

Taller de Arduino



La placa mágica

Ejercicio 1: Blink

Ejercicio 2: PulsadorDig

Ejercicio 3: PulsadorDigSerial

Ejercicio 4: LDR-Puls

Ejercicio 5: Servo

Ejercicio 6: ServoPot

Ejercicio 7: Rele-Puls

Ejercicio 8: Rele-LDR

Ejercicio 9: Bluetooth

Ejercicio 10: Infrarrojo

Ejercicio 11: RF-433 mhz

Ejercicio 12: Wifi-Led-Webserver

Ejercicio 13: Wifi-Led-WebserverAP

Ejercicio 14: Wifi-ADC-WebserverAP

Ejercicio 1: Blink

Es el "Hola mundo" de **Arduino**, el sketch (programa) más sencillo que podemos cargar y ejecutar en nuestra **Arduino**. Se trata de hacer que un led se encienda y se apague según unos intervalos definidos en el código del sketch.

Todo lo que necesitas para seguir este primer ejercicio es una placa Arduino Nano y un Led.

¿Porque lo llamamos **sketch** y no programa? pues por que el IDE de **Arduino** viene de **Processing** y en este lenguaje de programación enfocado al mundo gráfico cada código es considerado un boceto, en inglés "sketch" y al heredar **Arduino** el IDE de **Processing** y su forma de guardar el código generado (por eso nuestros sketches se guardan en formato .pde) ha heredado también el nombre para los "programas".

Al hacer la traducción de la web Arduino.cc al español en julio de 2010 hubo que proponer un conjunto de normas de estilo para que los textos traducidos fuesen lo más uniformes posibles. Dentro de esas normas de estilo habían algunos términos que se acordó no traducir, uno de ellos era "sketch" para recordar la procedencia del IDE de **Arduino**.

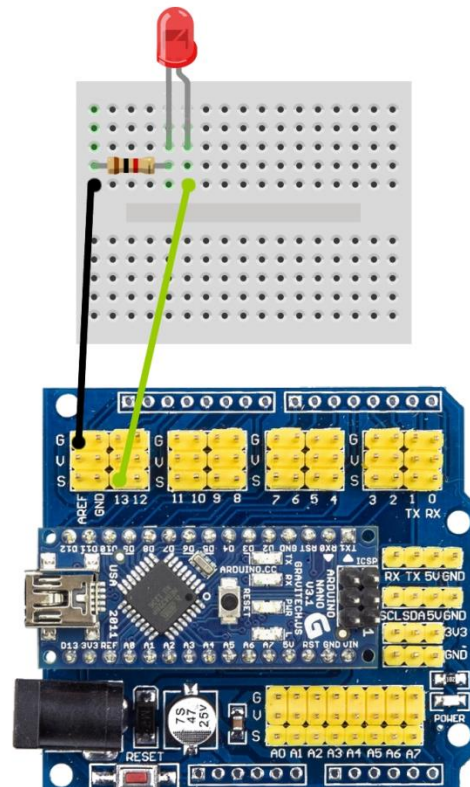
Las Arduino NANO están diseñadas para que hacer parpadear un LED sea muy fácil usando el pin digital 13. Algunas (como la Diecimila, la LilyPad y la UNO) tienen el LED directamente incorporado en la placa. En la mayoría de las otras (como en la Mini y la BT), existe una resistencia de 1KOhm en el pin permitiéndote conectar un LED directamente (para conectar un LED a otro pin digital, deberías usar una resistencia externa).

Circuito:

En la ilustración se utiliza una resistencia de 220 entre el pin GND y el cátodo.

Los LEDs tienen polaridad, lo que significa que solo si los orientas debidamente. Generalmente el terminal es el positivo y deberá estar conectado al pin 13. El otro debe conectarse con la tierra (GND); adicionalmente el bulbo del LED tiene un borde plano extremo. Si el LED no enciende, trata de conectarlo opuesto, intercambiando los terminales de posición (dañarás el LED si lo conectas en sentido opuesto por de tiempo corto).

Este ejemplo nos sirve también como primera aproximación a la estructura de un sketch



Ohms

encenderán
más largo
terminal

en su
de manera
(no
un periodo

Este es el código del sketch Blink (parpadeo), con los comentarios ampliados:

Comentarios iniciales descripción breve del proyecto.

```
/*  
BLINK  
Enciende un LED durante un segundo, lo apaga durante otro segundo, repetidamente.  
*/  
Configuramos el pin 13 como una Salida (OUTPUT)  
void setup() {  
pinMode(13, OUTPUT); // Establece un pin digital como salida. El Pin 13 tiene un LED conectado a el.  
}  
Ejecutamos el contenido loop() continuamente  
void loop() {  
digitalWrite(13, HIGH); // Pone el pin 13 a nivel alto (5v). Enciende el LED  
delay(1000);           // espera un segundo (1000 milisegundos)  
digitalWrite(13, LOW);  // Pone el pin 13 a nivel bajo (0v). apaga el LED  
delay(1000);           // espera un segundo (1000 milisegundos)  
}
```

Podemos variar el valor de **delay** y comprobar como cambian los tiempos de apagado y encendido

Ejercicio 2-PulsadorDig

En esta ocasión vamos a utilizar un pulsador para encender y apagar un LED. Comenzamos creando las variables que contendrán los números de los pines:

```
int pulsador = 2;  
int led= 13;
```

En el setup() los configuraremos como entradas o como según corresponda:

```
void setup()  
{  
  pinMode(pulsador, INPUT);  
  pinMode(led, OUTPUT);  
}
```

Ahora vamos al loop() en el que utilizaremos un comando condicional, el **if**, para que el LED amarillo se encienda si pulsamos el pulsador y se apague si no lo accionamos.

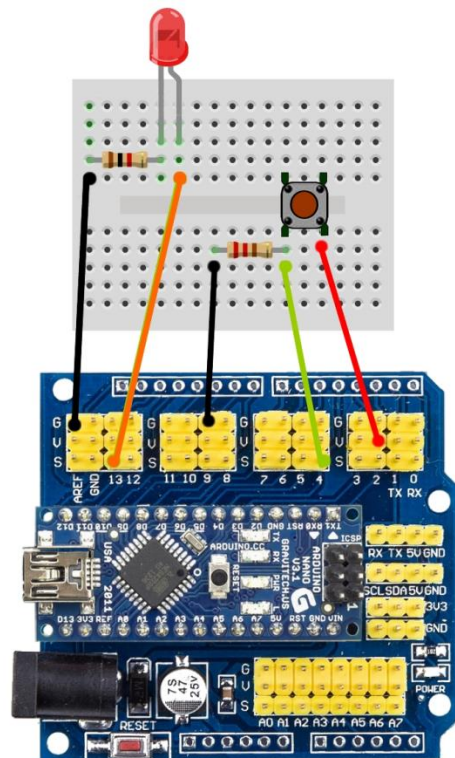
El comando if tiene una sintaxis así:

```
void loop()  
{  
  if (condicion)  
  {  
    // código que se ejecuta si se cumple la condición  
  }  
  else  
  {  
    // código que se ejecuta si no se cumple la condición  
  }  
}
```

Por lo que para adaptarlo a nuestro ejemplo lo deberemos dejar de la siguiente manera. **ATENCIÓN** a la condición:

```
void loop()  
{  
  if (digitalRead(pulsador) == HIGH)  
  {  
    digitalWrite(led, HIGH);  
  }  
  else  
  {  
    digitalWrite(led, LOW);  
  }  
}
```

Hemos visto un comando más **digitalRead()**, que sirve para hacer una lectura de un pin digital, nos devolverá un HIGH o un LOW según el pin esté activado o no.



salidas,

Código completo:

```
int pulsador = 2;
int led= 13;

void setup()
{
  pinMode(pulsador, INPUT);
  pinMode(led, OUTPUT);
}
void loop()
{
  if (digitalRead(pulsador) == HIGH)
  {
    digitalWrite(led, HIGH);
  }
  else
  {
    digitalWrite(led, LOW);
  }
}
```

Ejercicio 3-PulsDigSerial

Aprovechando que tenemos un pulsador para encender y apagar un LED leeremos el estado del pulsador a través del **Serial**.

Para ello introduciremos dos líneas nuevas de comando:

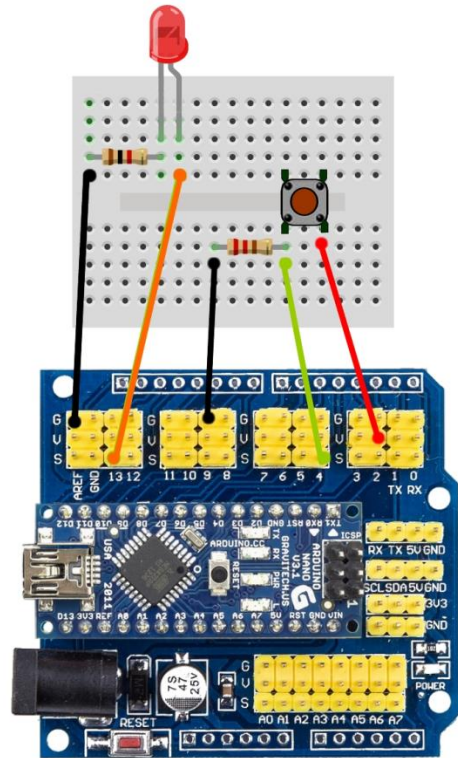
En el **setup()** inicializaremos el **Serial** a una velocidad transmisión de 9600 baudios

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

Y en **loop()** mostraremos el valor de (pulsador) enviándolo estado de éste por el **serial**.

```
Serial.println(digitalRead (pulsador));
```

Utilizando **Serial.println()** leeremos el estado de nuestras variables, entradas, salidas etc, con lo que nos ayudara a ver y entender en qué estado se encuentra programa.



de

el

nuestro

Código completo:

//Práctica del encendido de un led mediante un pulsador y lectura del estado del pulsador a traves del Puerto Serie

```
int pulsador = 2;
int led= 13;

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

    pinMode(pulsador, INPUT);
    pinMode(led, OUTPUT);
}
void loop()
{

    if (digitalRead(pulsador) == HIGH)
    {
        digitalWrite(led, HIGH);
    }
    else
    {
        digitalWrite(led, LOW);
    }

    Serial.println(digitalRead(pulsador));

}
```

Ejercicio 4: LDR-Puls

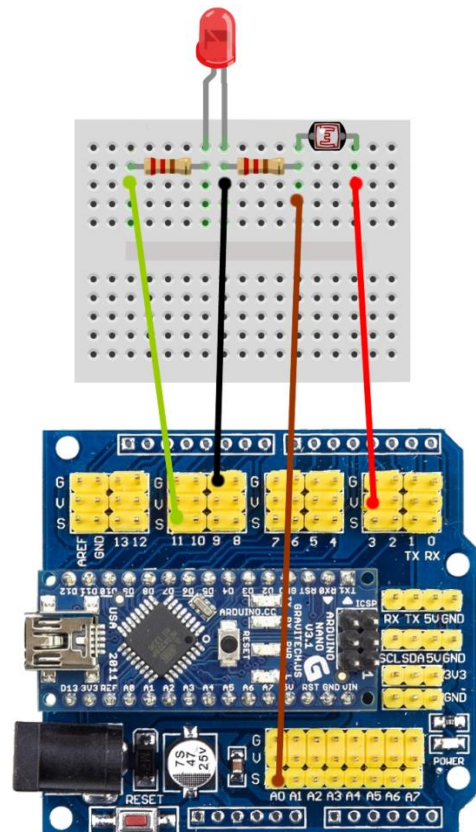
Uso de una fotoresistencia como sensor de luz

Este es un ejercicio muy sencillo en el que veremos cómo obtener lecturas analógicas de un sensor y darles salida por el monitor del puerto serie para verlas.

Necesitaremos una placa Arduino NANO y una resistencia sensible a la luz o fotoresistencia (LDR) que nos servirá como interruptor.

También utilizaremos un led para que nos de una salida (ON OFF)

Circuito:



1ªParte: Lectura del valor analógico de laLDR

Comenzamos creando las variables que contendrán los números de los pines:

```
int LDRPin = A0; // Pin para la fotorresistencia
```

En el **setup()** las configuraremos como entradas o como salidas, según corresponda e inicializaremos el puerto Serie:

```
void setup()
{
  Serial.begin(9600); // Inicia la comunicacion serie a 9600 baudios

  pinMode(LDRPin, INPUT);
}
```

Ahora vamos al **loop()**. Primero leemos el valor que nos da la fotorresistencia y lo mostramos por el puerto serie

```
void loop()
{
  Serial.println(analogRead(LDRPin));

  delay(100);
}
```

Tapando la LDR con la mano veremos la variación del valor analógico. Tomaremos como referencia un valor (**umbral**) en que creamos que debería apagarse nuestro led.

2ª Parte: Encendido y apagado del LED con LDR

Crearemos una variable **umbral**. Seguido utilizamos ese **umbral** para encender y apagar el Led en función de la cantidad de luz que recibe la LDR.

```
int LEDPin = 13; // Asigna LEDPin al pin 13
int LDRPin = A0; // Asigna LDRPin al pin A0
int umbral = 100; //este valor lo obtenemos poniendo la mano sobre la LDR
```

```
void setup()
{
  pinMode(LEDPin, OUTPUT); // Establece un pin digital como salida.
  pinMode(LDRPin, INPUT); // Establece un pin analógico como entrada.
}
```

```
void loop()
{
  if (analogRead(LDRPin)< umbral) { // compara el valor de la LDR con en que hemos asignado en la
  variable ðumbralö
    digitalWrite(LEDPin, HIGH);
  }
  else {
    digitalWrite(LEDPin, LOW);
  }
}
```

Ejercicio 5: Servo

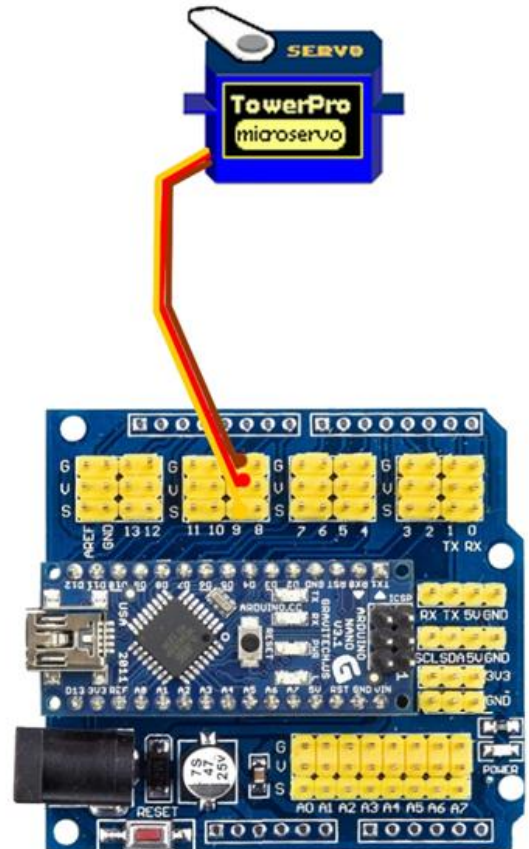
Vamos a empezar a utilizar un servo motor. En este ejercicio moveremos el servo a lo largo de sus 180°

Circuito:

Los servomotores tienen tres cables: alimentación (rojo), GND (que puede ser negro o marrón) y señal (que puede ser blanco o naranja). El cable de alimentación debe ser conectado al pin 5V de la placa Arduino. El cable GND debe ser conectado al pin GND de la placa Arduino. El pin de señal se conecta al pin 9 de la placa Arduino UNO para este ejercicio.

AVISO: En este ejercicio vamos a utilizar un micro servo de 9g cuyo consumo es tolerado por la placa, para otros montajes en los que el servo utilizado sea mayor NO se pueden conectar la tierra y la alimentación del servo a la placa Arduino si no directamente a la fuente de alimentación o a una

En este ejercicio vamos a ver como se incluye una librería para hacernos más fácil el interactuar con un dispositivo, en este caso el servo.



Código:

Utilizamos la librería del servo para facilitar el manejo, y la cargamos

```
#include <Servo.h>
```

Crea el objeto "myservo", y declaramos una variable como integer dándole valor inicial "0"

```
Servo myservo;
```

```
int pos = 0; // Asignamos un valor inicial a la variable "pos"
```

En el setup asignamos al objeto "myservo" al pin (9) del arduino

```
void setup() {
```

```
  myservo.attach(9); }
```

En el loop() moveremos el servo, grado a grado a la velocidad indicada en el "delay". Una vez alcanzados los 180º invertimos el sentido de giro.

```
void loop() {
```

```
  for (pos = 0; pos <= 180; pos += 1) { // Le damos valor "0" a la variable pos; verificamos que es igual o menor que 180º y le sumamos "1" a "pos".
```

```
    myservo.write(pos);          // indicamos al servo que se mueva a pos.
```

```
    delay(15);                  // esperamos 15 ms a que el servo vaya a su nueva posición.
```

```
  }
```

```
  for (pos = 180; pos >= 0; pos -= 1) { // Le damos valor "180" a la variable pos; verificamos que es igual o mayor que 180º y le restamos "1" a "pos".
```

```
    myservo.write(pos);          // le indicamos al servo que se mueva a pos.
```

```
    delay(15);                  // esperamos 15 ms a que el servo vaya a su nueva posición.
```

```
  }
```

```
}
```

Ahora podremos probar a cambiar los límites de los grados y el tiempo para ver lo que ocurre.

Ejercicio 6: servosPot

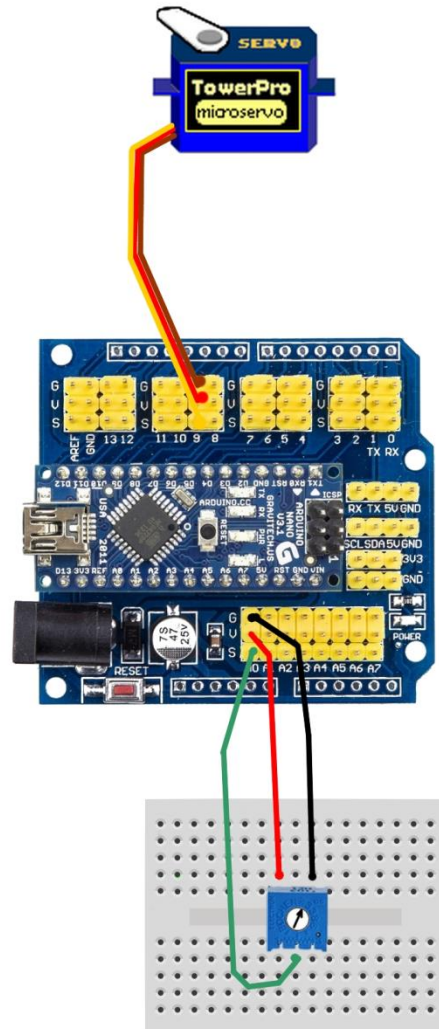
En este ejercicio vamos a intentar mover los 180° del servo motor a nuestro gusto, a través de un potenciómetro conectado a una entrada analógica de nuestro arduino

Circuito:

El potenciómetro debe ser conectado de manera que los dos pines exteriores estén conectado a alimentación (+5V) y a GND, el pin central lo conectaremos a la entrada analógica A0 en el Arduino.

En este ejercicio también vamos a incluir una librería para hacernos más fácil el interactuar con un dispositivo, en este caso el servo.

Con un código tan sencillo haremos que el servo se mueva según movamos el potenciómetro de modo que al estar el potenciómetro en el límite inferior de su recorrido el servo estará en la posición de 0°, si centramos el potenciómetro el servo estará en la posición de 90° y si llevamos el potenciómetro a su límite superior el servo alcanzará la posición de 180°.



Code:

// Controlando la posición de un servo usando un potenciómetro (resistencia variable)

Cargamos la librería <Servo.h> , le al potenciómetro un pin analógico (A0)

```
#include <Servo.h>
```

Creamos el objeto myservo, asignamos pines y creamos variables

```
Servo myservo; // crea el objeto "myservo"  
int potpin = A0; // Asigna al pin analógico A0 a potenciómetro  
int val; // variable para leer el valor del potenciómetro.
```

En el setup asignamos la variable "myservo" al pin (9) del arduino

```
void setup() {  
  myservo.attach(9); // Asigna el objeto "myservo" al pin (9)  
}
```

Leeremos el valor del potenciómetro, lo escalamos de 0, 1023 a 0, 180 y le enviamos al servo la posición

```
void loop() {  
  val = analogRead(potpin); // Lee el valor del potenciómetro (entre 0 y 1023)  
  val = map(val, 0, 1023, 0, 180); // escala o mapea el valor de val (valor entre 0 y 180)  
  myservo.write(val); // envía a "myservo" la posición de "val" escalada entre 0 y 180.  
  delay(15); // esperamos 15 ms a que el servo vaya a su nueva posición.  
}
```

Imaginemos que tenemos hecho nuestro Sketch, lo subimos a la placa y, a pesar de estar correctamente escrito (no da errores el compilador), no hace lo que nosotros queremos. El error puede estar en la forma en que trabajamos con los valores de forma interna y eso no lo podemos ver a menos que los mostremos por el monitor serie.

Otro caso puede ser que, una vez realizado nuestro montaje, queramos que el servo comience su recorrido desde una posición determinada y no sabemos cual es. Entonces podemos utilizar el monitor serie para calibrar nuestro servo y su recorrido.

Para ello activaremos el puerto serie incluyendo en el `setup()` la línea:

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

Esta instrucción inicializará la comunicación con el ordenador a través del puerto serie.

Luego iremos al final del Sketch, justo a la última línea, donde está el comando `delay(15)` y antes de ella insertaremos el comando que dará salida a los datos que veremos en el monitor serie:

```
Serial.println(val);
```

Por último, y para darle tiempo al monitor serie a mostrar los datos sin que estos bailen a toda velocidad cambiaremos el valor del comando `delay()` a 500ms. De esta forma nuestro Sketch moverá el servo y nos mostrará la posición en la que se encuentra cada medio segundo, así podemos dejar el servo en la posición exacta que queramos viendo el número contenido dentro de la variable val.

Ejercicio 7-Relé-Puls

En esta ocasión vamos a utilizar un pulsador para encender y apagar una lámpara a través de un relé. Como veréis el código es el mismo que el Ejercicio 2. Comenzamos creando las variables que contendrán los números de los pines:

```
int pulsador = 2;  
int rele = 13;
```

En el setup() los configuraremos como entradas o como salidas, según corresponda:

```
void setup()  
{  
  pinMode(pulsador, INPUT);  
  pinMode(rele, OUTPUT);  
}
```

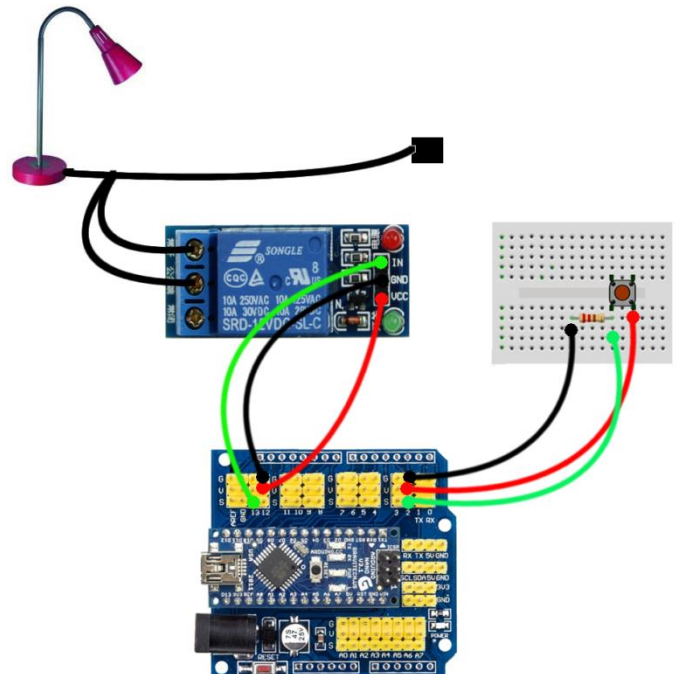
Ahora vamos al loop() en el que utilizaremos un comando condicional, el **if**, para que el relé amarillo se encienda si pulsamos el pulsador y se apague si no lo accionamos.

El comando if tiene una sintaxis así:

```
void loop()  
{  
  if (condicion)  
  {  
    // código que se ejecuta si se cumple la condición  
  }  
  else  
  {  
    // código que se ejecuta si no se cumple la  
    condición  
  }  
}
```

Por lo que para adaptarlo a nuestro ejemplo lo deberemos dejar de la siguiente manera. **ATENCIÓN** a la condición:

```
void loop()  
{  
  if (digitalRead(pulsador) == HIGH)  
  {  
    digitalWrite(rele, HIGH);  
  }  
  else  
  {  
    digitalWrite(rele, LOW);  
  }  
}
```



Hemos visto un comando más **digitalRead()**, que sirve para hacer una lectura de un pin digital, nos devolverá un HIGH o un LOW según el pin esté activado o no.

Código completo:

```
int pulsador = 2;
int rele= 13;

void setup()
{
  pinMode(pulsador, INPUT);
  pinMode(rele, OUTPUT);
}

void loop()
{
  if (digitalRead(pulsador) == HIGH)
  {
    digitalWrite(rele, HIGH);
  }
  else
  {
    digitalWrite(rele, LOW);
  }
}
```


Ejercicio 8-Relé-LDR

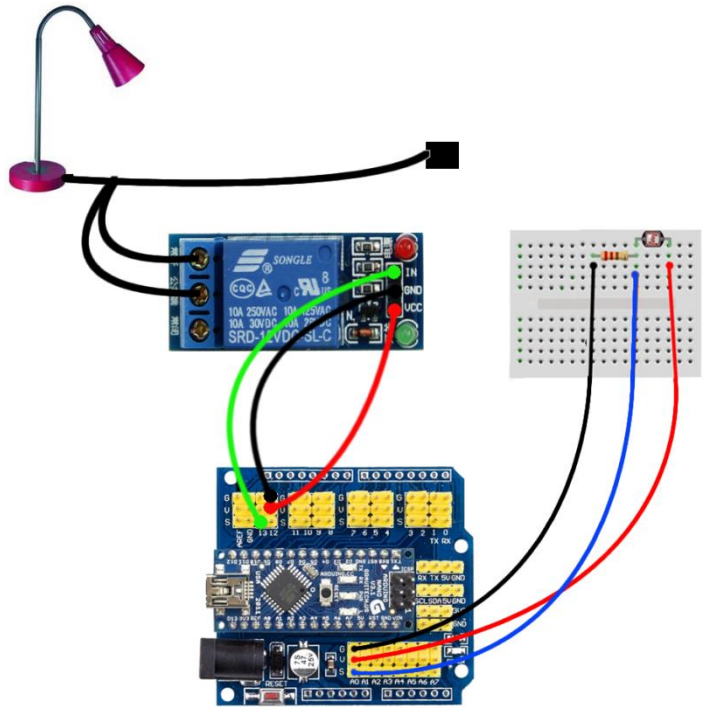
En esta ocasión vamos a utilizar una LDR para encender y apagar una lámpara a través de un relé. Como veréis el código es el mismo que el Ejercicio 4. Comenzamos creando las variables que contendrán los números de los pines:

Crearemos una variable **umbral**. Seguido utilizamos ese **umbral** para encender y apagar la lámpara en función de la cantidad de luz que recibe la LDR.

```
int RelePin = 13; // Asigna RelePin al pin 13
int LDRPin = A0; // Asigna LDRPin al pin A0
int umbral = 100; //este valor lo obtenemos
poniendo la mano sobre la LDR
```

```
void setup()
{
  pinMode(RelePin, OUTPUT); // Establece un
pin digital como salida.
  pinMode(LDRPin, INPUT); // Establece un
pin analógico como entrada.
}
```

```
void loop()
{
  if (analogRead(LDRPin) < umbral) { //
compara el valor de la LDR con en que hemos
asignado en la variable ñumbralö
    digitalWrite(RelePin, HIGH);
  }
  else {
    digitalWrite(RelePin, LOW);
  }
}
```



Ejercicio 9: Bluetooth

En esta experiencia vamos a transmitir información por el aire entre dos dispositivos bluetooth. Un dispositivo externo (ej: el móvil) actuará como emisor de datos mientras que nuestra arduino será el receptor.

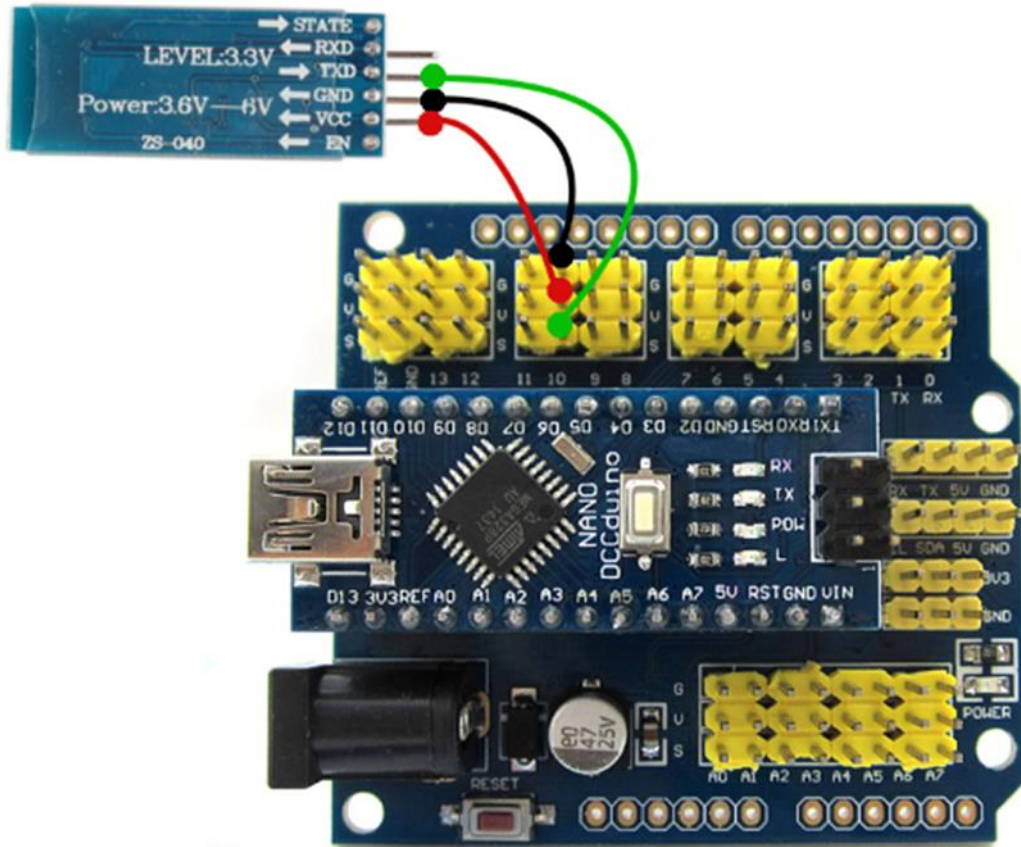
Bluetooth es un sistema de comunicaciones inalámbrico, para distancias cortas (ej: una sala), cuya conexión se suele realizar seleccionando el dispositivo e introduciendo un pin como contraseña. Los portátiles, móviles y otros dispositivos suelen incorporar esta tecnología.

