo que construyas para verlo en funcionamiento, lo encontramos en este fragmento de código:

```
// Controlar la luminosidad de un LED usando PWM y analogWrite()
int PWMpin = 9; // LED conectado al pin 9, con PWM.

void setup() {
  // no es necesario
}

void loop() {
  for (int i = 0; i <= 255; i++) {
    analogWrite(PWMpin, i);
    delay(10);
  }
}
```

Como podemos comprobar, el bucle for se ejecuta continuamente por estar dentro de la función **void loop()** y lo que hace es encender continuamente el LED. En este caso únicamente lo enciende, no se encarga de apagarlo.

1. Conectaremos con un cocodrilo uno de nuestros LED al pin 9.

Para ello, conectaremos un extremo del cocodrilo al signo + del LED y el otro extremo del cocodrilo al pin 9, con MUCHO CUIDADO de que el cocodrilo exclusivamente esté en contacto con ese pin y no con los de los laterales:



2. Conectaremos un cocodrilo al signo - del LED y el otro extremo a un pin GND de nuestro Arduino.

3. Copiaremos y pegaremos el código anterior a un sketch nuevo de Arduino, lo guardaremos, lo subiremos a nuestro microcontrolador y...

aquí lo podemos ver en funcionamiento: