INTRODUCCIÓN

Puede dividirse en: ESTRUCTURA, VARIABLES, FUNCIONES.

I. ESTRUCTURA GENERAL DE UN SKETCH/PROGRAMA

1.1. ESTRUCTURA BÁSICA DEL PROGRAMA:

Se compone de tres secciones principales:

1.1.1. Sección de declaración de variables: Ubicadas al inicio.

1.1.2. Sección llamada "*void setup*":

Delimitada por llaves de apertura y cierre. Es la primera función a ejecutar en el programa. Se ejecutan una única vez en el momento de encender o resetear la placa ARDUINO.
Se usa para inicializar:

- Los modos de trabajo de los pins E/S (*PinMode*)
- Comunicación en serie

Ejemplo:

```
int buttonPin = 3;
void setup()                    // Se ejecuta una zona vez, cuando el programa inicia
{
 Serial.begin(9600);            // Comunicación en serie
 pinMode(buttonPin, INPUT);     // Configura en pin digita l#3 como ENTRADA de datos
}
void loop()
{
 // ...
}
```

1.1.3. Sección llamada "void loop()":

Delimitada por llaves de apertura y cierre; incluye el código que se ejecuta continuamente leyendo entradas, activando salidas, etc. Esta función es el núcleo de todos los programas ARDUINO y hace la mayor parte del trabajo. Se ejecutan justo después de la sección "void setup()" infinitas veces hasta que la placa se apague (o se resetee).

Ejemplo:

```
Int pin = 10;
void setup()
 {
pinMode(pin,  OUTPUT);          // Establece  'pin'  como salida
}
void loop()
{
digitalWrite(pin, HIGH);        // pone en uno (on, 5v) el ´pin´
delay(1000);                    // espera un segundo (1000 ms)
digitalWrite(pin, LOW);         // pone en cero (off, 0v.) el ´pin´
delay(1000);                    // espera un segundo (1000 ms)
}
```

1.2. ESTRUCTURA DE CONTROL:

BLOQUES CONDICIONALES

1.2.1. Bloque "if" , "if/else" y "If/else if" :

- Las sentencias **if** comprueban si cierta condición ha sido alcanzada y ejecutan todas las sentencias dentro de las llaves si la declaración es cierta. Si es falsa el programa ignora la sentencia.

Ejemplo:

```
byte x=50;
void setup() {
        Serial.begin(9600);
        if (x >=10 && x <=20){
        Serial.println("Frase 1");
        }
        if (x >=10 || x <=20) {
        Serial.println("Frase 2");
        }
        if (x >=10 && !(x<=20)) {
        Serial.println("Frase 3");
        }
}
        void loop(){
}
```

Veremos que solamente se muestran la "Frase 2" y la "Frase 3". Esto es debido a que la condición del primer "if" es falsa: en él se comprueba si la variable "a" es mayor que 10 Y A LA VEZ si es menor de 20. En cambio, la condición del segundo "if" es cierta: en él se comprueba si la variable "a" es mayor que 10 O BIEN menor de 20: como ya se cumple una de las condiciones –la primera–, la condición total ya es cierta valga lo que valga el resto de condiciones presentes. La condición del tercer "if" también es cierta: en él se comprueba si la variable "a" es mayor que 10 y a la vez, gracias al operador "!", que NO sea menor de 20.

- Las sentencias **if/else** permiten un mayor control sobre el flujo del código que la declaración *if* básica, por permitir agrupar múltiples comprobaciones. *"Si esto no se cumple haz esto otro".*

Ejemplo:

```
String cad1="hola";
String cad2="hola";
void setup() {
        Serial.begin(9600);
}
        void loop() {
        if (cad1 == cad2){
        Serial.println("La cadena es
igual");
}
else
{
        Serial.println("La cadena es
diferente");
}
}
```

Hay que tener en cuenta que solamente se pueden utilizar los operadores de comparación cuando ambas cadenas han sido declaradas como objetos String: si son arrays de caracteres no se pueden comparar. En este sentido, comparar si una cadena es mayor o menor que otra simplemente significa evaluar las cadenas en orden alfabético carácter tras carácter (por ejemplo, "alma" sería menor que "barco", pero "999" es mayor que "1000" ya que el carácter "9" va después del "1"). Recalcar que las comparaciones de cadenas son case-sensitive (es decir, el dato de tipo String "hola" no es igual que el dato de tipo String "HOLA").

- Las sentencias **if/else if** pueden realizar múltiples comprobaciones en una misma estructura de condiciones. Cada comprobación procederá a la siguiente sólo cuando su propio resultado sea *FALSE*. Cuando el resultado sea *TRUE*, su bloque de código contenido, será ejecutado, y el programa esquivará las siguientes comprobaciones hasta el final de la estructura de comprobaciones.

Ejemplo:

```
int numero;
        void setup(){
        Serial.begin(9600);
        }
void loop(){
        if (Serial.available() > 0) {    //El nº introducido ha de estar en el rango del tipo "int"
        numero=Serial.parseInt();
        if (numero == 23){
        Serial.println("Numero es igual a 23");
        }
        else if (numero < 23) {
                Serial.println("Numero es menor que 23");
                }
                else {
                Serial.println("Numero es mayor que 23");
                }
        }
}
```

En el código anterior aparece un primer "if" que comprueba si hay datos en el buffer de entrada del chip TTL-UART pendientes de leer. Si es así (es decir, si el valor devuelto por *Serial.available()* es mayor que 0), se ejecutarán todas las instrucciones en su interior. En cambio, si la condición resulta ser falsa (es decir, si *Serial.available()* devuelve 0 y por tanto no hay datos que leer), fijarse que la función "loop()" no ejecuta nada.

En el momento que la condición sea verdadera, lo que tenemos dentro del primer "if" es la función *Serial.parseInt()* que reconoce y extrae de todo lo que se haya enviado a través del canal serie (por ejemplo, usando el "Serial monitor") un número entero, asignándolo a la variable "numero". Y aquí es cuando llega otro "if" que comprueba si el valor de "numero" es igual a 23. Si no es así, se comprueba entonces si su valor es menor de 23. Y si todavía no es así, solo queda una opción: que sea mayor de 23. Al ejecutar este sketch a través del "Serial monitor", lo que veremos es que cada vez que enviemos algún dato a la placa, esta nos responderá con alguna de las tres posibilidades.

Nota: Ahora que ya sabemos las diferentes sintaxis del bloque "if", veamos qué tipo de condiciones podemos definir entre los paréntesis del "if". Lo primero que debemos saber es que para escribir correctamente en nuestro sketch estas condiciones necesitaremos utilizar alguno de los llamados operadores de comparación,

☑ x == y (x es igual a y)

☑ x != y (x no es igual a y)

- ☑ x < y (x es menor a y)
- ☑ x > y (x es mayor a y)
- ☑ x <= y (x es menor o igual a y)
- ☑ x >= y (x es mayor o igual a y)

1.2.2. Bloque "switch/case":

```
void setup(){
        byte x=50;
        Serial.begin(9600);
        switch (x) {
                case 20:
        Serial.println("Vale 20 exactamente");
        break;
                case 50:
        Serial.println("Vale 50 exactamente");
        break;
        default:
        Serial.println("No vale ninguna de los
valores anteriores");
        }
}
void loop(){

}
```

Consta en su interior de una serie de secciones "case" y, opcionalmente, de una sección "default". Nada más llegar a la primera línea del "switch", primero se comprueba el valor de la variable o expresión que haya entre sus paréntesis. Si el resultado es igual al valor especificado en la primera sección "case", se ejecutarán las instrucciones del interior de la misma y se dará por finalizado el "switch", en caso de no ser igual el resultado de la expresión a lo especificado en el primer "case" se pasará a comprobarlo con el segundo "case", y si no con el tercero, etc. Por último, si existe una sección "default" (opcional) y el resultado de la expresión no ha coincidido con ninguna de las secciones "case", entonces se ejecutarán las sentencias de la sección "default".

BLOQUES REPETITIVOS (BUCLES)

1.2.3. Bloque "while":

Este bucle repite la ejecución de las instrucciones que están dentro de sus llaves de apertura y cierre mientras la condición especificada entre sus paréntesis sea cierta ("true",1), sin importar el número de veces repita. *El número de iteraciones realizadas depende del estado de la condición definida.*

```
byte x=1;
void setup(){
         Serial.begin(9600);
}
void loop(){
        while (x <= 50)
        {
        Serial.println("Es menor
de 50");
        x=x+1;
        delay(250);
}
        Serial.println("Es mayor
que 50");
```

Ejemplo: El "Serial monitor" los primeros 50 mensajes que aparecerán indicarán que la variable es menor de 50 porque se está ejecutando el interior del "while". Pero en el momento que la variable sea mayor (porque en cada iteración del bucle se le va aumentando en una unidad su valor), la condición del "while" dejará de ser cierta y saltará de allí, mostrando entonces el mensaje (ya de forma infinita) de que la variable es mayor que 50.

1.2.4. Bloque "do":

```
int pulsador = 24;
int led= 25;
void setup() {
  pinMode(pulsador, INPUT);
  pinMode(led, OUTPUT);
}
void loop() {
  do
  {
    digitalWrite(led, HIGH);
    delay(1000);
    digitalWrite(led, LOW);
    delay(1000);
  }
while (digitalRead(pulsador) == HIGH);
}
```

La condición es evaluada después de ejecutar las instrucciones escritas dentro de las llaves. Esto hace que las instrucciones siempre sean ejecutadas como mínimo una vez aun cuando la condición sea falsa, porque antes de llegar a comprobar esta, las instrucciones ya han sido leídas (a diferencia del bucle "while", donde si la condición ya de entrada era falsa las instrucciones no se ejecutaban nunca).

1.2.5. Bloque "For":

El bucle **for** es una estructura que se utiliza cuando queremos que una serie de acciones se repita un número determinado de veces, para ello se necesita de una variable índice, una condición y un incrementador. Por tanto, usaremos el bucle "for" para ejecutar un conjunto de instrucciones (escritas dentro de llaves de apertura y cierre) un número concreto de veces y estas son:

```
for (valor_inicial_contador;condicion_final;incremento){
//Instrucciones que se repetirán un número determinado de veces
}
```

- ✓ **Valor inicial del contador:** en esta parte se asigna el valor inicial de una variable entera que se utilizará como contador en las iteraciones del bucle. *Por ejemplo, si allí escribimos x=0 , se fijará la variable "x" a cero al inicio del bucle. A partir de entonces, a cada repetición del bucle, esta variable "x" irá aumentando (o disminuyendo) progresivamente de valor.*

- ✓ **Condición final del bucle:** en esta parte se especifica una condición. Justo antes de cada iteración se comprueba que sea cierta para pasar a ejecutar el grupo de sentencias internas. Si la condición se evalúa como falsa, se finaliza el bucle "for". *Por ejemplo, si allí escribimos x<10, el grupo interior de sentencias se ejecutará únicamente cuando la variable "x" valga menos de 10.*

- ✓ **Incremento del contador**: en la última de las tres partes es donde se indica el cambio de valor que sufrirá al inicio de cada iteración del bucle la variable usada como contador. Este cambio se expresa con una asignación (x++,x=x+1).

Ejemplos:

Un led parpadee 5 veces tras accionar un pulsador

```
int pulsador = 24;
int led= 25;
void setup()
{
pinMode(pulsador, INPUT);
pinMode(led, OUTPUT);
}
void loop()
{
 if (digitalRead(pulsador) == HIGH)
 {
  for (int i = 0; i<=4; i++)
  {
   digitalWrite(led, HIGH);
   delay(1000);
   digitalWrite(led, LOW);
   delay(1000);
  }
 }
else
 {
  digitalWrite(led, LOW);
 }
}
```

```
byte suma=0;
byte x;
void setup(){
   Serial.begin(9600);
   for (x=0;x<7;x=x+1){ //Sumo 7 veces el
número 5
  suma= suma + 5;
 }
}
 Serial.println(suma);
 Serial.println(suma/7); //Esto es la media
}
void loop(){}
```

En la primera repetición del bucle "for" se le asigna el valor del primer dato sumar; en la segunda repetición a ese valor de "suma" se le suma el segundo valor (por lo que en ese momento "suma" vale la suma de los dos primeros valores). En la tercera repetición se le añade el tercer valor a esa suma parcial (por lo que en ese momento "suma" vale la suma de los tres primeros valores), en la cuarta repetición el cuarto valor, y así y así hasta que se acaba de recorrer el bucle "for" y todos los datos se han ido añadiendo uno tras otro hasta conseguir la suma total. Finalmente, muestro el resultado, y muestro también la media, que no es más que esa suma total entre el número de datos introducidos en la suma.

```
byte x;
void setup(){
 Serial.begin(9600);
 for (x=0;x<10;x=x+1){
 delay(500);
Serial.println(x);
 }
}
void loop(){}
```

La sentencia "for" primero se inicializa el contador (x=0) y seguidamente se comprueba la condición. Como efectivamente, x<10, se ejecutará la –única en este caso– sentencia interior, que muestra el propio valor del contador por el "Serial monitor". Justo después, se realiza el incremento (x=x+1), volviéndose seguidamente a comprobar la condición. Si "x" (que ahora vale 1) sigue cumpliendo la condición (sí, porque 1<10), se volverá a ejecutar la instrucción interna, mostrándose ahora el valor "1" en el "Serial monitor". Justo después de esto se volverá a realizar el incremento (valiendo x ahora por tanto 2) y se volverá a comprobar la condición, y así y así hasta que llegue un momento en el que la condición resulte falsa (cuando "x" haya incrementado tanto su valor que ya sea igual o mayor que 10), momento en el cual se finalizará el "for" inmediatamente.

NOTA: Se presenta una tabla de "equivalencia" de algunos de los operadores compuestos.

x++	Equivale a x=x+1 (Al operador "++" se le llama operador "incremento")
x--	Equivale a x=x-1 (Al operador "--" se le llama operador "decremento")
x+=3	Equivale a x=x+3
x-=3	Equivale a x=x-3
x*=3	Equivale a x=x*3
x/=3	Equivale a x=x/3

```
int led;
void setup(){
 for(led=24;led<31;led++){
   pinMode(led,OUTPUT);
 }
}
 void loop(){
  for(led=24;led<31;led++){
    digitalWrite(led,HIGH);
    delay(1000);
    //digitalWrite(led,LOW);
  }
  for(led=30;led>23;led--){
    //digitalWrite(led,HIGH);
    //delay(1000);
    digitalWrite(led,LOW);
    delay(1000);
  }
 }
```

INSTRUCCIONES

1.2.6. **Instrucción "break":** Ejemplo

```
int x;
void setup() {
  Serial.begin(9600);
}
void loop(){
  while (Serial.available() > 0)
 {
 char dato = Serial. read();
     if (dato =='X') break;  //termina rutina al leer una letra X
 switch (dato)
    {
    case 'R':
         Serial.println("Se eligio el color rojo");
    break;
    case 'G':
        Serial.println("Se eligio el color verde");
    break;
    case 'B':
       Serial.println("Se eligio el color azul");
   break;
    }
  }
}
```

Es usada para salir de los siguientes bloques de bucle: "do","for" y "while", "switch", sin tener que esperar a que termine el bloque de instrucciones o a que deje de cumplirse la condición lógica.