La principal ventaja de bluetooth es que muchos de los dispositivos del hogar ya tienen disponible esta tecnología y por tanto crear la conexión es fácil, rápido y sin cables.

Pasos:

- Necesitamos un programa para enviar órdenes. Hay muchas aplicaciones gratis en los móviles para encender/apagar una bombilla, como joystick, etc. Seleccionad la que más os guste. Nosotros hemos probado con *Control Car* en android.
https://play.google.com/store/apps/details?id=appinventor.ai_el_profe_garcia.Arduino_Control_Car
- En arduino conectaremos el módulo bluetooth por el puerto serie. Para poder visualizar en nuestro ordenador lo que va pasando necesitamos otro serial así que para el módulo bluetooth utilizaremos la librería *SoftwareSerial* que permite utilizar el serial por los pines que queramos.
- El módulo bluetooth tiene 4 pines. Pin de energía (5V), pin de tierra o GND, pin de transmitir TX que se conecta al recibir de arduino (al revés) y pin de recibir RX que se conecta al transmitir de arduino (como no vamos a enviar datos al móvil, no es necesario conectarlo). Aunque pone voltaje = 3.3V, como solo vamos a enviar los datos, no hace falta un dispositivo de cambio de voltaje.
- El programa del móvil o del pc nos irá enviando los datos, que leeremos en la función *loop()* y actuaremos en consecuencia. Inicialmente sacaremos en la pantalla del ordenador lo que nos llega y posteriormente podemos utilizar esos datos para conectar un led, un servo, activar un relé, etc.

Esquema de conexiones:



El código relevante para la aplicación es:

```
// utilizar la librería para poder usar varios serials
#include <SoftwareSerial.h>

// leer los datos que llegan via bluetooth
if (softwareSerial.available()) {
    int comando = softwareSerial.read();
}
```

El código completo de la aplicación:

```
/**
    Recibir datos por BLUETOOTH
    Instalar programa en el móvil para enviar los datos

https://play.google.com/store/apps/details?id=appinventor.ai\_el\_profe\_garcia.Arduino\_Control\_Car

    el módulo BT04 es un buetooth de tipo esclavo -> sólo recibirá datos
    PIN (contraseña del bluetooth) = 1234

    Conexiones:
    - GND a tierra
    - VCC a 5 voltios (aunque ponga 3.3V no pasa nada porque la radio sólo la vamos a usar como receptor)
    - RXD al pin de transmision que configuremos en arduino (no es necesario conectarlo para éste ejercicio)
    - TXD al pin de recepción que configuremos en arduino

*/

#include <SoftwareSerial.h>

int PIN_RX = 10; // arduino RX = radio TX
int PIN_TX = 11; // arduino TX = radio RX
SoftwareSerial softwareSerial(PIN_RX, PIN_TX);

void setup() {
    Serial.begin(9600);
    softwareSerial.begin(9600); // Es un serial adicional para poder comunicarnos con la radio
}

void loop() {
    if (softwareSerial.available()) {
        int comando = softwareSerial.read();
        Serial.write(comando);
        ejecutar(comando);
    }
}
```

```
}
```

```
void ejecutar(int comando) {  
    switch (comando) {  
        case 'f' :  
            Serial.println( "botón ON");  
            break;  
        default:  
            ;  
    }  
}
```

Ejercicio 10: Infrarrojo (mando a distancia)

<http://www.dx.com/p/hengjiaan-ir-infrared-receiver-module-kit-w-remote-controller-461997>

La tecnología de emisión de datos por infrarrojos es característica de la mayoría de los mandos a distancia que tenemos en el hogar para controlar los dispositivos multimedia. Su funcionamiento consiste en emitir rayos de luz intermitentes que configuran una señal que es recibida por un aparato con forma de bombilla. Si el mando tiene una bombillita o cristal por delante, funciona por infrarrojos.

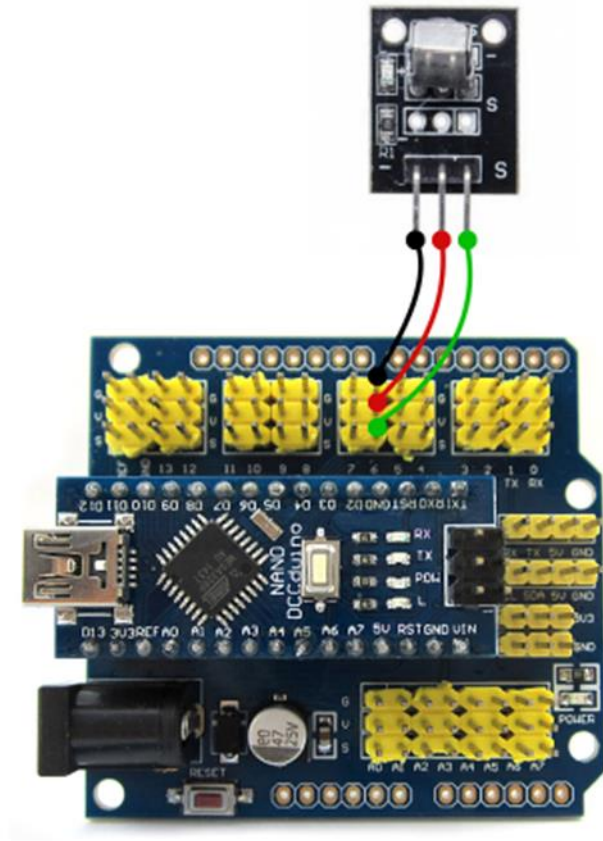
Las ventajas más destacables son la sencillez de utilización, el bajo coste, y la disponibilidad de esta tecnología en multitud de aparatos.

Las desventajas principales son que no puede haber obstáculos en la línea entre emisor y receptor, tampoco atraviesa paredes, y la principal, que cada aparato tiene su propio mando y acabamos llenando la mesita de la sala de mandos.

En este ejercicio vamos a recibir señales de un mando a distancia para poder ejecutar las acciones que queramos, por ejemplo encenderemos y apagaremos una luz, o aumentaremos y disminuirémos la intensidad de esa luz.

Al ser una tecnología muy madura, ya se conoce el formato de envío que utilizan los diferentes fabricantes, con lo cual la librería que vamos a usar contiene funciones sencillas para trabajar con cada tipo de mando. No obstante, a modo de ejemplo vamos a mostrar todos los datos que vienen desde un mando cualquiera ya que si no conocemos su modelo, también podemos copiar sus órdenes e integrar todas en un mando único, que controle los aparatos comerciales y los ñpropiosö.

El esquema de conexiones necesario:



El código relevante para la aplicación:

```
// instalar previamente la librería IRRemote
#include <Irremote.h>

// asignar el pin donde llegan los datos y configurar para infrarrojo
IRrecv irrecv(PIN_Lectura);
irrecv.enableIRIn();

// copiar los datos que llegan en la variable mensaje
irrecv.decode(&mensaje)
```

El código completo:

```
/**
 * Ejemplo de mando a distancia por infrarrojos
 *
 * Borrar la librería IRRemoteRobot si existe porque colisiona con la que vamos a instalar
 * (cerrar arduino, ir a la carpeta librerías y borrarla)
 *
 * Instalar la librería IRRemote
 *
 * Ayuda: http://www.pcbheaven.com/userpages/RC\_Protocol\_and\_Modulation/
 *
 * PINSET:
 * - GND a tierra
 * - VCC a 5 Voltios
 * - Signal a pin 6
 */
#include <IRremote.h>

int PIN_LECTURA = 6;
IRrecv irrecv(PIN_LECTURA);
decode_results mensaje;

void setup() {
  Serial.begin(9600);
  irrecv.enableIRIn();
}

void loop() {
  // copiar los datos que llegan en la variable mensaje
  if (irrecv.decode(&mensaje)) {
    // mostrar todos los datos posibles;
    dump(&mensaje);
    irrecv.resume();
    delay (100);
    // ejecutar en función del botón del mando
    if (mensaje.value == 0xFF02FD) {
```



```

        Serial.println("iiiiiiiiiii BOTON OK !!!!!!!!!!!");
    }
}
}

void dump(decode_results *results) {
    // Dumps out the decode_results structure.
    // Call this after IRrecv::decode()
    int count = results->rawlen;
    if (results->decode_type == UNKNOWN) {
        Serial.print("Unknown encoding: ");
    }
    else if (results->decode_type == NEC) {
        Serial.print("Decoded NEC: ");
    }
    else if (results->decode_type == SONY) {
        Serial.print("Decoded SONY: ");
    }
    else if (results->decode_type == RC5) {
        Serial.print("Decoded RC5: ");
    }
    else if (results->decode_type == RC6) {
        Serial.print("Decoded RC6: ");
    }
    else if (results->decode_type == PANASONIC) {
        Serial.print("Decoded PANASONIC - Address: ");
        Serial.print(results->address, HEX);
        Serial.print(" Value: ");
    }
    else if (results->decode_type == LG) {
        Serial.print("Decoded LG: ");
    }
    else if (results->decode_type == JVC) {
        Serial.print("Decoded JVC: ");
    }
    else if (results->decode_type == AIWA_RC_T501) {
        Serial.print("Decoded AIWA RC T501: ");
    }
}

```

```
else if (results->decode_type == WHYNTER) {
    Serial.print("Decoded Whynter: ");
}
Serial.print(results->value, HEX);
Serial.print(" (");
Serial.print(results->bits, DEC);
Serial.println(" bits)");
Serial.print("Raw (");
Serial.print(count, DEC);
Serial.print("): ");
for (int i = 1; i < count; i++) {
    if (i & 1) {
        Serial.print(results->rawbuf[i]*USECPERTICK, DEC);
    }
    else {
        Serial.write('-');
        Serial.print((unsigned long) results->rawbuf[i]*USECPERTICK, DEC);
    }
    Serial.print(" ");
}
Serial.println();
}
```

Ejercicio 11: RF_433Mhz

Los dispositivos que emiten/reciben a la frecuencia de 433Mhz se caracterizan por ser muy baratos frente a otras alternativas inalámbricas, consumen muy poco, debido a su baja frecuencia pueden atravesar paredes, tienen un rango de hasta 300 metros libre / 30 metros espacio cerrado (dependiendo de la antena y potencia), no se necesita una licencia para su uso, y son muy sencillos de usar (puerto serie).

