INTRODUCCIÓN AL ESP32

ARDUINO IDE

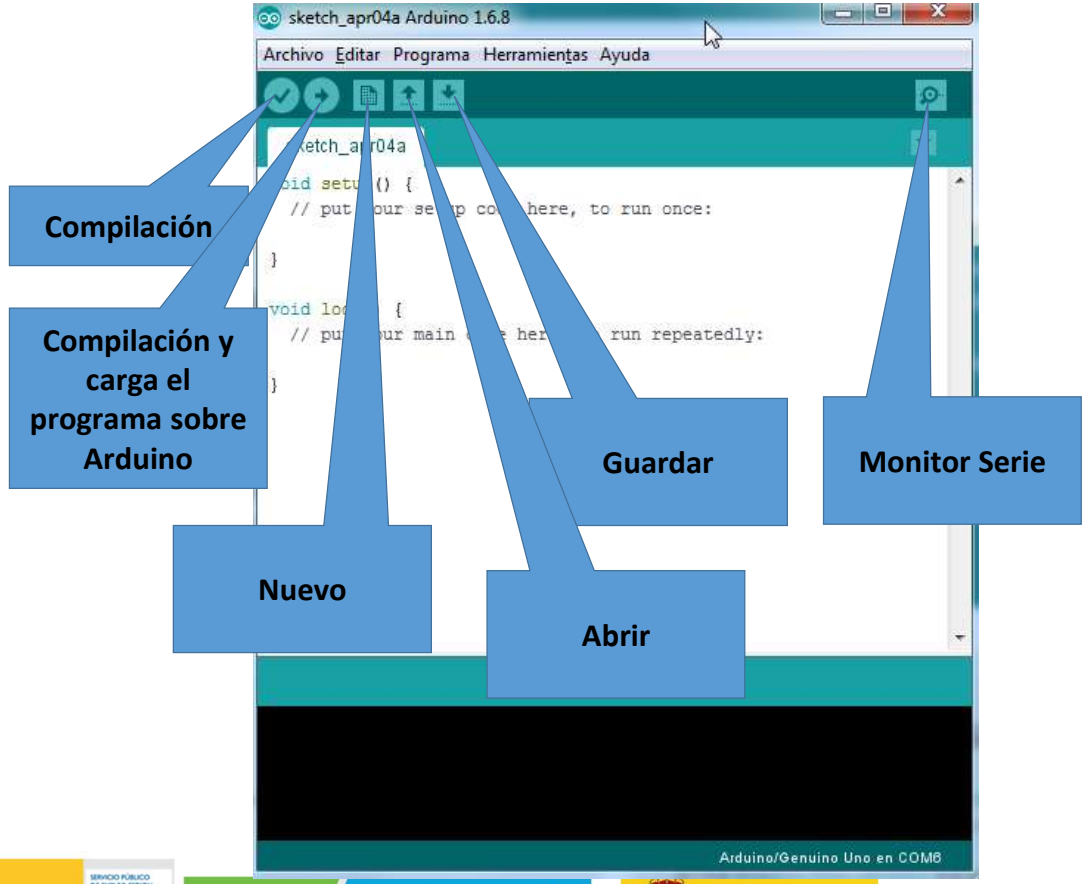
El entorno de programación de Arduino permite editar, compilar y ejecutar programas sobre cualquier Arduino.

Tiene una consola de comunicación serie con el Arduino que permite recibir o enviar cadenas de caracteres.

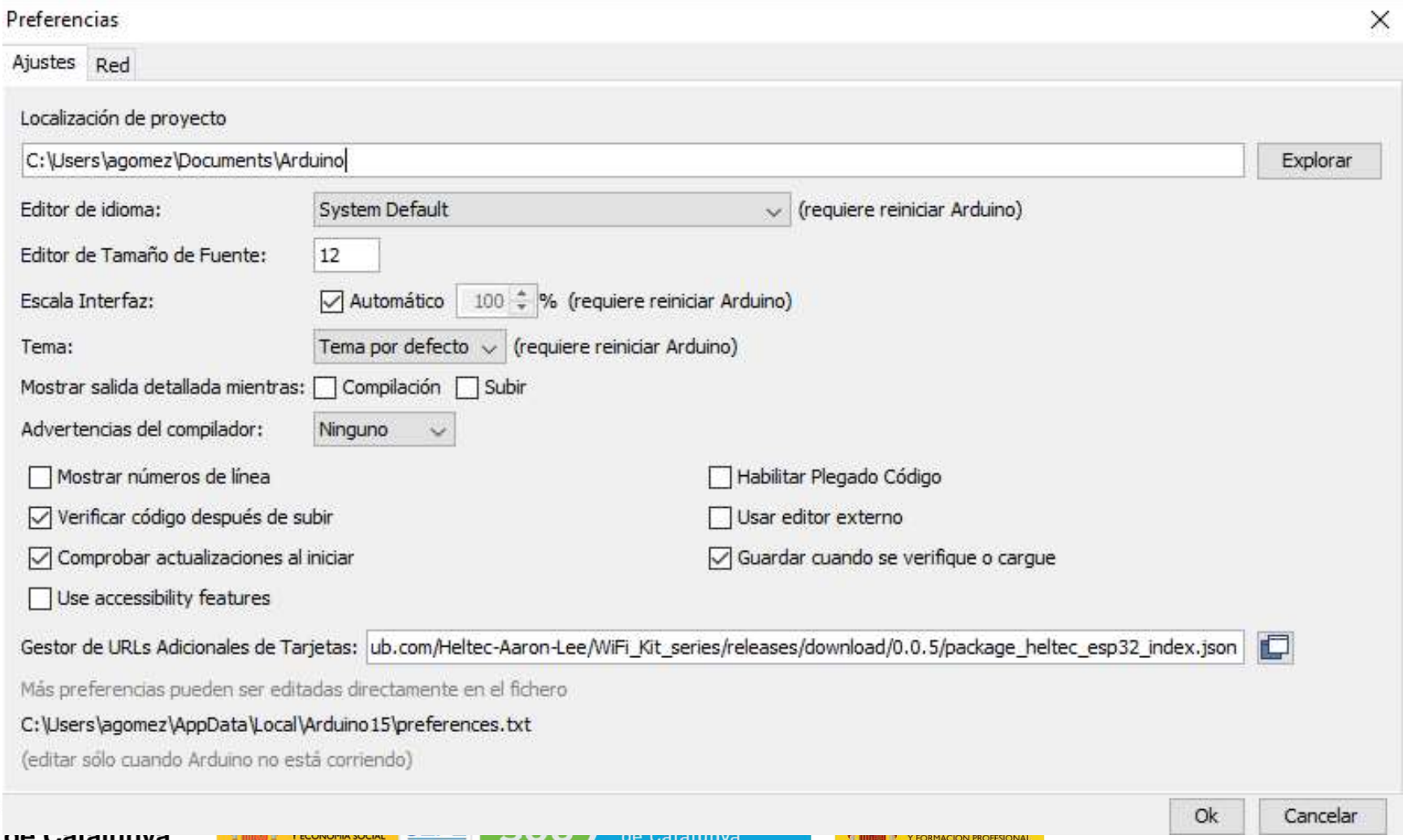
El IDE (actualmente versión 1.8) se puede descargar de la página oficial:

https://www.arduino.cc/en/Main/Software

ARDUINO IDE


sketch_apr04a Arduino 1.6.8
Archivo Editar Programa Herramientas Ayuda
Arduino/Genuino Uno en COM8
Compilación
Compilación y carga el programa sobre Arduino
Nuevo
Abrir
Guardar
Monitor Serie

ARDUINO IDE


Preferencias
Ajustes
Red
Localización de proyecto
C:\Users\agomez\Documents\Arduino
Explorar
Editor de idioma:
System Default
(requiere reiniciar Arduino)
Editor de Tamaño de Fuente:
12
Escala Interfaz:
Automático
100
%
(requiere reiniciar Arduino)
Tema:
Tema por defecto
(requiere reiniciar Arduino)
Mostrar salida detallada mientras:
Compilación
Subir
Advertencias del compilador:
Ninguno
Mostrar números de línea
Habilitar Plegado Código
Verificar código después de subir
Usar editor externo
Comprobar actualizaciones al iniciar
Guardar cuando se verifique o cargue
Use accessibility features
Gestor de URLs Adicionales de Tarjetas:
ub.com/Heltec-Aaron-Lee/WiFi_Kit_series/releases/download/0.0.5/package_heltec_esp32_index.json
Más preferencias pueden ser editadas directamente en el fichero
C:\Users\agomez\AppData\Local\Arduino15\preferences.txt
(editar sólo cuando Arduino no está corriendo)
Ok
Cancelar

ARDUINO IDE

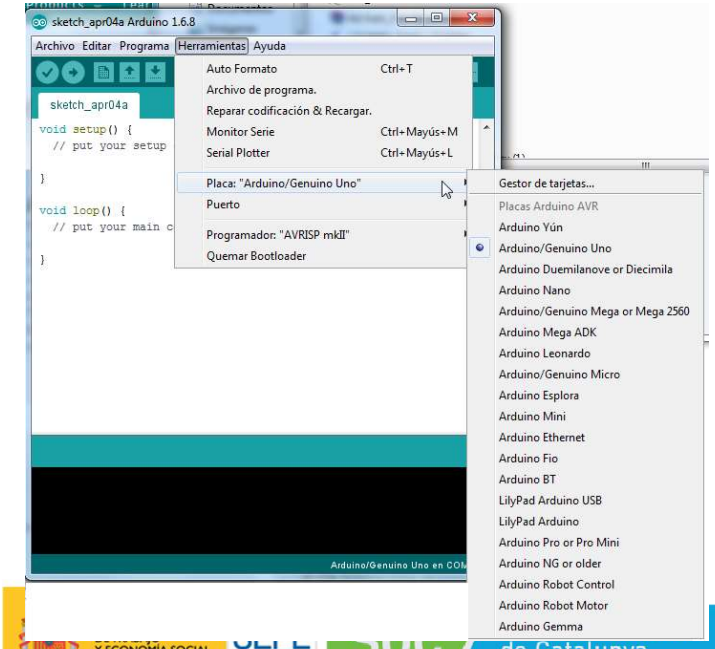
Herramientas-Tarjeta: Selección de tarjeta según el dispositivo que vayamos a programar:

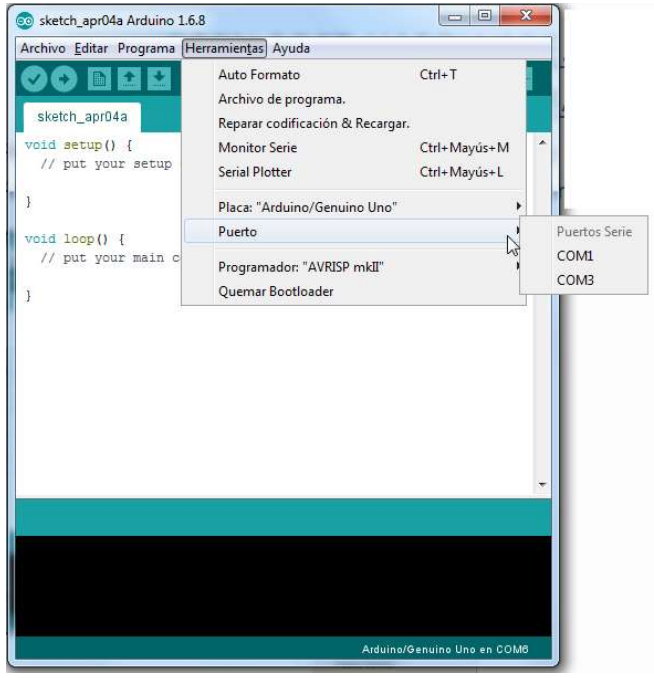
"Heltec ESP32 Arduino"

"WIFI LoRa 32(V2)"

"ESP32 Dev Module"

Herramientas-Puerto Serial: Selección de puerto serie asignado.





ARDUINO IDE

Estructura de un programa Arduino

```
#include <math.h>
#include <Wire.h>
```

Librerías: Código externo que adjuntamos en nuestro programa

```
int iVar=0; //variable global entero
char cVar=0; //variable global carácter
float fVar=0; //variable global decimal
boolean bVar=false; //variable global lógica
char sVar[10]; //variable global cadena de 10 caracteres
```

Variables globales: variables accesibles desde cualquier lugar del programa.

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

setup: Función de inicialización. Aquí se inician procesos y variables. Se ejecuta solo una vez, al reiniciar el dispositivo.

```
void loop(){
  iVar=iVar+1;
  EscribeConsola(iVar);
  delay(1000);

}
```

loop: Función principal. Se ejecuta continuamente.

```
void EscribeConsola(int iPar){
  char cadena[20]="Valor iVar:\0";
  Serial.print(cadena);
  Serial.println(iPar);
}
```

Funciones: Se ejecuta cada vez que son invocadas en otra función o en “loop”. Pueden definirse variables, pero solo serán válidas en el ámbito de la función.

INTRODUCCIÓN A LA PROGRAMACIÓN

Conceptos importantes:

Variable:
Zona de memoria que almacena un determinado valor que puede cambiar a lo largo de la ejecución del programa. La posición tiene un nombre para poder acceder a su valor. También podemos acceder al valor mediante la dirección de memoria.

Constante:
Zona de memoria que almacena un determinado valor que **no** puede variar. Tiene un nombre para acceder a este valor. También podemos acceder a la dirección. La diferencia con las variables es que su valor no cambia y que sus accesos son más rápidos.

Sentencia:
Instrucción que se ejecuta en una sola vez. Finaliza siempre con un punto y coma (;)

Bloque:
Conjunto de sentencias que forman una unidad de código. Se acotan mediante llaves ({}). Un bloque puede contener otro bloque. En este sentido, **es muy importante la anidación**.

Función:
Bloque al que se le asigna un nombre para poder ser invocado desde cualquier parte del código. Se programa fuera de Main. Se puede intercambiar información a través de los parámetros de entrada y salida. **Siempre retorna un valor(es)**.

Procedimiento:
Bloque al que se le asigna un nombre para poder ser invocado desde cualquier parte del código. Se programa fuera de Main. Se puede intercambiar información a través mediante de los parámetros de entrada y salida. **No retorna nada**.

INTRODUCCIÓN A LA PROGRAMACIÓN. VARIABLES

Variables y constantes

- **Variables**: Espacio reservado en memoria para almacenar un dato, cuyo valor puede variar a lo largo de la ejecución del programa.
- **Constantes**: Espacio reservado en memoria para almacenar un dato, cuyo contenido **no** varia a lo largo de la ejecución del programa. El acceso a este dato es más rápido que el de las variables.

Una variable/constante se compone de:

- **Un nombre:** es la manera que tenemos para hacer referencia a la variable. Normalmente se utiliza un nombre que nos indique lo que estamos guardando. Debe de ser alfanumérico: siempre ha de tener letras, pueden incluirse números y determinados signos. No pueden llevar caracteres reservados como: espacios, puntos, ampersand (&), admiraciones, comas, etc.
  Por ejemplo: temperatura_sensor, humedad1, temperatura_media…..

- **Un tipo:** indica la naturaleza del valor que se va a guardar en memoria: si la variable es un número (y que tipo de número), un carácter, un booleano, una cadena, etc,
- **Un valor:** en el caso de variables es posible asignar el valor en cualquier parte del programa, respetando su ámbito de actuación. Si la declaramos global el acceso al valor es posible en cualquier parte del programa.
- **Una dirección en memoria:** en que dirección de memoria se va a guardar el valor. Esta dirección se asigna automáticamente.

Para declarar o crear una variable es **obligatorio asignar un nombre y un tipo**. Es aconsejable asignar un valor.