Observar que ninguna de ellas incorpora paréntesis, pero como cualquier otra instrucción, en nuestros sketches deben ser finalizadas con un punto y coma.

Debe estar escritas dentro de las llaves que delimitan las sentencias internas de un bucle.

1.2.7. Instrucción "continue":

También debe estar escrita dentro de las llaves que delimitan las sentencias internas de un bucle y sirve para finalizar la iteración actual y comenzar inmediatamente con la siguiente. Es decir, esta instrucción forzará al programa a "volver para arriba" y comenzar la evaluación de la siguiente iteración aun cuando todavía queden instrucciones pendientes de ejecutar en la iteración actual. En caso de haber varios bucles anidados (unos dentro de otros) la sentencia "continue" tendrá efecto únicamente en el bucle más interior de ellos.

Ejemplo:

```
byte x;
void setup(){
   Serial.begin(9600);
   for(x=0;x<10;x++){
      if (x==4){
      break;        //continue;
      }
  Serial.println(x);
  }
}
void loop(){}
```

¿Qué pasaría si sustituimos la instrucción ***break*** por la sentencia ***continue***, dejando el resto del código anterior exactamente igual? Que veríamos una lista de números desde el 0 hasta el 9, excepto precisamente el 4. Esto es así porque la instrucción continue interrumpe la ejecución de la iteración en la cual "x" es igual a 4 (y por tanto, la instrucción Serial.println() correspondiente no se llega a ejecutar) pero continúa con la siguiente iteración del bucle de forma normal, en la cual a "x" se le asigna el valor 5.

1.2.8. Instrucción "goto": Ejemplo

```
void setup(){
Serial.begin(9600);
}
void loop()
{
 inicio:
Serial.println("Ciclo infinito");
delay(500);
goto inicio;
}
```

Ésta es la instrucción que te otorga más libertad, pero con la que más cuidado hay que tener, se escribe una etiqueta en cualquier parte del programa, y al usar la instrucción: goto etiqueta el programa inmediatamente saldrá de cualquier estructura de selección o iteración para dirigirse a la etiqueta seleccionada.

1.2.9. Instrucción "return":

Ésta instrucción nos devolverá algo de una función (que no sea void). Lo que escribamos por debajo de return en la función donde lo usemos, no se ejecutará nunca. Ya que, cuando llega a return, vuelve a la función que lo llamó.

Hemos visto las funciones " void " (quiere decir que no devuelven ningún valor) , pero también existen las que sí lo hacen, como las " int " , " float " ,etc (vamos, que ponemos el tipo de variable del valor que queremos retornar)

Ejemplo:

```
int suma(int numero_A, int numero_B);  // declaramos nuestra función
  void setup()
 {
  Serial.begin(9600);
  int resultado = 0;
  int primerNumero = 19;
  int segundoNumero = 2;
  resultado = suma( primerNumero, segundoNumero );  // aquí vemos que el nombre de los parámetros cambió, eso es porque realmente no opera con el nombre, si no, con las posiciones
  Serial.print("El resultado es: ");
  Serial.println(resultado); // imprimimos el resultado
 }
  void loop()
 {
 }
 int suma( int numero_A, int  numero_B)  // aqui tienen que tener los mismos nombres que arriba
 {
    return numero_A + numero_B; // aquí nos devolverá la suma de los dos operandos
 }
```

1.3. ELEMENTOS DE SINTAXIS:

Para evitar errores de compilación, tener en cuenta lo siguiente:

- ✓ **" ; "(**punto y coma).- El punto y coma es uno de los símbolos más usados en C, C++; y se usa con el fin de indicar el final de una línea de instrucción.
  Ejemplo: int a = 13;

- ✓ "**{ }**" (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.
  Ejemplos:

- ✓ **/*….*/** Bloque de comentario: Son áreas de texto que pueden abarcar más de una línea, lo que escribamos entre esos símbolos será ignorado por el programa.

- ✓ **//** Línea de Comentario: Funciona como el bloque de comentario, con la diferencia que solo será ignorado el texto que este a su derecha, al cambiar de línea perderá el efecto.

- ✓ **#define**.- permite al programador dar un nombre a un valor constante antes de compilar el programa. Constantes definidas en ARDUINO no ocupan ningún espacio en el chip. El compilador de reemplazar las referencias a estas constantes con el valor definido en tiempo de compilación.

  Ejemplo:

```
#define ledPin 3 // El compilador reemplazará cualquier mención de ledPin con el valor 3 cuando se ejecute el programa.
```

- ✓ **#include**.- se utiliza para incluir bibliotecas fuera de programa. Esto le da acceso al programador para un gran grupo de bibliotecas estándar de C (grupos de funciones pre-hechos), y también bibliotecas escrito especialmente para ARDUINO.
  Ejemplo:

```
#include <avr/pgmspace.h>

prog_uint16_t myConstants[] PROGMEM = {0, 21140, 702  , 9128,  0, 25764, 8456,
0,0,0,0,0,0,0,0,29810,8968,29762,29762,4500};
```

NOTA: Tenga en cuenta que "#include", al igual que "#define", no termina en ";" (punto y coma), y el compilador producirá mensajes de error si se le agrega.

Ejemplo de Aplicación:

Se pretende realizar un programa que controle el tiempo de estado alto y bajo de un led, es decir que le introduzcamos por el puerto serie un numero de 0 a 9, que indican 0mseg y 900mseg de tiempo de estado alto y estado bajo, y podamos controlar el parpadeo del mismo. Por lo que vamos a programar con el fin de que nos pida por pantalla que introduzcamos el tiempo de estado alto y el tiempo de estado bajo y una vez introducido que se realice el control del parpadeo del led.

```
//Inicialización de variables
char datosbyte;
byte f=0;
int x=0;
int y=0;
void setup() {
    Serial.begin(9600);          //Inicializamos el puerto serie
    pinMode(13, OUTPUT);    //Asignamos al pin 13 como salida donde conectaremos el LED
//Escribimos por puerto serie la primera vez
   Serial.println("Introduce el tiempo de estado ALTO:");
}
void loop () {
//Vamos a leer por puerto serie el valor de estado alto y bajo
   if (Serial.available() > 0) {       //Compruebo si el puerto serie esta accesible
      if(f==0){                              //Utilizamos una variable llamada bandera que nos
                                             realizara la función de conmutación de tiempo alto y bajo
        datosbyte=Serial.read();   //Leo los datos del puerto serie
        x=datosbyte-48;                 // paso de código ASCII a decimal
        Serial.print("El tiempo de estado ALTO es: ”);
        Serial.println(x,DEC);                  // Muestra el tiempo introducido
         Serial.println("Introduce el tiempo de estado BAJO:");
         f=1;
         }
   }
   if (Serial.available() > 0){
      if(f==1) {
        datosbyte=Serial.read();              //Voy leyendo los datos del puerto serie
        y=datosbyte-48;               //Convierto de código ASCII a decimal
        Serial.print("El tiempo de estado BAJO es: ");
        Serial.println(y,DEC);                  // Muestro el tiempo bajo introducido
        Serial.println("Introduce el tiempo de estado ALTO:");
        f=0;
    }
  }
digitalWrite(13, HIGH); // Pongo el Led en estado alto
int x1=1000*x;  // multiplico los segundos introducidos para pasarlos a milisegundos
delay(x1);
digitalWrite(13, LOW) ; // Pongo el Led en estado bajo
int y1=1000*y;              // Vuelvo a multiplicar
delay(y1);                  // Y mantengo en estado bajo
}
```