Sus inconvenientes son: mensajes cortos (no se puede enviar vídeo por ejemplo), hay que probar /eliminar el ruido electromagnético, y no tienen ningún tipo de seguridad integrada. La longitud y calidad de la antena suele ser el factor determinante para conseguir la distancia máxima (y varía mucho!!!).

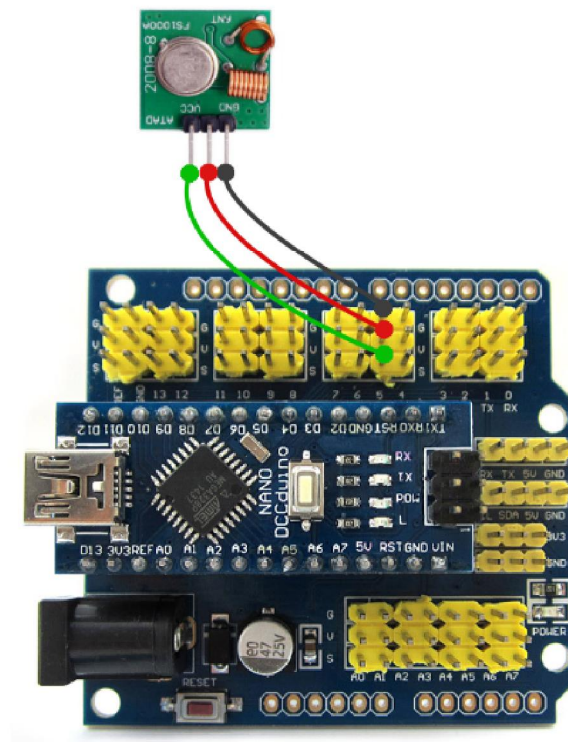
Generalmente vienen como dos dispositivos diferentes, uno es el emisor y el más grande suele ser el receptor. También hay dispositivos que combinan emisor y receptor.

Un caso de uso podría ser monitorizar las variables de un invernadero cercano a un caserío.

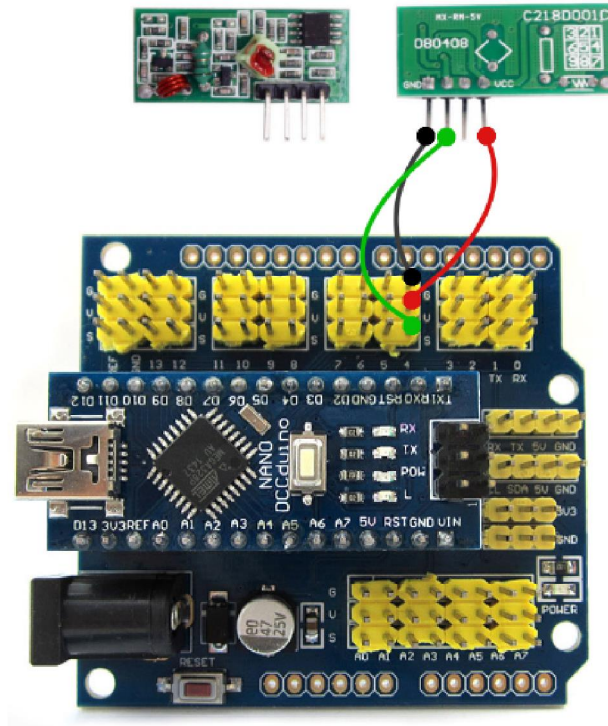
En esta experiencia vamos a enviar un texto desde un arduino, y otro arduino recibirá este texto. Una vez comprobado podríamos realizar una prueba completa instalando un sensor (un pulsador, una fotorresistencia,í) y ejecutando una acción en el otro arduino (encender una luz, mover un reléí).

El esquema para el

emisor:



El esquema para el receptor (cualquiera de los dos pines centrales es válido ya que son dos pines de datos ñgemelosö):



Tendremos que instalar la librería `RadioHead` (sketch/include library/manageí).

El código relevante para el funcionamiento de la aplicación:

```
#include <RH_ASK.h> // objetos para comunicarse por 433 Mhz
#include <SPI.h> // necesario para la librería rh_ask

RH_ASK radio(2000, 4, 5); // velocidad de transmisión, pin de lectura, pin de escritura
radio.send(mensaje, strlen(mensaje)); // enviar datos

bool hayMensaje = radio.recv(mensaje, longitud); // recibir mensaje
```

El código completo para el emisor:

```
/**
 * Radio de 433Mhz para TRANSMITIR datos inalámbricamente
 * Instalar librería "RadioHead"
 * 3 pines:
 *   - GND a tierra
 *   - VCC a 5 voltios
 *   - ATAD al pin de transmisión (en este caso PIN 5, ver debajo)
 */

#include <RH_ASK.h> // objetos para comunicarse por 433 Mhz
#include <SPI.h> // necesario para la librería rh_ask
.
// configurar la radio
RH_ASK radio(2000, 4, 5); // velocidad de transmisión, pin de lectura, pin de escritura

void setup() {
  Serial.begin(9600);
  if (!radio.init()) Serial.println("error en la radio");
}

void loop() {
  char mensaje[] = "Enviando 433"; //20 caracteres máximo, ver código del receptor
  radio.send(mensaje, strlen(mensaje));
  radio.waitPacketSent();
  Serial.println("paquete enviado");
  delay(1000);
}
```

El código completo para el receptor:

```
/**
    Radio de 433Mhz para RECIBIR datos inalámbricamente
    Instalar librería "RadioHead"
    3 pines:
    - GND a tierra
    - VCC a 5 voltios
    - cualquiera de los pines del medio para recibir (en este caso PIN 4, ver debajo)
*/

#include <RH_ASK.h>
#include <SPI.h> // necesario para la librería rh_ask

// configurar la radio
RH_ASK radio(2000, 4, 5); // velocidad de transmisión, pin de lectura, pin de escritura

void setup() {
    Serial.begin(9600);
    if (!radio.init()) Serial.println("error en la radio");
}

void loop() {
    char mensaje[20]; // configurar la longitud máxima del mensaje a recibir (20 caracteres)
    unsigned int longitud = sizeof(mensaje);
    bool hayMensaje = radio.recv(mensaje, longitud); // recibir mensaje
    if (hayMensaje) {
        Serial.println((char*)mensaje);
    }
}
```

ARDUINO: PRACTICAS WIFI

Instalación de la librería ESP-Wifi.

En la carpeta Wifi está incluida la librería que utilizaremos para hacer las prácticas.

Para incluir la librería abrir el IDE de arduino y seleccionar en el menú:

Programa → Incluir Librería.--> Añadir librería .ZIP

Buscar el zip de la librería donde hayas descargado las prácticas:

..\Wifi\Libreria WifiESP\ WiFiEsp-master.zip

La librería dispone de varios ejemplos de uso.

- [ConnectWPA](#) - Demonstrates how to connect to a network that is encrypted with WPA2 Personal
- [WebClient](#) - Connect to a remote webserver
- [WebClientRepeating](#) - Make repeated HTTP calls to a webserver
- [WebServer](#) - Serve a webpage from the WiFi shield
- [WebServerAP](#) - Serve a webpage from the WiFi shield starting a local Access Point
- [WebServerLed](#) - Turn on and off a led from a webpage
- [UdpNTPClient](#) - Query a Network Time Protocol (NTP) server using UDP

Mini introducción a los módulos Wifi basados en ESP8266.

Los módulos basados en ESP8266 se comunican con el arduino a través del puerto serie.

Las comunicaciones están basadas en comandos AT.

Para manejar el módulo wifi se crea una conexión por puerto serie y se envían los comandos AT, la respuesta a los comandos se recibe también por el puerto serie.

Al final se adjunta una lista de comandos AT soportados por los módulos wifi ESP8266 (en castellano) y en la carpeta Wifi hay un pdf con un manual del fabricante con información detallada (en Inglés)

Para facilitar el manejo de los módulos existen varias librerías que permiten abstraernos de los comandos AT y manejar el módulo wifi mediante las clases que la librería proporciona.

Los módulos wifi vienen configurados a una velocidad desconocida, normalmente 115200 baudios. En el arduino con un único puerto serie hardware, este puerto se utiliza para depuración y por tanto se tiene que emular otro puerto serie que será el que se comunicará con el módulo wifi. Los puertos serie emulados no funcionan bien a altas velocidades por lo que es necesario bajarla. La velocidad recomendada es 9600.

Clases de la librería WifiESP

La mayor parte de los métodos de la librería standard de Arduino Wifi están disponibles.

Para más detalles ir a [WiFi library page](#).

WiFiEsp class

- begin() - Not all authentication types
- disconnect() - YES
- config()
- setDNS() - NO (no AT command available)
- SSID() - YES
- BSSID() - YES
- RSSI() - YES
- encryptionType() - NO (no AT command available)
- scanNetworks() - YES
- getSocket()
- macAddress() - YES

WiFiEspServer class

The WiFiEspServer class creates servers which can send data to and receive data from connected clients (programs running on other computers or devices).

- WiFiEspServer() - YES
- begin() - YES
- available() - YES
- write() - YES
- print() - YES
- println() - YES

Client class

The WiFiEspClient class creates clients that can connect to servers and send and receive data.

- WiFiEspClient() - YES
- connected() - YES
- connect() - YES
- write() - YES

- `print()` - YES
- `println()` - YES
- `available()` - YES
- `read()` - YES
- `flush()` - YES
- `stop()` - YES

WiFiEspUDP class

The UDP class enables UDP message to be sent and received.

- `WiFiUDP` - YES
- `begin()` - YES
- `available()` - YES
- `beginPacket()` - YES
- `endPacket()` - YES
- `write()` - YES
- `parsePacket()` - YES
- `peek()`
- `read()` - YES
- `flush()`
- `stop()`
- `remoteIP()` - YES
- `remotePort()` - YES

Comandos AT soportados por los módulos ESP8266

La siguiente tabla está actualizada hasta la versión 0.20, que corresponde con el firmware 0.9.4 de diciembre de 2014, la versión (oficial) más avanzada que soportan los módulos ESP8266 con 4 Mbit, que son la mayoría de los disponibles actualmente.