INTRODUCCIÓN A LA PROGRAMACIÓN. VARIABLES

Tipos de variables

int	Enteros de 4 bytes • signed -2^{31} *a* $2^{31}-1$ • unsigned 0 *a* $2^{32}-1$
char	Caracteres 1 byte
float	Números en coma flotante (7 dígitos significativos) 4 bytes •signed -3.402823E+38 a -1.175494E-38 •unsigned 1.175494E-38 a 3.402823E+38
double	Números en coma flotante de doble precisión (7 dígitos significativos) 8 bytes •signed -1.79169E+308 a -2.22507E-308 •unsigned 2.22507E-308 a 1.79169E+308
void	no-tipo (se emplea con punteros)

short	Entero corto de 2 bytes • signed -2^{15} *a* $2^{15}-1$ • unsigned 0 *a* $2^{16}-1$
long	Entero largo de 8 bytes • signed -2^{31} *a* $2^{31}-1$ • unsigned 0 *a* $2^{32}-1$

Ejemplos

```
char letra ='2';
char letraAscii =50;
char letraHex =0x32;
char letraBin = 110010b;

int entero = 15;
int enteroHexa = 0xF;
int enteroBin= 1111b;

float decimal = 0.00004;
double decimalDb = 54.3e200;
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Sentencia bucle for

Las sentencias bucle son muy importantes porque permiten ejecutar una sentencia un número de veces. Se utiliza para acceder a los valores de un array porque permite recorrerlo elemento a elemento.

La sintaxis es:

for (expresión_inicial; condición; expresión_de_paso)
sentencia

- ”expresión_inicial”: inicialización de la variables que se va a interar.

- ”condición”: si es verdadera ejecuta ”sentencia” cíclicamente hasta que la “condición” sea falsa.
- ”expresión_de_paso”, expresión que incrementa una variable.
- “sentencia”: sentencia(s) que se ejecutan en cada bucle. Si son varias sentencias se incluyen en un bloque. Los bloques se acotan mediante llaves:

```
{
    sentencia1;
    sentencia2;
    …
}
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Ejemplo bucle:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  int i=0;
  for (i=0; i<10;i++)
  {
    Serial.println(i);
  }

}
```

Inicio puerto serie

Creación de variable de incrementación ‘i’

Incrementación de ‘i’ desde 0 a 10

Envía por puerto serie el valor de ‘i’

```
0
1
2
3
4
5
6
7
8
9
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Sentencia bucle while

Ejecuta una sentencia varias veces. La *sentencia* se ejecuta una y otra vez mientras la *condición* sea cierta.

while (condición)
sentencia

Por ejemplo, podemos implementar un contador o incrementador. En este caso, la variables contador se incrementa de uno en uno, en cada ciclo hasta llegar a 20

```
int i=0;
while (i<20)
{
  i++;
  Serial.println(i);
}
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Sentencia condicional

Ejecuta una “sentencia” (simple o bloque) solo sí la “condición” es cierta:

if (condición)
sentencia o bloque
else
sentencia o bloque

Si es **falsa** se ejecuta el **else**

Si se comparan dos expresiones, se utilizarán los siguientes signos

A == B ¿A es igual a B?
A <= B ¿A es menor que B?
A >= B ¿A es mayor o igual que B?
A != B ¿A no es igual a B?

Podemos utilizar operaciones lógicas para trabajar con varias condiciones:

Y, AND: (condicion1 && condicion2) ¿Se cumplen las 2 condiciones?
O,OR: (condicion1 || condicion2) ¿Se cumple una de las condiciones?

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

En el siguiente ejemplo se diferencian los valores menores y mayores de 10

```
int i=0;
while (i<20)
{
  i++;
  if (i<10)
  {
    Serial.print("Menor de 10:");
    Serial.println(i);
  }
  else
  {
    Serial.print("Mayor o igual que 10:");
    Serial.println(i);

  }
}
```

```
Menor de 10:1
Menor de 10:2
Menor de 10:3
Menor de 10:4
Menor de 10:5
Menor de 10:6
Menor de 10:7
Menor de 10:8
Menor de 10:9
Mayor o igual que 10:10
Mayor o igual que 10:11
Mayor o igual que 10:12
Mayor o igual que 10:13
Mayor o igual que 10:14
Mayor o igual que 10:15
```

...

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Ejemplo if…else anidado, importante el uso de bloques:

```
int i=0;
while (i<20)
{
  i++;
  if (i<10)
  {
    if (i<5)
    {
      Serial.print("Menor que 5:");
      Serial.println(i);
    }
    else
    {
      Serial.print("Menor de 10 y mayor que 5:");
      Serial.println(i);

    }
  }
  else
  {
    Serial.print("Mayor o igual que 10:");
    Serial.println(i);

  }
}
```

```
Menor que 5:1
Menor que 5:2
Menor que 5:3
Menor que 5:4
Menor de 10 y mayor que 5:5
Menor de 10 y mayor que 5:6
Menor de 10 y mayor que 5:7
Menor de 10 y mayor que 5:8
Menor de 10 y mayor que 5:9
Mayor o igual que 10:10
Mayor o igual que 10:11
Mayor o igual que 10:12
Mayor o igual que 10:13
Mayor o igual que 10:14
Mayor o igual que 10:15
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Contador o incrementador

Un contador aumenta una variable un incremento determinado. En este caso se aumenta la variable 'contador' en 1

```
int contador = 0 ;
contador = contador + 1;
```

Otras formas de incrementar:

```
int contador = 0 ;
contador += 1;
```

```
int contador = 0 ;
contador ++;
```

```
int contador = 0 ;
contador += 2;
```

INTRODUCCIÓN A LA PROGRAMACIÓN. SENTENCIAS

Acumulador

Suma de los históricos de una variable. Es la acumulación de todos los valores anteriores de una variable.