Nota: ASCII-48=DEC

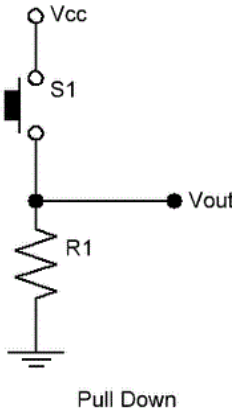
II. FUNCIONES:

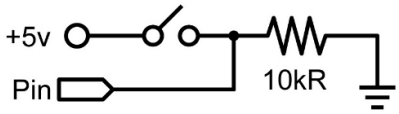
2.1. ENTRADAS Y SALIDAS DIGITALES:

2.1.1. pinMode():

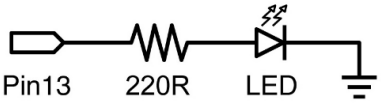
Se usa en **void setup()** para configurar un pin específico para comportarse como:

☑ ***Entrada (INPUT).-*** Pines configurados como INPUT están en un estado de alta impedancia. Pines de entrada hacen extremadamente pequeñas demandas en el circuito que están muestreo, esto equivale a una resistencia en serie de 100 MΩ en frente del pin. Esto significa que se necesita muy poca corriente para mover el pin de entrada de un estado a otro. Desactiva explícitamente estas resistencias "pull-ups" internas. Se aplican para tareas como la implementación de un sensor táctil, la lectura de un LED como un fotodiodo, o la lectura de un sensor analógico con un esquema tales como ***RCTime***[1].





☑ ***Salida (OUTPUT).-*** Los pines configurados como OUTPUT están en un estado de **baja impedancia** y pueden proporcionar 40mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un ***diodo LED*** (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.



☑ ***INPUT_PULLPUP.-*** Hay 20 kΩ de "resistencias pull-up" integradas en el chip Atmega que se puede acceder por el software, configurando el pinMode() como ***INPUT_PULLUP***. Esto invierte efectivamente el comportamiento del modo de entrada, donde HIGH significa que el sensor está apagado, y LOW significa que el sensor está encendido.

2.1.2. digitalWrite():

[1] RCTIME se puede utilizar para medir el tiempo de carga o descarga de un circuito de resistencia / condensador. Esto le permite medir la resistencia o capacitancia; utilizar sensores R o C como termistores o sensores de humedad o responder a la entrada del usuario a través de un potenciómetro. En un sentido más amplio, RCTIME también puede servir como un cronómetro rápido, preciso para eventos de muy corta duración.

Si el pin especificado en *digitalWrite()* está configurado como salida, usando la constante OUTPUT:

- La constante **HIGH** equivale a una señal de salida de hasta **40 mA** y de **5 V.**
- La constante **LOW** equivale a una señal de salida de **0 V**.

Si el pin está configurado como entrada usando la constante INPUT:

- Enviar un valor **HIGH** equivale a **activar la resistencia interna "pull-up"** en ese momento (es decir, es idéntico a usar directamente la constante INPUT_PULLUP).
- Enviar un valor **LOW** equivale a **desactivarl**a de nuevo. Esta función no tiene valor de retorno.

NOTA:

Es posible que el valor HIGH (5V) ofrecido por un pin digital de salida no sea suficiente para alimentar componentes de alto consumo (como motores o matrices de LEDs, entre otros), por lo que estos deberían alimentarse siempre con circuitería extra (evitando así además posibles daños en el pin por sobrecalentamiento).

2.1.3. digitalRead():

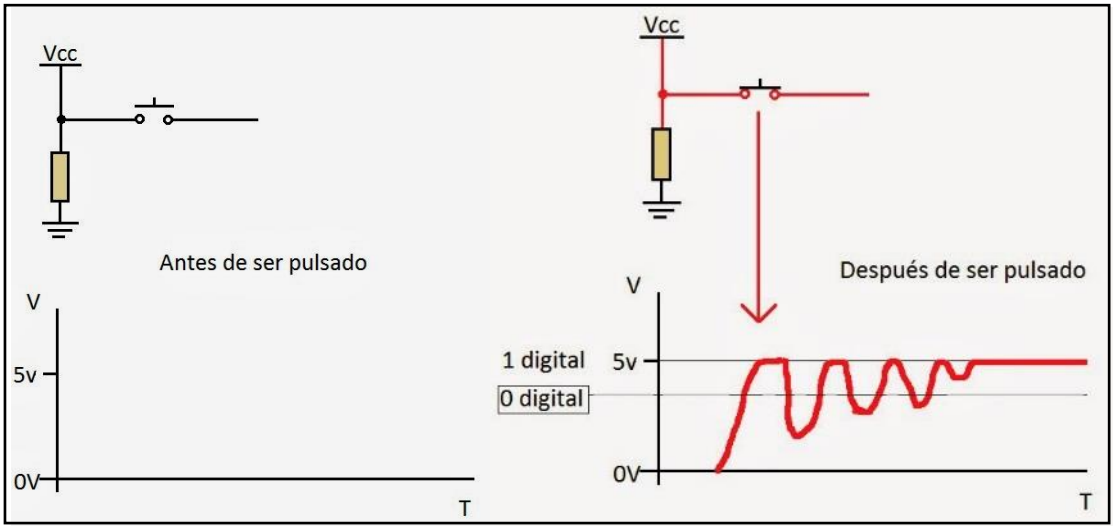
Devuelve el valor leído del pin digital (configurado como entrada mediante pinMode()) cuyo número se haya especificado como parámetro. Este valor de retorno es de tipo "int" y puede tener dos únicos: la constante **HIGH (1)** o **LOW (0)**.

Ejemplo de Subcapítulo 3.1. :

```
void setup(){
//Inicia la comunicación serial
  Serial.begin(9600);
//configura pin26 como ENTRADA y habilita la resistencia pull-up interna del ARDUINO
  pinMode(26,INPUT_PULLUP);
  pinMode(24, OUTPUT);
  }
void loop(){
//read the pushbutton value into a variable
  int sensorVal = digitalRead(26);
//print out the value of the pushbutton
  Serial.println(sensorVal);
// Keep in mind the pullup means the pushbutton's
// logic is inverted. It goes HIGH when it's open,
// and LOW when it's pressed. Turn on pin 13 when the
// button's pressed, and off when it's not:
 if (sensorVal == HIGH) {
 digitalWrite(24, LOW);
 }
 else {
   digitalWrite(24, HIGH);
   }
}
```

NOTA: Otra forma alternativa de activar la resistencia "pull-up", diferente de la descrita en el párrafo anterior, es utilizar la constante INPUT y además la función *digitalWrite()* de una forma muy concreta.

Gráficamente:



2.2. ENTRADAS Y SALIDAS ANALÓGICAS:

2.2.1. analogReference():

Asigna el modo para seleccionar el voltaje usado como referencia para el comando analogRead(), este valor será este será el voltaje máximo usado como referencia. El valor por defecto para las tarjetas es 5 voltios. Los posibles modos son:

- ☑ **DEFAULT:** a referencia analógica por defecto de 5 voltios o 3.3 voltios.
- ☑ I**NTERNAL1V1**: una referencia incorporada, igual a 1.1 volts en el ATmega168 o ATmega328 (Sólo ARDUINO Mega).
- ☑ **INTERNAL2V56:** Es una referencia de tensión interna de 2.56 voltios en el Atmega8 (Sólo ARDUINO Mega).
- ☑ **EXTERNAL:** Se usara una tensión de referencia externa que tendrá que ser conectada al pin AREF. Debe estar en el rango 0-5V solamente).

