PRIMEROS PASOS CON ARDUINO Y C

ESTRUCTURA DE UN PROGRAMA

```
void setup() {
instruciones;
}

void loop() {
instruciones;
}
```

Las instrucciones del bloque setup, se ejecutan sólo una vez y al principio de la ejecución del programa.
Las instrucciones del bloque loop, se ejecutarán de forma cíclica mientras no se apague la placa arduino.

FUNCIONES

Una función es un bloque de código que tiene un nombre, y que se ejecuta cuando es llamado dentro de otra función o bloque.
Una función normalmente recibe unos parámetros o valores con los que debe hacer algo. Estos parámetros se pasan a la función entre paréntesis.
Una función normalmente devolverá otro valor, que será del tipo (entero, carácter,byte..) que se declare. Si la función no devuelve nada, se declara como "void".
Ejemplo:

```
int suma(int a,int b)
{
  return a+b;
}
```

{} 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.

; punto y coma

El punto y coma ";", se utiliza para separar instrucciones en lenguaje C. Siempre lo pondremos al final de la línea, aunque algunas instrucciones, como if, no lo llevan.

/*…*/ BLOQUE DE COMENTARIOS

Los bloques de comentarios ayudan a entender el código. Se deben poner todos los que se considere necesarios.

```
/*   esto es un bloque de comentario
     no se debe olvidar cerrar los comentarios
     estos deben estar equilibrados
*/
```

// LINEA DE COMENTARIOS

Para comentarios de un solo renglón, usamos la doble barra.

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

VARIABLES

Una variable es una maenra de nombrar y almacenar un valor numérico para su uso. Como su nombre indica, las variables son números que se pueden variar continuamente.
Una variable debe ser declarada y, opcionalmente, asignarle un valor.
Ejemplo

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

TIPOS DE VARIABLES

Las variables pueden contener valores de diferentes tipos, números enteros, números reales, caracteres, bytes, etc.
A la hora de declarar la variable, tendremos que decir el tipo de dato que contendrá para que el compilador le asigme memoria de forma adecuada.
Los tipos disponibles son:

- byte. Contienen un byte o número de 8 bits, del 0 al 255.
- int. Contienen un entero corto, del -32768 al 32767
- long. Contienen un entero largo. Desde -2.000 millones a +2.000 millones
- float. Contien u número con decimales. En la memoria se guardan en formato mantisa-exponente, como las calculadoras, por lo que pueden perder precisión.
- Char. Guarda una letra.

```
byte unaVariable = 180; // declara 'unaVariable' como tipo byte
int unaVariable = 1500; // declara 'unaVariable' como tipo entero
long unaVariable = 90000;//declara 'unaVariable' como tipo long
float unaVariable = 3.14; // declara 'unaVariable' como tipo flotante
char a='a'; //declara 'a' como tipo char y valor la letra 'a'
```

ARRAYS

Un array es un conjunto de valores a los que se puede acceder con un índice. Cualquier valor puede ser recogido o asignado con el nombre de la matriz y el úmero del índice.
Al primer valor le corresponde el índice 0.
Se pueden asignar valores antes de ser utilizado.

```
int miArray[5]; // declara un array de enteros de 6 posiciones
miArray[3] = 10; // asigna 1 valor 10 a la posición 4
byte parpadeo[] = {180,30,255,200,10,90,150,60}; // array de 8 valores
```

ARITMÉTICA

Son las operaciones habituales, suma, resta, multiplicación, división.

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

ASIGNACIONES COMPUESTAS

Son operaciones que combinan una operación aritmética con una variable.

```
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
```

OPERADORES DE COMPARACIÓN

Nos permiten comparar el valor de una variable con un valor determinado.

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

OPERADORES LÓGICOS

Nos permiten comparar dos expresiones y devolver un verdadero o falso.
Tenemos 3:

- and (y)
- or (o)
- not (no)

```
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

Son valores que vienen definidos en el lenguaje C de arduino, y al usarlos, son sustituidos por un valor numérico.

- VERDADERO/FALSO(TRUE/FALSE). Se traducen e un 0 ó un 1
- HIGH/LOW. Tambien se traducen por 0 ó 1.
- INPUT/OUTPUT. Indican entrada o salida

```
digitalWrite(13,  HIGH); /activa la salida 13 con un nivel alto (5v.)
pinMode(13,  OUTPUT); // designamos que el PIN 13 es una salida
if  (b == TRUE);
{
ejecutar las instrucciones;
}
```

CONDICIONAL if

Es una instrucción que nos permite comprobar si una determinada condición se cumple, como por ejemplo si una entrada digital tiene un valor alto o bajo.

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

BUCLE for

Nos permite repetir las instruciones encerradas entre llaves un número determinado de veces.
La declaración for, tiene tres partes separadas por ";".
El model es el siguiente:

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

Mientras se cumpla la condición se ejecutará el bucle.

```
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.
}
```

BUCLE while (mientras)

While especifica un bucle que se repite mientras se cumpla una condición.

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

BUCLE do…while (hacer..mientras)

Similar al bucle anterior, pero la condición es evaluada al final del bucle, con lo que nos aseguramos que el bucle se ejecutará la menos 1 vez.

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