Por ejemplo, queremos sumar todos los valores que de la variable 'valor'. Utilizamos una variable que vaya acumulando todos los valores de 'valor':

```
suma = suma + valor;
```

Otra forma:

```
suma += valor;
```

El código sería,

```
int suma = 0;
int valor = 0;
int i=0;
for (i=0; i<10; i++)
{
    valor = 5;
    suma += valor;
}
```

INTRODUCCIÓN A LA PROGRAMACIÓN. FUNCION LOOP DE ESP32

Vamos a hacer un contador en la función LOOP que la variable se incremente en 1 en cada ciclo (loop)

```
int varA=0;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  varA=3;

}

void loop() {
  // put your main code here, to run repeatedly:
  varA=varA+1;
  Serial.println(varA);

}
```

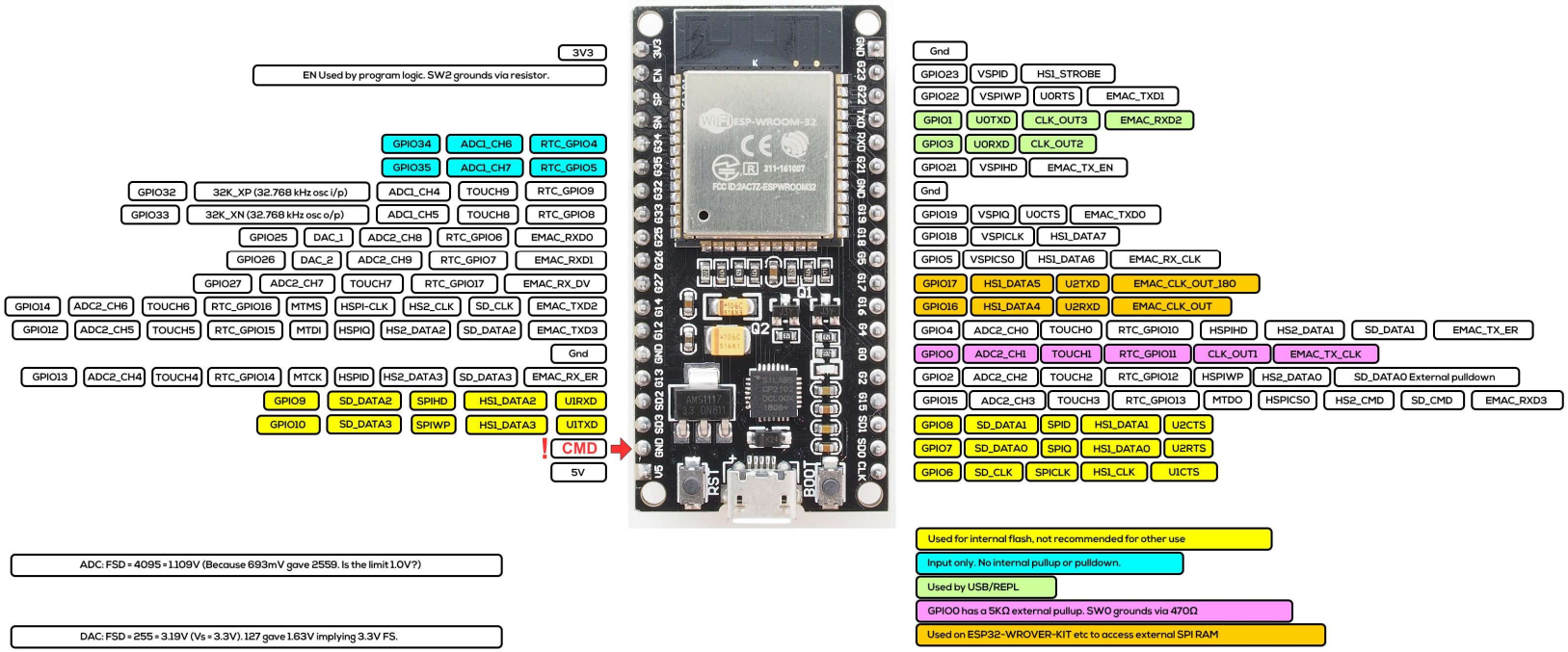
La función *delay* pausa la ejecución un tiempo definido

delay(1000) //pausa la ejecución 1000ms (1 segundo)