NOTA:

NO aplique al pin AREF un voltaje fuera del rango 0-5V. Siempre que se use un voltaje de referencia externo (cualquier cosa conectada al pin AREF) se debe llamar el comando analogReference(EXTERNAL) antes de llamar el comando analogRead(), un buen lugar para hacerlo es en la sección setup(). Si el pin AREF está conectado a una ***referencia externa*** no use ninguna de los demás modos de referencia de voltaje en su programa, ya que hará un corto circuito con el voltaje externo y puede causar daño permanente al microcontrolador de la tarjeta, además es recomendable que cuando se use la ***referencia externa*** se conecte al pin AREF usando una ***resistencia de 5K.*** Tenga en cuenta que esa resistencia afectara la tensión que se use como tensión de referencia ya que el pin AREF posee una resistencia interna de 32K. Ambas resistencias forman un divisor de tensión.

Por lo tanto, si por ejemplo se conecta una tensión de 2.5V con la resistencia de 5K la tensión que se utilice como tensión de referencia será de 2.2V ya que esa será la tensión que caiga en el pin AREF debido al divisor de tensión al que se ha hecho referencia. (Segunda Ley de Kirchhoff).

Uso del pin AREF

La tensión aplicada en el pin AREF será la que haga que el conversor A/D de su máxima lectura (1023) cuando lea una tensión igual a la aplicada en ese pin. Tensión por debajo de esa tensión conectada a AREF será escalada proporcionalmente, así cuando se usa la tensión de referencia por defecto (DEFAULT) el valor que nos devuelve una tensión de 2.5V en una entrada analógica será 512.

La configuración por defecto del Arduino es la de no tener nada conectado de forma externa al pin AREF. En este caso la configuración de la tensión de referencia será DEFAULT lo cual conecta a VCC (Alimentación positiva +5 V) de forma interna al pin AREF. Este pin es un pin de baja impedancia (mucha corriente) por lo que si usando la configuración DEFAULT de la tensión de referencia se conecta otra tensión que no sea la que posee AVCC, podría dañar el chip ATMEGA.

El pin AREF también puede conectarse de forma interna a una fuente de referencia de 1.1 voltios (o 2.54 en los Atmega8) usando el comando analogReference (INTERNAL).Con esta configuración las tensiones aplicada a los pines analógicos que sean iguales a la tensión de referencia interna o superior nos devolverán 1023 cuando se lleva a cabo su lectura con la función analogRead(). Tensiones más bajas devolverán valores proporcionales, por ejemplo, si tenemos una tensión de 0.55 voltios en el pin analógico, nos devolverá 512.

$$Vreferencia = \frac{R\,\text{int}\,erna}{R\,\text{int}\,erna + \text{Re}\,xterna} \times Vin$$

Donde:

- Vin = Tensión que introducimos a Vref
- Rinterna = Resistencia interna de Arduino de 32KΩ
- Rexterna = Resistencia mediante la cual alimentamos Vref
- Vrefencia = La tensión sobre que tomará referencia nuestro programa.

De manera que si por ejemplo, estamos introduciendo una tensión de +3v a través de una resistencia de 5KΩ, la tensión real de referencia en nuestro Arduino será de:

$$Vrefencia = \frac{32k}{32k+5k} * 3V \cong 2.59$$

2.2.2. analogRead():

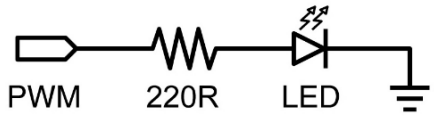
Lee el valor de un determinado pin definido como entrada analógica, por defecto, nuestra placa Arduino nos realizará una conversión analógico-digital para toda señal (40mA máximo) de 0v a 5v con una resolución de 10 bit. Esto significa que convertirá tensiones entre 0 y 5 voltios a un número entero entre 0 y 1023. Esto proporciona una resolución en la lectura de: 5 voltios / 1024 unidades, es decir, 0.0049 voltios (4.9mV) por unidad.

```
int valorPot=0;
void setup(){
  Serial.begin(9600);
  }
void loop(){
  valorPot=analogRead(2);
  Serial.println(valorPot);
  delay(100);
}
```

Ejemplo:
Veremos por el "Serial monitor" las lecturas realizadas a través del pin analógico 2, que no son más que valores entre 0 y 1023. Notar que no se ha utilizado la función *pinMode()* como hasta ahora porque los pines-hembra analógicos de la placa ARDUINO solo pueden ser de entrada.

2.2.3. analogWrite() – PWM:

Envía un valor de tipo "byte" (especificado como segundo parámetro) que representa una señal PWM, a un pin digital configurado como OUTPUT (especificado como primer parámetro). Después de llamar a analogWrite (), el pin generará una onda cuadrada constante del ciclo de trabajo especificado hasta la siguiente llamada a analogWrite () (o una llamada a digitalRead () o digitalWrite () en el mismo pin). La frecuencia de la señal PWM en la mayoría de los pins es de aproximadamente 490 Hz y algunas placas de ARDUINO sus pines tienen una frecuencia de 980 Hz. Puede ser usado para controlar la luminosidad de un LED o la velocidad de un motor.



Ejemplo:

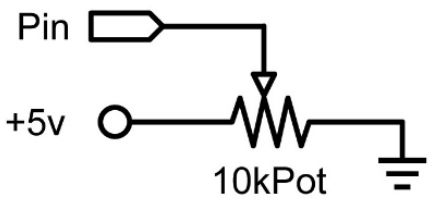
```
int brillo = 0;
void setup(){
   pinMode(9, OUTPUT); //El LED está conectado al pin 9 (PWM)
   }
void loop(){
//Incrementa el brillo (de mínimo a máximo)
   for(brillo = 0 ; brillo<= 255; brillo=brillo+5) {
      analogWrite(9, brillo);
      delay(30);
   }
//Disminuye el brillo (de máximo a mínimo)
  for(brillo = 255; brillo>=0; brillo=brillo-5) {
      analogWrite(9, brillo);
     delay(30);
  }
}
```

Ejemplo:

Nuestro diodo LED variará su intensidad lumíinoza en función del valor que esté aportando el potenciómetro a nuestra entrada analógica, esto es el llamado efecto fading (desvanecimiento).

```
int pinSensor = A0;    // Entrada para el potenciómetro.
int pinLed = 2;        // Seleccionamos pin para el Led.
int valorSensor = 0;   // variable para el valor del sensor.
void setup() {
  pinMode(pinLed, OUTPUT); // Establecemos el pin como salida.
}
void loop() {
  // Leemos el valor del sensor y lo almacenamos:
  valorSensor = analogRead(pinSensor);
  // Establecemos el valor analógico para la salida PWM
  analogWrite(pinLed, valorSensor / 4);
//Nos permitirá emular una señal analógica a partir de una digital en nuestros circuitos
    delay(30);
}
```

Entrada del potenciómetro:



NOTA:

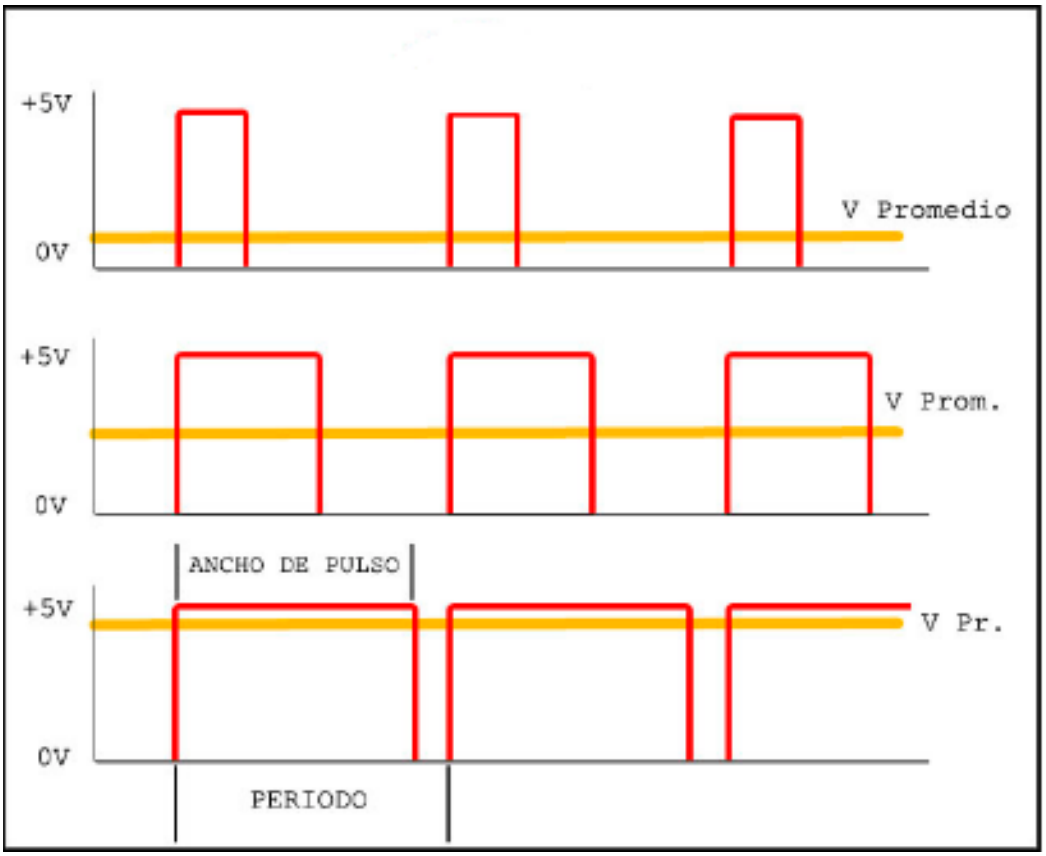
Recordemos que una señal **PWM (Pulse Width Modultation o Modulación de Ancho de Pulso)**, es una señal digital cuadrada que simula ser una señal analógica.
El valor simulado de la señal analógica dependerá de la duración que tenga el pulso digital (es decir, el valor HIGH de la señal PWM).

- ☑ Si el segundo parámetro de esta función vale 0, significa que su pulso no dura nada (es decir, no hay señal) y por tanto su valor analógico "simulado" será el mínimo (0V).

- ☑ Si vale 255 (que es el máximo valor posible, ya que las salidas PWM tienen una resolución de 8 bits, y por tanto, solo pueden ofrecer hasta 2^8=256 valores diferentes de 0 a 255, pues), significa que su pulso dura todo el período de la señal (es decir, es una señal continua) y por tanto su valor analógico "simulado" será el máximo ofrecido por la placa (5 V).

- ☑ Cualquier otro valor entre estos dos extremos (0 y 255) implica un pulso de una longitud intermedia (por ejemplo, el valor 128 generará una onda cuadrada cuyo pulso es de la misma longitud que la de su estado bajo) y por tanto, un valor analógico "simulado" intermedio (en el caso anterior, 2,5 V).

Esta imagen describe la acción de un pin PWM.



Ejemplo del Subcapítulo 3.2. :

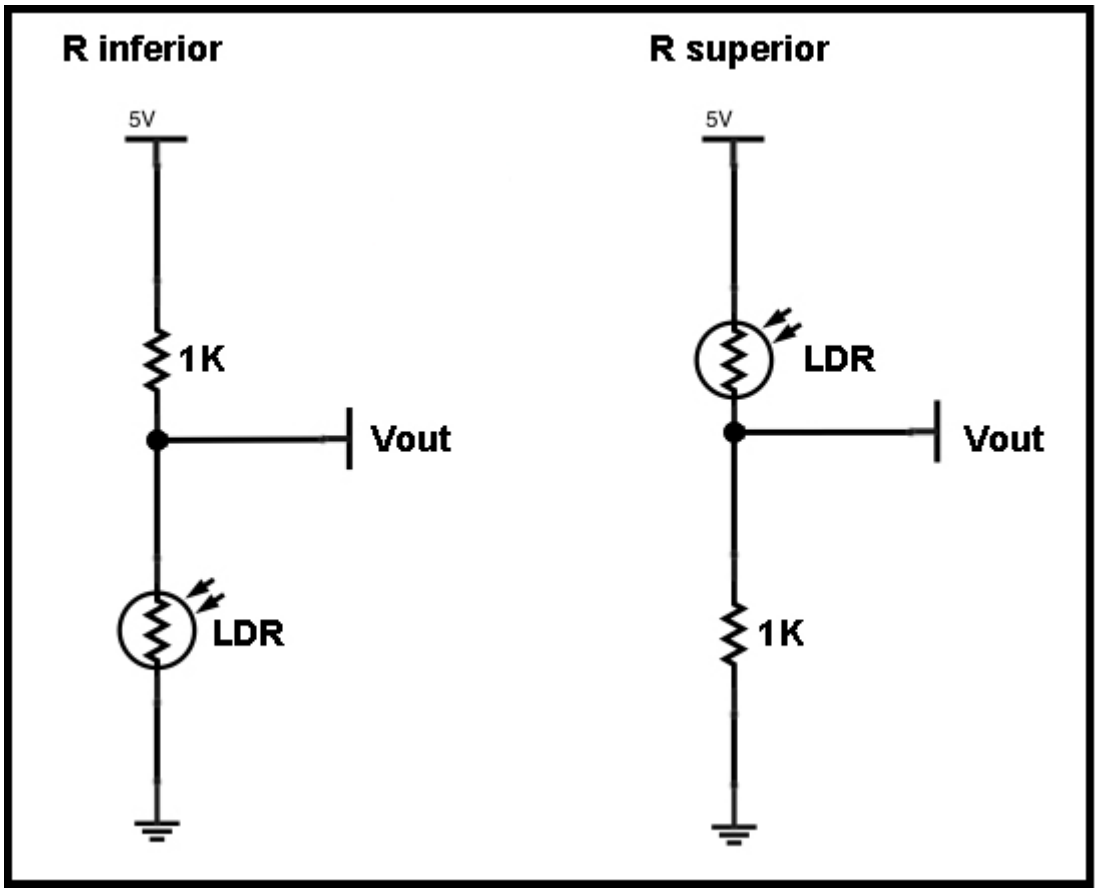
Se va a utilizar un LDR (Light Dependent Resistor o resistencia dependiente de la luz) para simular una hipotética compensación lumínica de 5 niveles, es decir, a través de una resistencia que varía su valor dependiendo de la luz recibida, aprovecharemos dicha variación para hacer un programa que nos encienda o apague una serie de LED dependiendo de si hay más luz o menos luz.