Orden AT	Descripción
AT	Sirve para verificar si el sistema de órdenes AT está operativo
AT+GMR	Muestra información sobre el firmware del módulo
AT+RST	Reinicia el módulo
ATE1 ATE0	Activa o desactiva, respectivamente, el eco de los comandos AT. Normalmente el eco está activo por lo que se muestra tanto el resultado de la operación como la propia operación. Con esta orden se puede reservar alguna información, la clave del punto de acceso, por ejemplo, mientras se muestra por una consola el resto de la información.
AT+GSLP	Entra en modo de reposo (si el hardware lo permite) durante el tiempo indicado en milisegundos con el formato AT+GSLP={tiempo} siendo {tiempo} el tiempo de reposo expresado en milisegundos.
AT+CWMODE	Establece el modo de funcionamiento WiFi del módulo. Hay tres disponibles (1) estación, (2) punto de acceso y (3) mixto: punto de acceso y estación. Por ejemplo, para establecer el modo de funcionamiento WiFi del módulo como estación se usaría AT+CWMODE=1 y para consultar el modo actual de funcionamiento se usaría AT+CWMODE?
AT+CIOBAUD AT+IPR	Permite establecer la velocidad de comunicación del módulo (expresada en baudios) según el formato AT+CIOBAUD={velocidad} Utilizada con el formato AT+CIOBAUD? informa de la velocidad actualmente establecida. Los valores permitidos son 9600, 19200, 38400, 57600, 74880, 115200, 230400, 460800, 921600 y la configuración por defecto es 115200 En realidad, por algún tipo de error, CIOBAUD no funciona y sí IPR (que no está en la documentación) El inconveniente es que AT+IPR? no está implementado y hay que cambiar la velocidad a ciegas.
AT+CSYSWDTENABLE AT+CSYSWDTDISABLE	Activa o desactiva el watchdog (perro guardián) que permite reiniciar el módulo automáticamente si se produce un error de funcionamiento.
AT+CIPSTAMAC AT+CIPAPAMAC	Permite establecer una dirección MAC o consultar la actual tanto en el punto de acceso (CIPAPAMAC) como en la estación (CIPSTAMAC). Para establecer una nueva dirección MAC se utiliza la orden con el formato AT+CIPSTAMAC={MAC} siendo {MAC} la dirección MAC en el formato 01:34:67:89:AB:CD Para consultar la dirección MAC actual se usa la orden con el formato AT+CIPSTAMAC?

AT+CWLAP	<p>Muestra una lista con los puntos de acceso disponibles en el alcance del módulo, indicando para cada uno de ellos la codificación, el SSID, el nivel de señal y la dirección MAC.</p> <p>Las diferentes codificaciones se representan por un número: abierta=0, WEP=1, WPA/PSK=2, WPA2/PSK=3 y WPA/WPA2/PSK=3.</p> <p>Se puede limitar la búsqueda a los puntos de acceso que cumplan ciertas condiciones como el SSID, la MAC o el número de canal indicándolos en la orden.</p>
AT+CWJAP	<p>Usando la orden con el formato AT+CWJAP={SSID},{clave} el módulo se conecta al punto de acceso indicado por el SSID usando la clave especificada.</p> <p>Para saber a qué punto de acceso se encuentra actualmente conectado el módulo se puede usar la orden en el formato AT+CWJAP?</p>
AT+CWQAP	<p>Desconecta el punto de acceso.</p> <p>Para comprobar si se ha desconectado se puede usar la orden con el formato AT+CWQAP?</p>
AT+CWSAP	<p>Configura el modo de funcionamiento del punto de acceso según el formato AT+CWSAP={SSID},{clave},{canal},{cifrado}</p> <p>Los modos de cifrado corresponden con los códigos abierta=0, WEP=1, WPA/PSK=2, WPA2/PSK=3 y WPA/WPA2/PSK=4</p>
AT+CWDHCP	<p>Activa o desactiva el DHCP (asignación dinámica de dirección IP) en el punto de acceso y/o en la estación.</p> <p>El formato de la orden es AT+CWDHCP={modo},{estado} siendo {modo} el código del que se activa (0 para el punto de acceso, 1 para la estación o 2 para ambos) y {estado} activar/desactivar DHCP según se use 1 ó 0 como valor.</p>
AT+CIPSTA AT+CIPAP	<p>Establece la dirección IP de la estación (AT+CIPSTA) o del punto de acceso (AT+CIPAP) usada en el formato AT+CIPSTA={IP} o muestra la actual usada en el formato AT+CIPSTA?</p>
AT+CWLIF	<p>Muestra una lista de las direcciones IP de los dispositivos conectados al módulo en modo punto de acceso.</p>
AT+CIFSR	<p>Muestra la dirección IP local actual del módulo</p>
AT+CIPMUX	<p>Establece el modo de conexión simple o múltiple según se use con el formato AT+CIPMUX=0 ó AT+CIPMUX=1 respectivamente.</p>
AT+CIPSERVER	<p>Activa o desactiva el modo de servidor según se use en el formato AT+CIPSERVER=0 (que requerirá un reinicio para hacerse efectivo) o AT+CIPSERVER=1,{puerto} que comenzará a aceptar peticiones en el puerto correspondiente.</p> <p>Para poder establecer el modo servidor debe activarse el modo multi-conexión con AT+CIPMUX=1</p>
AT+CIPSTO	<p>Configura el tiempo de espera (timeout) cuando el módulo funciona en modo servidor con el formato AT+CIPSTO={segundos}</p> <p>Si se usa con el formato AT+CIPSTO? informa del timeout actual.</p>

AT+CIPSTART	<p>Inicia una conexión con un servicio. Es necesario indicar el tipo de conexión (TCP/UDP) la dirección IP (o el nombre del servidor, si se tiene acceso a un DNS) y el puerto al que se realiza la conexión con el formato AT+CIPSTART={id},{TCP UDP},{IP},{puerto} Siendo {id} el número de identificador de la conexión (si se aceptan varias, es decir si se utilizó AT+CIPMUX=1, si AT+CIPMUX=0 no hay que indicarlo)</p> <p>También es posible añadir al final de la orden (separado por comas como el resto de parámetros) el puerto local y el modo si se trata de una conexión UDP.</p> <p>Usada con el formato AT+CIPSTART=? muestra la información anterior relativa a la conexión actualmente en curso.</p>
AT+CIPSTATUS	<p>Muestra el estado de la conexión actual en el formato STATUS:{estado} siendo {estado} un valor que representa el estado de la conexión como desconectado por un 4, con una IP asignada por un 2, conectado por un 3 y conectado con IP asignada por un 5.</p> <p>Después del estado genérico detalla la conexión en el formato +CIPSTATUS:{id},{tipo},{dirección},{puerto},{modo} siendo {modo} un valor que representa si el módulo está funcionando como cliente (0) ó como servidor (1)</p>

Ejercicio 12: Ejemplo de la librería WebserverLed

Descripción: En este ejemplo el arduino se conecta a un red wifi y crea un servidor web con una página html simple que permite encender y apagar el led que está incluido en la placa del arduino (pin 13).

1. Para realizar esta práctica necesitaremos un punto de acceso al que conectará el arduino, este punto lo crearemos con un móvil de Tecnalía. Para ello, hay que ir a:

Ajustes → Conexión Compartida → Activar Conexión compartida.

Apuntar el SSID (nombre del punto de acceso y la clave) las necesitaremos más adelante.

2. Seleccionar el tipo de arduino que vamos a utilizar (nano)
3. Cargar el ejemplo. Para ello, abrir el IDE de arduino y en el menú ir a:

Archivo → Ejemplos → WifiESP → WebServerLed

4. Para que el ejemplo funcione en nuestra plataforma hay que hacer algunos cambios en el código.

- a) Comentar las dos líneas siguientes para asegurar que se crea el puerto emulado.

Para comentar se utiliza la doble barra inclinada a la derecha (/).

```
//#ifndef HAVE_HWSERIAL1
```

```
#include "SoftwareSerial.h"
```

```
SoftwareSerial Serial1(6, 7); // RX, TX
```

```
//#endif
```

- b) Cambiar el SSID y el Password que tiene el ejemplo por los del punto de acceso al que nos conectaremos, el del móvil.

```
char ssid[] = "Twim"; // your network SSID (name)
```

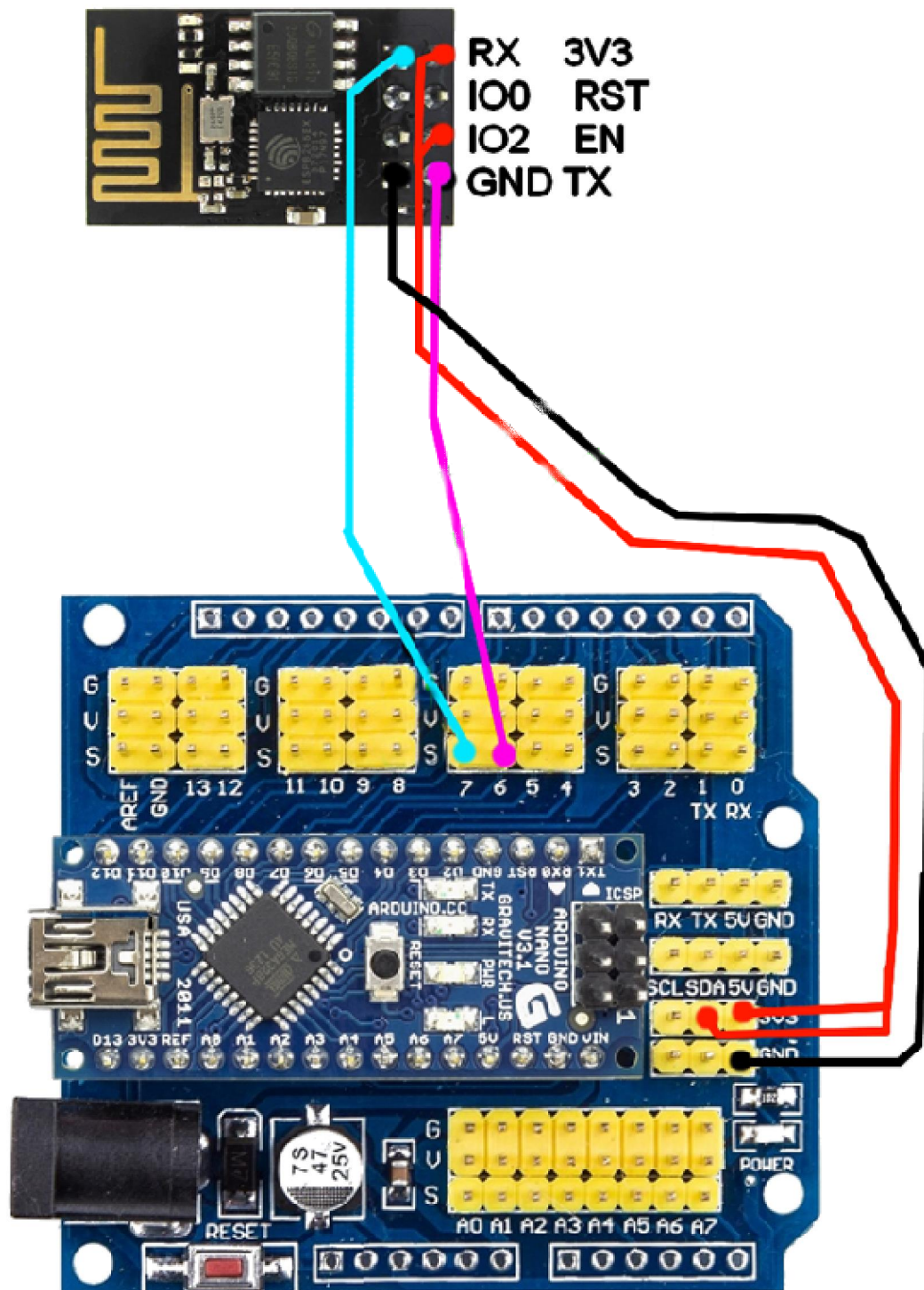
```
char pass[] = "12345678"; // your network password
```

- c) Cambiar la velocidad del serial de 115200 a 9600.