Programación GPIO ESP32

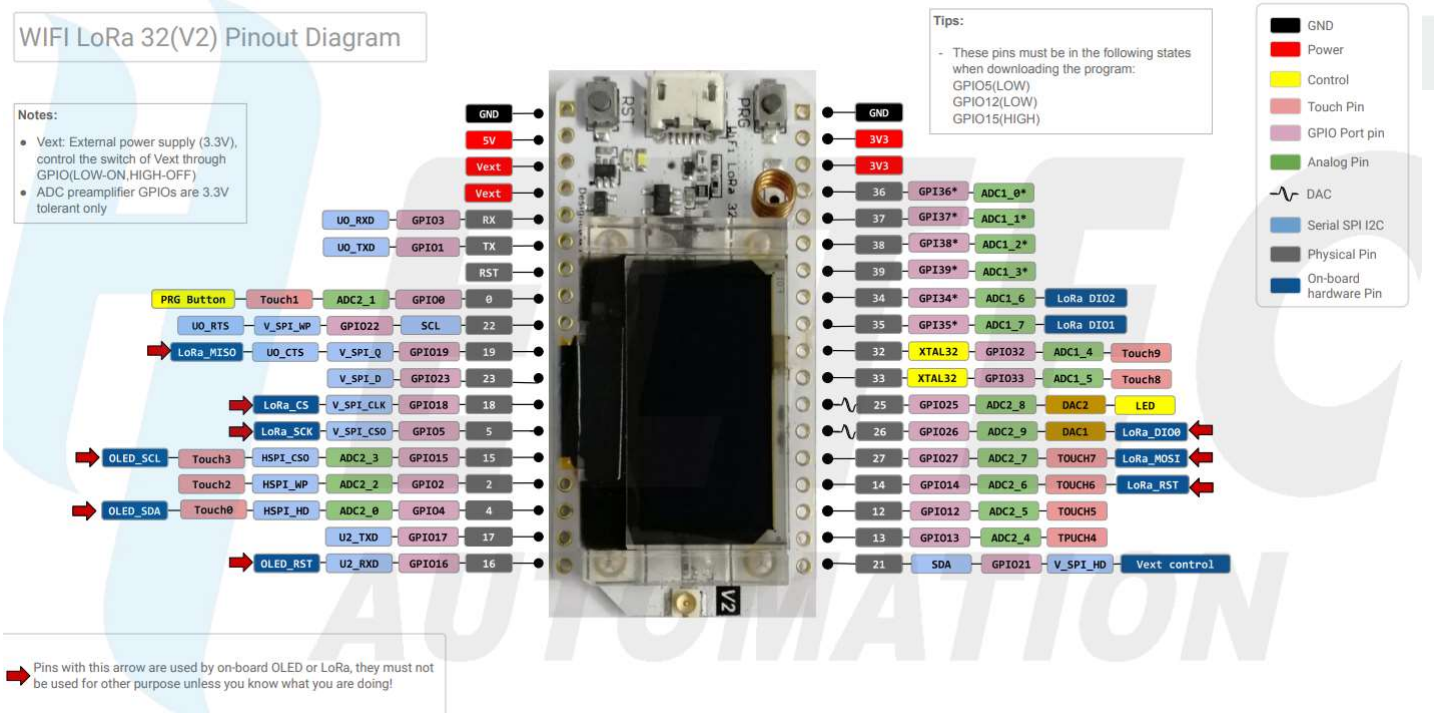
Programación GPIO ESP32

EZ-DELIVERY NodeMCU ESP32 tiene puertos de señales digitales (entrada/salida) de carácter digital. Estos puertos manejan tensiones de 0 y de 3.3V y permiten una circulación de corriente máxima de 15mA por puerto.



Programación GPIO ESP32

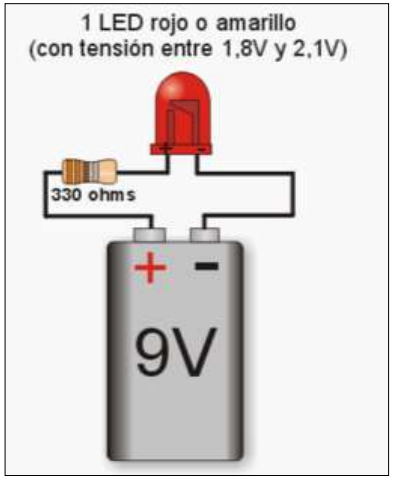
HELTEC WIFI LORA-32 tiene puertos de señales digitales (entrada/salida) de carácter digital. Estos puertos manejan tensiones de 0 y de 3.3V y permiten una circulación de corriente máxima de 15mA por puerto.

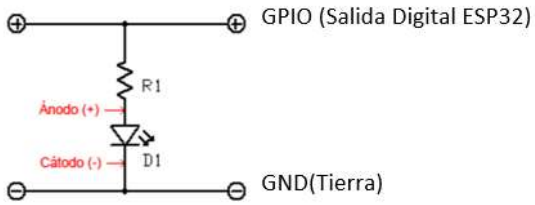


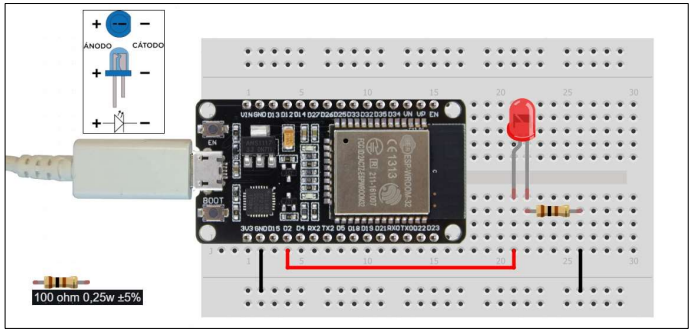
Programación GPIO ESP32

Ejemplo utilización GPIO digital:

Conexión eléctrica de un LED mediante una batería y mediante un ESP32. Utilizamos una resistencia para limitar la intensidad que circula por el circuito y la intensidad de luz.