Mayor Cantidad de Luz reciba → Menor Resistencia

Además, vamos a colocar un potenciómetro para regular el umbral de luz mínima, a partir del cual, comenzará a funcionar nuestro circuito de luz artificial para que sea adaptable a cualquier entorno.

Si añadimos una resistencia más, podemos utilizar el LDR para hacer el ya conocido ***divisor de tensión*** de donde sacaremos la señal para conectar a nuestra entrada analógica de Arduino.

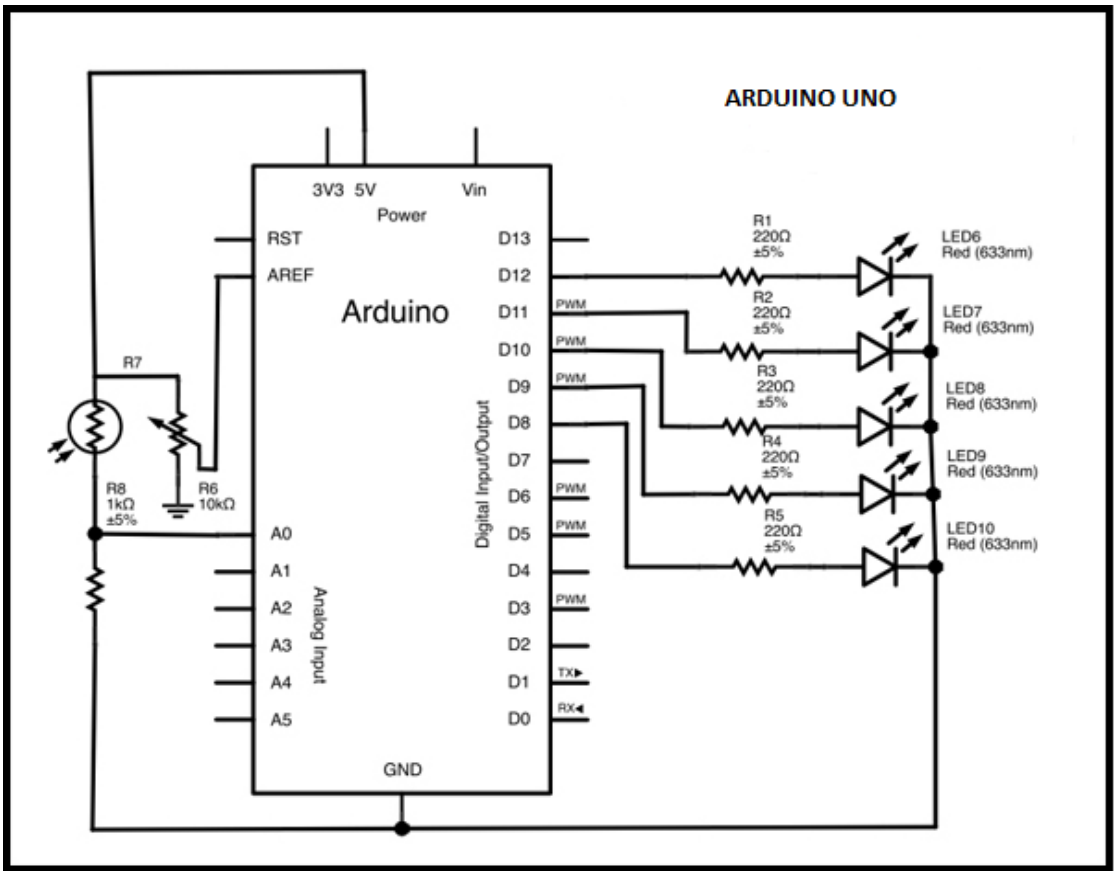
Podemos conectarlo de dos maneras diferentes:



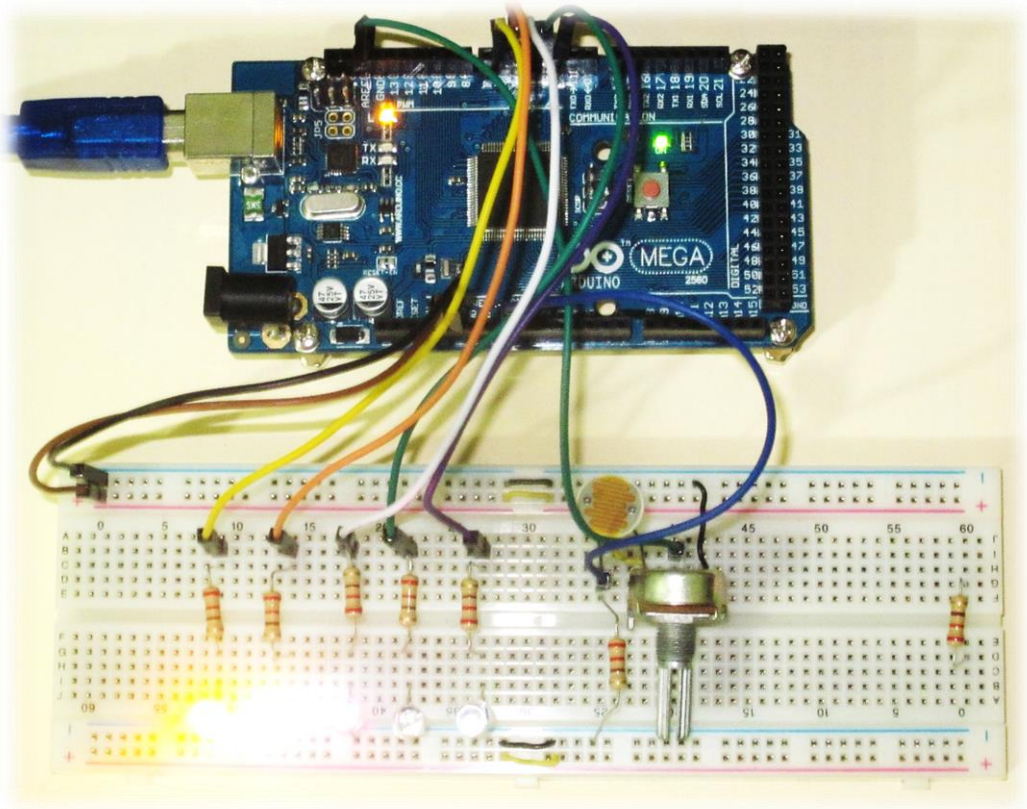
- Si utilizamos el LDR como ***resistencia inferior del divisor de tensión, nos dará la tensión máxima cuando tengamos el LDR en plena oscuridad***, ya que estará oponiendo el máximo de su resistencia al paso de la corriente derivándose esta por Vout al completo

- Si lo utilizamos como ***resistencia superior,*** el resultado será el inverso, ***tendremos la tensión máxima cuando esté completamente iluminado***, ya que se comportará prácticamente como un ***cortocircuito***, con una resistencia de 50Ω o 100Ω.