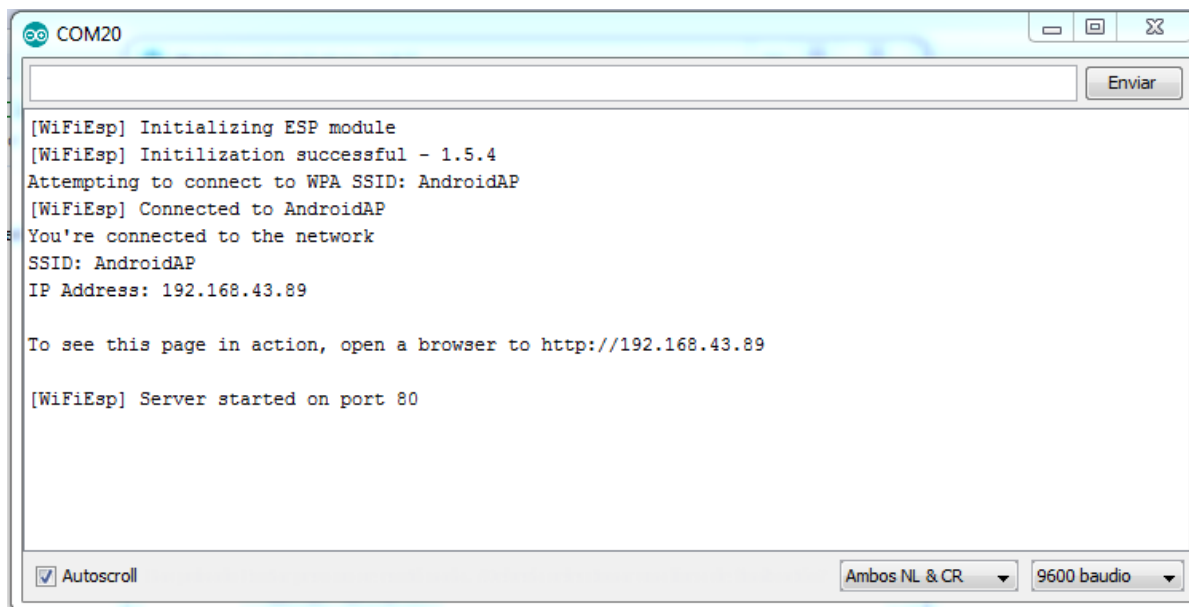
```
Serial.begin(9600); // initialize serial for debugging
```

```
Serial1.begin(9600); // initialize serial for ESP module
```

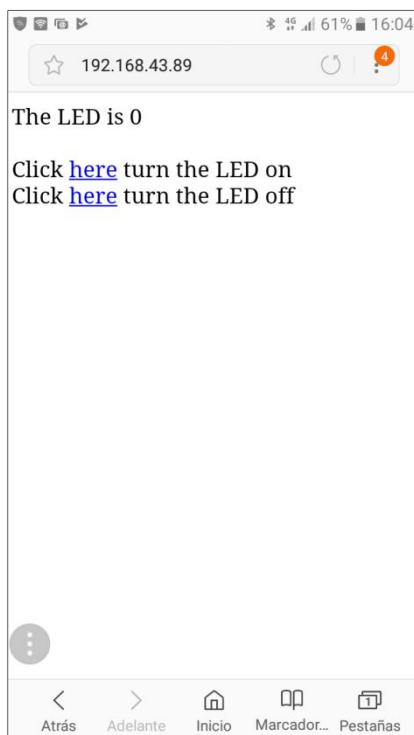
5. Conectar el módulo ESP8266 al arduino como se muestra en la figura siguiente:



6. Conectar el arduino con el PC mediante el cable USB.
7. Compilar y subir el programa.
8. Abrir el monitor serie en el menú: *Herramientas* → *Monitor Serie* y configurarlo a 9600bps.
9. En el monitor serie se mostrará la inicialización del módulo.



10. Para ver la página web hay que abrir el explorador de internet en el móvil e introducir la URL que se indica en el monitor.
11. En la página web aparecerán dos enlaces, uno para encender el LED y otro para apagar el LED



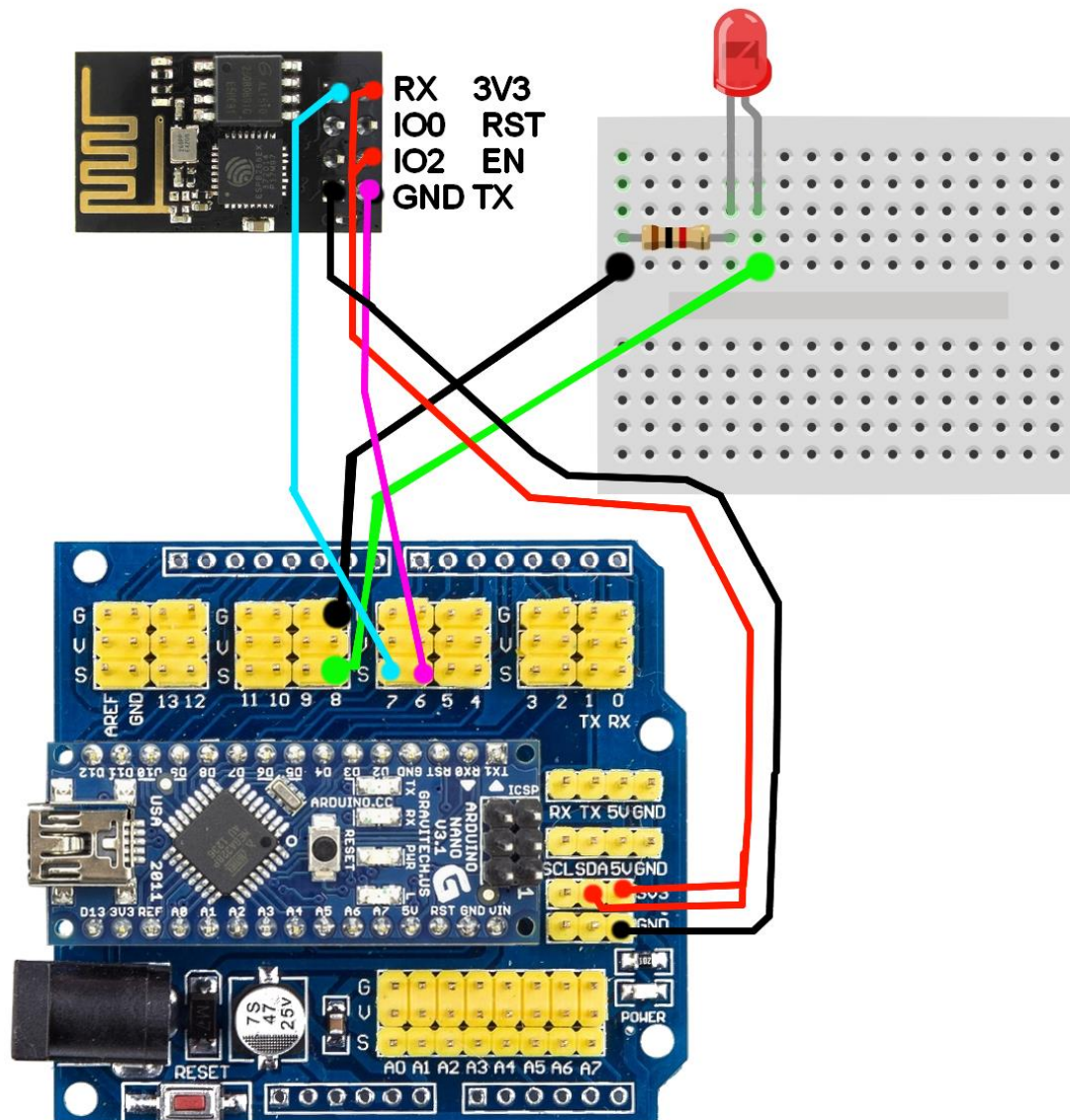
Ejercicio 13: Wifi-Led-WebserverAP con Arduino como punto de acceso.

Descripción: Esta práctica es parecida a la anterior con dos diferencias principales:

- Ahora el arduino será el punto de Acceso y por tanto desde el móvil nos conectaremos al punto de acceso del arduino para visualizar la página web.
- Utilizaremos otro led diferente del que incluye el nano en su placa.