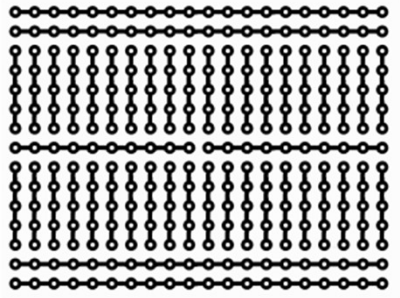


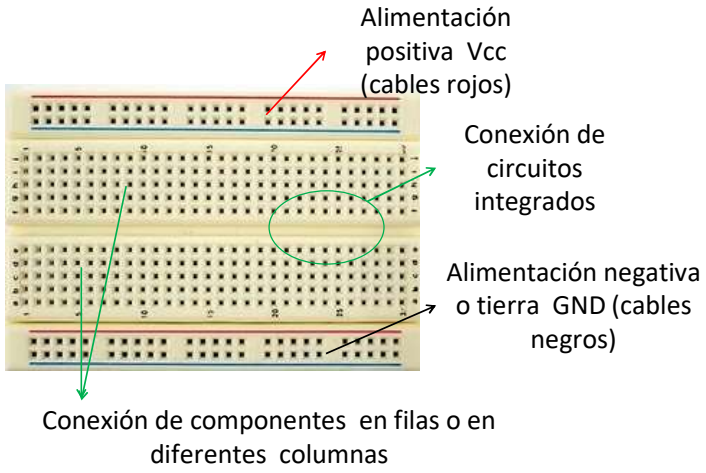


Programación GPIO ESP32

Protoboard

Son placas de prueba de uso temporal compuestas por bloques de plástico perforados y numerosas láminas delgadas que unen éstas perforaciones, creando una serie de líneas de conducción paralelas.





Programación GPIO ESP32

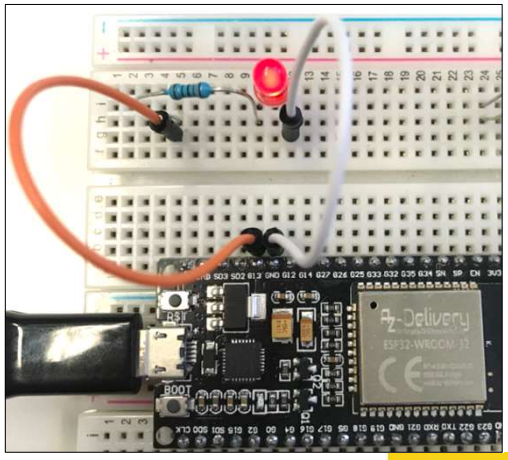
Conectamos el ánodo del LED (**pata corta**) en la salida **GPIO13**. El cátodo lo conectamos a **GND** a través de una resistencia. Si suponemos una caída de tensión de 1.5v en el LED y un intensidad máxima de salida de GPIO de 15mA, obtenemos una R de 100 Ohms:

$$V = RI$$

$$3.3 - 1.8 = R\ 0{,}015$$

$$R = 100 \text{ Ohms}$$

Esta es la resistencia más pequeña que podemos conectar. Ponemos una R de **330 Ohms**



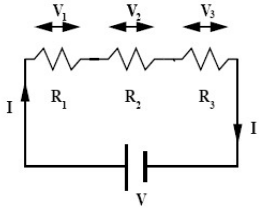
Programación GPIO ESP32

Configuramos la salida GPIO13 como OUTPUT (salida) mediante el procedimiento **pinMode**. Conmutamos la salida a valor alto y bajo cada 1s. La asignación la realizamos mediante el procedimiento **digitalWrite**. El programa quedaría así:

```
void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

  digitalWrite(13, HIGH)
  delay(1000);
  digitalWrite(13, LOW)
  delay(1000);

}
```

Programación GPIO ESP32

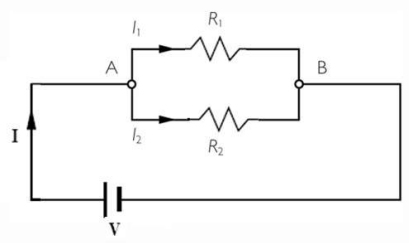
Resistencias en serie:



$$V = V_1 + V_2 + V_3$$

$$\left.\begin{aligned} V &= R_1 I + R_2 I + R_3 I = (R_1 + R_2 + R_3) I \\ V &= R_{serie} I \end{aligned}\right\} \Rightarrow R_{Serie} = (R_1 + R_2 + R_3)$$

Resistencias en paralelo:



$$I = I_1 + I_2$$

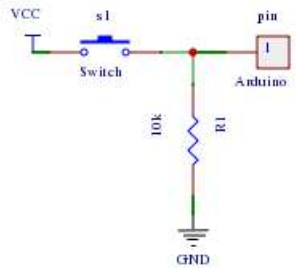
$$\left.\begin{aligned} I &= \frac{V}{R_1} + \frac{V}{R_2} \\ I &= \frac{V}{R_{Paralelo}} \end{aligned}\right\} \Rightarrow \frac{1}{R_{Paralelo}} = \frac{1}{R_1} + \frac{1}{R_2}$$