Conexiones:



Esquema gráfico:



Programa:

```
/*Conectamos una foto-resistencia a la entrada analógica para controlar cinco salidas en
función de la luz ambiente.*/

//Aquí almacenamos los datos recogidos del LDR:
int valorLDR = 0;

//Decimos que pines vamos a utilizar para LED
int pinLed1 = 6;
int pinLed2 = 5;
int pinLed3 = 4;
int pinLed4 = 3;
int pinLed5 = 2;
int pinLDR = 0;  //pin para la LDR es el numero 0, A0

void setup()
{
 //Establecemos como salida los pines para LED
 pinMode(pinLed1, OUTPUT);
 pinMode(pinLed2, OUTPUT);
 pinMode(pinLed3, OUTPUT);
 pinMode(pinLed4, OUTPUT);
 pinMode(pinLed5, OUTPUT);
 analogReference(EXTERNAL);  //Le decimos que vamos a usar una referencia externa
}
void loop()
{
   valorLDR = analogRead(pinLDR);  //Guardamos el valor leido en una variable
 //Y comenzamos las comparaciones:
 if(valorLDR >= 1023)
 {
  digitalWrite(pinLed1, LOW);
  digitalWrite(pinLed2, LOW);
  digitalWrite(pinLed3, LOW);
  digitalWrite(pinLed4, LOW);
  digitalWrite(pinLed5, LOW);
 }
 else if((valorLDR >= 823) & (valorLDR < 1023))
 {
  digitalWrite(pinLed1, HIGH);
  digitalWrite(pinLed2, LOW);
  digitalWrite(pinLed3, LOW);
  digitalWrite(pinLed4, LOW);
  digitalWrite(pinLed5, LOW);
 }
```

```
else if((valorLDR >= 623) & (valorLDR < 823))
 {
  digitalWrite(pinLed1, HIGH);
  digitalWrite(pinLed2, HIGH);
  digitalWrite(pinLed3, LOW);
  digitalWrite(pinLed4, LOW);
  digitalWrite(pinLed5, LOW);
 }
 else if((valorLDR >= 423) & (valorLDR < 623))
 {
  digitalWrite(pinLed1, HIGH);
  digitalWrite(pinLed2, HIGH);
  digitalWrite(pinLed3, HIGH);
  digitalWrite(pinLed4, LOW);
  digitalWrite(pinLed5, LOW);
 }
 else  if((valorLDR >= 223) & (valorLDR < 423))
 {
  digitalWrite(pinLed1, HIGH);
  digitalWrite(pinLed2, HIGH);
  digitalWrite(pinLed3, HIGH);
  digitalWrite(pinLed4, HIGH);
  digitalWrite(pinLed5, HIGH);
 }
 else
 {
  digitalWrite(pinLed1, HIGH);
  digitalWrite(pinLed2, HIGH);
  digitalWrite(pinLed3, HIGH);
  digitalWrite(pinLed4, HIGH);
  digitalWrite(pinLed5, HIGH);
 }
}
```

2.3. Tiempo:

2.3.1. millis():

Devuelve el número de milisegundos (ms) desde que la placa Arduino empezó a ejecutar el sketch actual. Este número se reseteará a cero aproximadamente después de 50 días (cuando su valor supere el máximo permitido por su tipo, que es “unsigned long”). No tiene parámetros.

2.3.2. micros():

Devuelve el número de microsegundos (µs) desde que la placa Arduino empezó a ejecutar el sketch actual. Este número –de tipo “unsigned long”– se reseteará a cero aproximadamente después de 70 minutos. Esta instrucción tiene una resolución de 4 µs (es decir, que el valor retornado es siempre un múltiplo de cuatro).

2.3.3. delay():

Pausa el sketch durante la cantidad de milisegundos especificados como parámetro –de tipo "unsigned long"–. No tiene valor de retorno.

2.3.4. delayMicroseconds():

Pausa el sketch durante la cantidad de microsegundos especificados como parámetro –de tipo "unsigned long"– . Actualmente el máximo valor que se puede utilizar con precisión es de 16383. El mínimo valor que se puede utilizar con precisión es de 3 µs. No tiene valor de retorno.

2.4. Comunicación:

2.4.1. Serial:

El ARDUINO se comunica por serie vía USB con nuestro ordenador a través del chip FTDI. La comunicación serial se puede utilizar para hacer un debugging (saber lo que está pasando en nuestro programa) o para comunicarnos con otros programas.

Instrucciones Seriales:

- **Serial.begin(speed):**

Abre el canal serie para que pueda empezar la comunicación por él. Por tanto, su ejecución es imprescindible antes de realizar cualquier transmisión por dicho canal. Por eso normalmente se suele escribir dentro de la sección "void setup()". Además, mediante su único parámetro –de tipo "long" y obligatorio–. Para comunicarse con el ordenador, utilice una de estas velocidades: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 o 115200.

- **Serial.print(val, [format]):**

Envía a través del canal serie un dato (especificado como parámetro) desde el microcontrolador hacia el exterior.

- val: el valor a imprimir - de cualquier tipo
- format: especifica la base (formato) a usar; los valores permitidos son:
  - ✓ BYTE, BIN (binarios o base 2)
  - ✓ OCT (octales o base 8)
  - ✓ DEC (decimales o base 10)
  - ✓ HEX (hexadecimales o base 16).

Para números de coma flotante, este parámetro especifica el número de posiciones decimales a usar. Por defecto son dos.

Ejemplo:

Serial.print(78) imprime "78"	Serial.print(78, DEC) imprime "78"
Serial.print('N') imprime "N"	Serial.println(1.23456, 0) imprime "1.23"
Serial.print(78, BYTE) imprime "N"	Serial.println(1.23456, 2) imprime "1.23"

- **Serial.println(val, [format])**:

Hace exactamente lo mismo que *Serial.print(),* pero además, añade automáticamente al final de los datos enviados dos caracteres extra: el de retorno de carro (código ASCII nº 13, o '\r') y el de nueva línea (código ASCII nº 10, o \n). La consecuencia es que al final de la ejecución de *Serial.println()* se efectúa un salto de línea. Tiene los mismos parámetros y los mismos valores de retorno que *Serial.print().*

```
int cont=0;
void setup (){
  Serial.begin(9600); //iniciamos el puerto de serie
}
void loop () {
//Imprimimos el valor del contador
  Serial.print("Contador: ");
  Serial.println(cont);
  cont++; //incrementamos el contador y esperamos  ½ segundo
  delay(500);
}
```

- **Serial.flush()**:

La transmisión de los datos realizada por *Serial.print()* es asíncrona. Eso significa que nuestro sketch pasará a la siguiente instrucción y seguirá ejecutándose sin esperar a que empiece a realizarse el envío de los datos. Si este comportamiento no es el deseado, se puede añadir justo después de *Serial.print()* la instrucción *Serial.flush()* –que no tiene ningún parámetro ni devuelve ningún valor de retorno–, instrucción que espera hasta que la transmisión de los datos sea completa para continuar la ejecución del sketch.

- available()
- If(Serial)
- end()
- fin()
- finUntil()
- parseFloat()
- parseInt()
- peek()
- Serial.read()
- readBytes()
- readBytesUntil()
- write()

- SetTimeout
- SerialEvent()

Sintaxis:

- ✓ DEC // Imprime la información en formato decimal.
- ✓ '\t' Un tabulador de 8 espacios de texto.
- ✓ '\n' Cambio de línea, ASCII 10.
- ✓ '\r' Retorno de carro ASCII 13.

Ejemplos:

```
void setup()                    // Se ejecuta una zona vez, cuando el programa inicia
{
 Serial.begin(9600);            // Velocidad de la transmisión y apertura del puerto
 Serial.println("Hola Mundo!"); // Imprime Hola Mundo, en el monitor serial
}
void loop()                     // Se ejecuta indefinidamente
{
                                // No hace nada
}
```

NOTA: La velocidad para comunicarnos con el Monitor serial es 9600 Baudios. Con otros dispositivos puede ser otra, por ejemplo el protocolo MIDI sería a 31250 Baudios.

- Baudios: representa el número de símbolos por segundo en un medio de transmisión digital. Un baudio puede contener varios bits.