12. Realizar el montaje de la siguiente Figura.

(Ojo con la polaridad del LED)



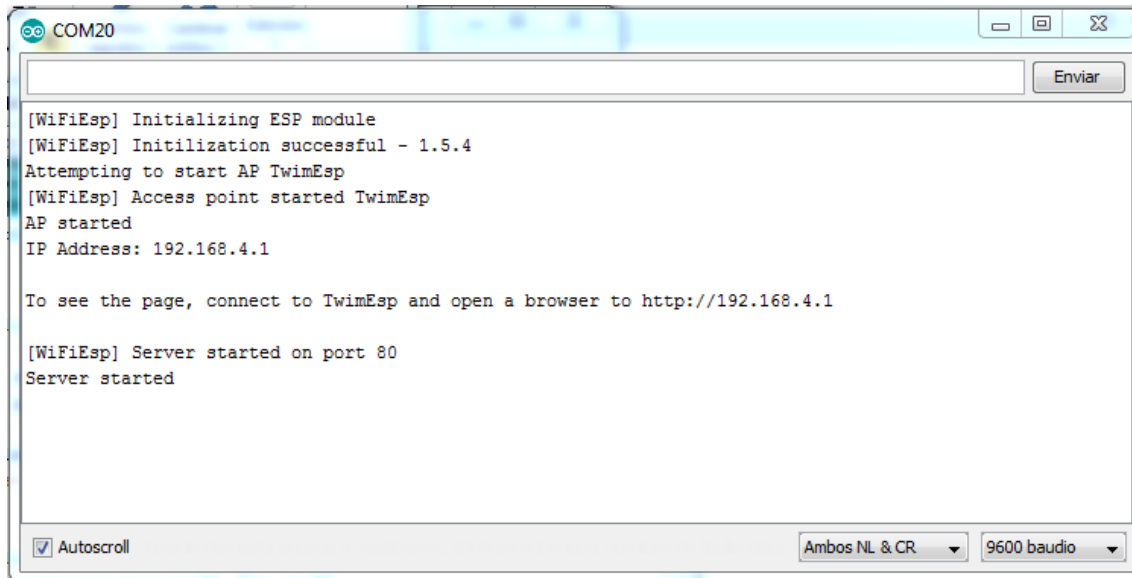
13. Abrir el IDE de arduino, comprobar que esta seleccionado el arduino nano. Y si no lo está, seleccionarlo.

14. Cargar el ejercicio 13. Para ello, abrir el IDE de arduino y en el menú ir a:

Archivo → Abrir y buscar en la carpeta Wifi el Ejercicio WebServer_ControlLed.ino

15. Compilar y subir el programa

16. Abrir el monitor serie y comprobar que este configurado a 9600bps. En el monitor veremos la inicialización del módulo.



The screenshot shows the Arduino IDE Serial Monitor window for COM20. The window has a title bar with standard Windows icons and a 'Enviar' button. The main text area displays the following output:

```
[WiFiEsp] Initializing ESP module
[WiFiEsp] Initilization successful - 1.5.4
Attempting to start AP TwimEsp
[WiFiEsp] Access point started TwimEsp
AP started
IP Address: 192.168.4.1

To see the page, connect to TwimEsp and open a browser to http://192.168.4.1

[WiFiEsp] Server started on port 80
Server started
```

At the bottom of the window, there is a status bar with an 'Autoscroll' checkbox (checked), a dropdown menu set to 'Ambos NL & CR', and another dropdown menu set to '9600 baudio'.

17. Conectarse al punto de acceso creado por el arduino.

18. Abrir un explorador de internet y poner la URL que se indica en el monitor:

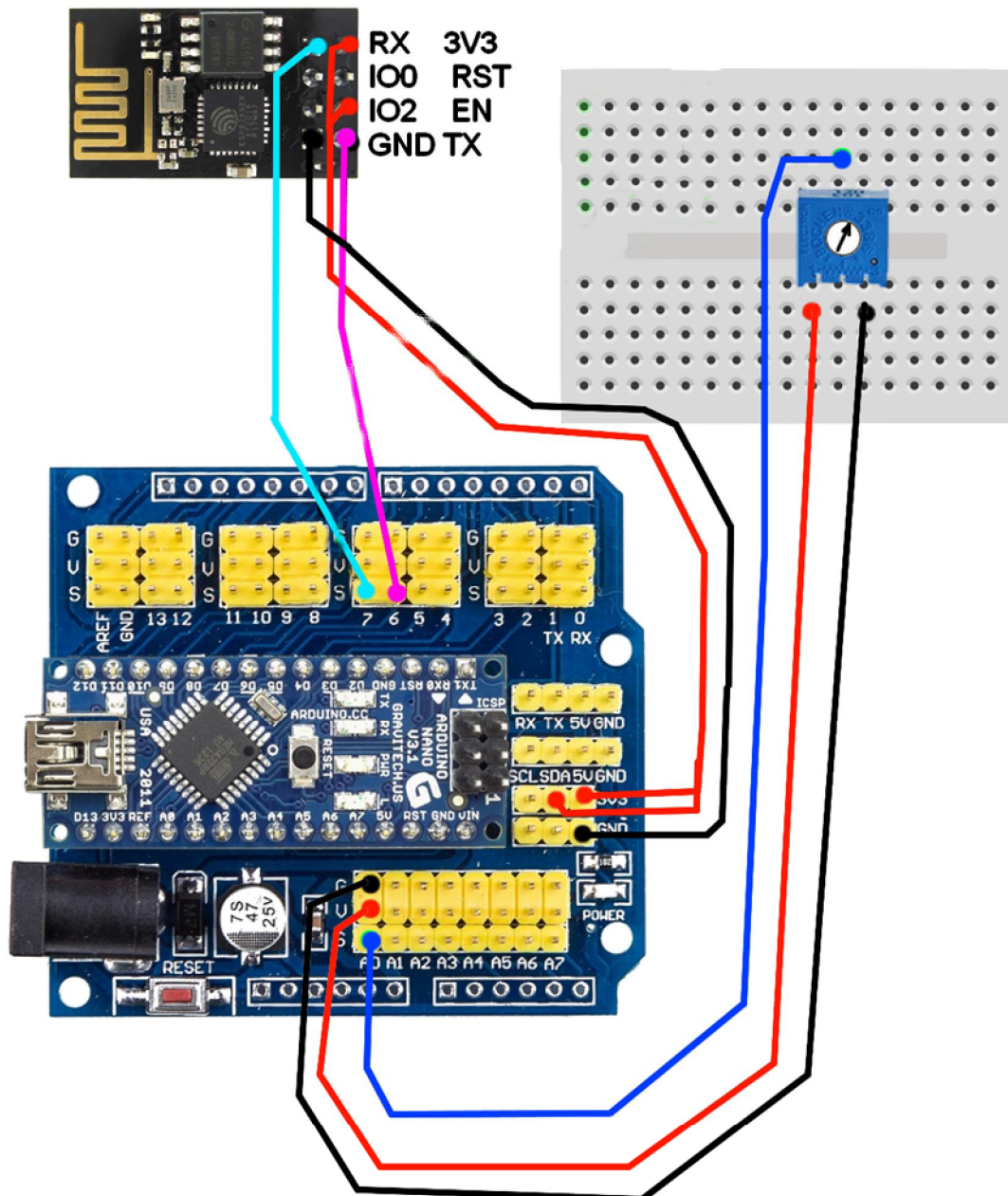
<http://192.168.4.1>

Con el botón que aparecerá en la página web se puede encender y apagar el LED

Ejercicio 14: Wifi-ADC-WebserverAP con Arduino como punto de acceso.

Descripción: Esta práctica es parecida a la anterior con una diferencia. Ahora leeremos una magnitud analógica y la presentaremos en la página web refrescando la medida cada 10 segundos. La magnitud analógica la controlaremos con un potenciómetro

19. Realizar el montaje de la siguiente Figura.



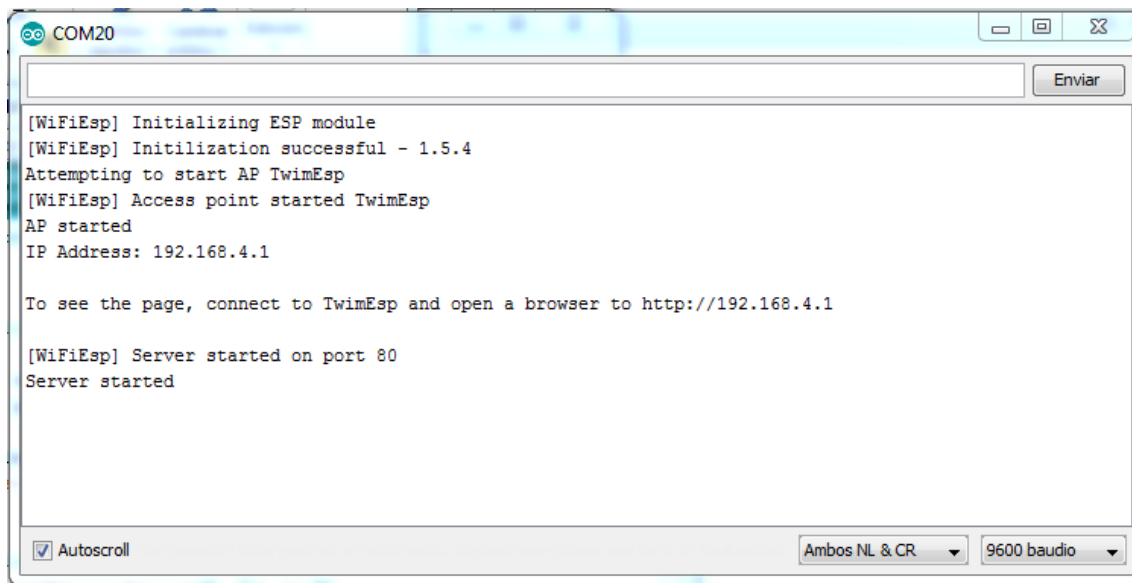
20. Abrir el IDE de arduino, comprobar que esta seleccionado el arduino nano. Y si no lo está, seleccionarlo.

21. Cargar el ejercicio 13. Para ello, abrir el IDE de arduino y en el menú ir a:

Archivo → Abrir y buscar en la carpeta Wifi el Ejercicio WebServer_Controlled.ino

22. Compilar y subir el programa.

23. Abrir el monitor serie y comprobar que este configurado a 9600bps. En el monitor veremos la inicialización del módulo.



The screenshot shows the Arduino IDE Serial Monitor window for COM20. The window has a title bar with standard OS controls and a close button. Below the title bar is a text input field and an 'Enviar' button. The main area displays the following text:

```
[WiFiEsp] Initializing ESP module
[WiFiEsp] Initilization successful - 1.5.4
Attempting to start AP TwimEsp
[WiFiEsp] Access point started TwimEsp
AP started
IP Address: 192.168.4.1

To see the page, connect to TwimEsp and open a browser to http://192.168.4.1

[WiFiEsp] Server started on port 80
Server started
```

At the bottom of the window, there is a checkbox for 'Autoscroll' which is checked, and two dropdown menus: 'Ambos NL & CR' and '9600 baudio'.

24. Conectarse al punto de acceso creado por el arduino.

25. Abrir un explorador de internet y poner la URL que se indica en el monitor:

<http://192.168.4.1>

Veremos la página web refrescarse cada 10 segundos con el valor analógico seleccionado con el potenciómetro.

También aparece un contador con el número de veces que se ha refrescado la medida.