Programación GPIO ESP32

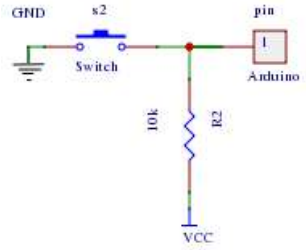
Entradas

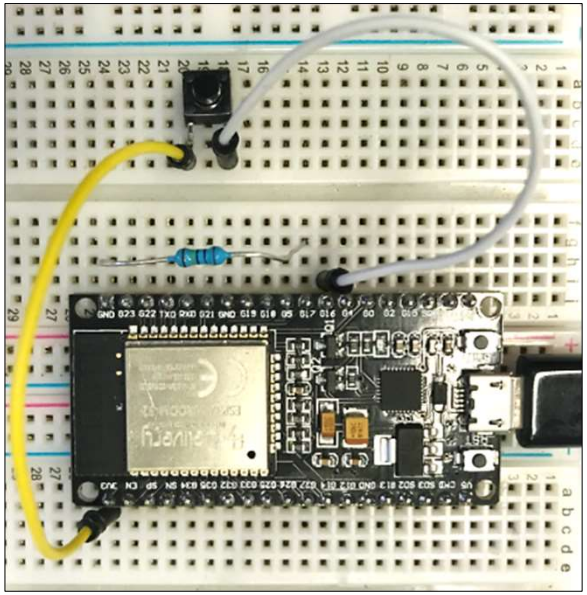
Las señales de entrada se activan/desactivan conectándolas a 3.3V o a GND mediante un interruptor (switch). La función de lectura es: **digitalRead.** Conectamos la **GPIO16**

Conexión del interruptor a 5V y a GND mediante resistencia. Al pulsar entra 5V en el pin.



Conexión del interruptor a GND y a 5V mediante resistencia. Al pulsar entra 5V en el pin.





Programación GPIO ESP32

Ejecutamos el código que lee el estado de la entrada GPIO16. Comprobar las dos configuraciones: activar o desactivar la entrada al pulsar el interruptor.

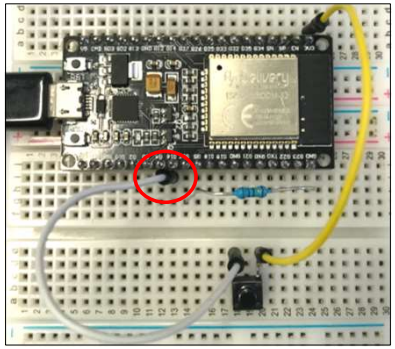
Comprobar el cambio de estado de la salida GPIO16 en la consola.

```
void setup() {
  pinMode(16, INPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:

  bool a = digitalRead(16);
  Serial.println(a);
  delay(100);
}
```

Programación Puertos analógicos ESP32

Lectura de valores analógicos

El puerto G4 tiene in conversor A/D de 12 bits. Puede manejar valores discretos de 0 a 4096.

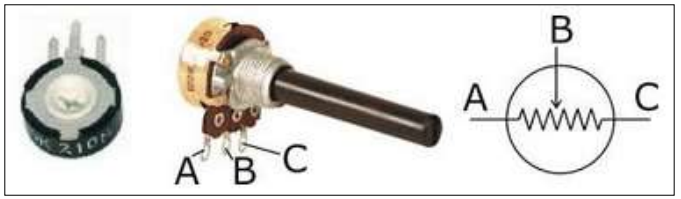


Teniendo en cuenta que el procesador trabaja con valores digitales de 0 (como valor LOW) y 5V (como valor HIGH), la resolución será:

$$\frac{3.3V}{4096} = 0.0008V = 0.8mV$$

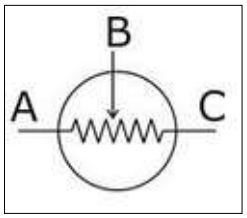
Programación Puertos analógicos ESP32

Un potenciómetro es un dispositivo que permite regular la potencia que circula por él manualmente. Para ello utiliza una resistencia variable, cuyo valor podemos ajustar mediante un mando, generalmente circular.



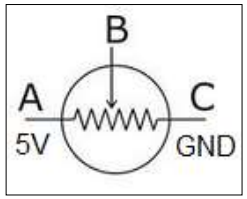
Suponemos un potenciómetro de 100kOhms circular:

- Si movemos el mando hacia un extremo la resistencia del potenciómetro será de 0Ohms
- Si movemos el mando hacia el otro extremo será de 100kOhms
- Los valores intermedios serán proporcionales a la posición.



Suponemos que conectamos A y C y medimos la tensión en B:

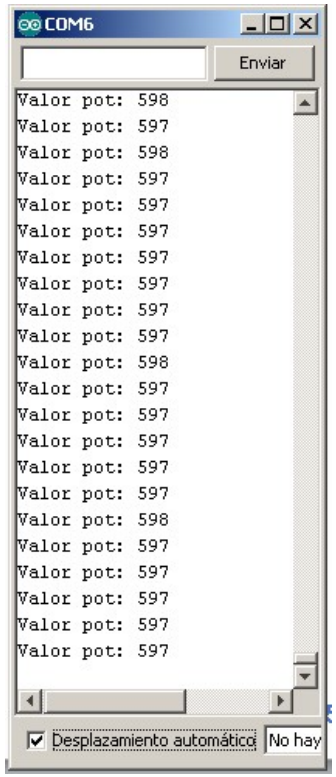
- Si movemos el mando hacia un extremo en B tenemos 0V
- Si movemos el mando hacia el otro extremo tenemos 5V
- Los valores intermedios serán proporcionales a la posición.



Programación Puertos analógicos ESP32

Vamos a hacer lecturas del potenciómetro conectando el pin B en la entrada G4, y las patas A a GND y C a 5V.

```
void loop() {
  // put your main code here, to run repeatedly:
  // this speeds up the simulation
  delay(10);
  lecturaAnaloga = analogRead(35);
  Serial.println(lecturaAnaloga);


}
```



Si movemos el potenciómetro a los extremos, obtendremos valores de 0 y de 4095. Probar la función **map** para convertir estos valores a 0 y 3.3V.