Programar ESP32 con Arduino IDE

José Guerra Carmenate

En un artículo anterior ya he hablado sobre **las ventajas y características de los chips ESP32**. Sin embargo, en ese caso no se trató nada relativo a su programación y usos en **proyectos *makers***.

Es por eso que en este artículo se describe, paso a paso, **cómo programar una placa de desarrollo basada en ESP32 utilizando Arduino IDE**. Para esto verás:

- una breve reseña sobre la **placa de desarrollo Heltec Wireless Stick Lite**.
- cómo preparar el **software Arduino IDE** para **programar cualquier placa de desarrollo basada en ESP32**.
- un ejemplo simple sobre **cómo utilizar ESP32 para implementar un servidor Web**.
- y por último, te dejo un listado de proyectos que puedes realizar con tu **placa ESP32**.

Sin más dilación, comenzamos…

Indice de contenidos

¿Qué es ESP32?

ESP32 es una **serie de SoC** (por sus siglas en inglés, *System on Chip*) y módulos de **bajo costo** y **bajo consumo de energía** creado por **Espressif Systems.**

Esta nueva familia es la sucesora del famoso **ESP8266** y su característica más notable es que además de **Wi-Fi**, también soporta **Bluetooth**.

En el mercado existen una infinidad de placas de desarrollo basadas en estos chips. Algunas **especializadas en ciertas áreas como el IoT**, las **redes de sensores** o **aplicaciones de bajo consumo** y otras de uso general.

Lo cierto es que todas permiten implementar **proyectos basados en ESP32** de forma muy simple, tal y como lo haces con una **placa Arduino** cualquiera.

Para este tutorial se utilizará la placa **Heltec Wireless Stick Lite** desarrollada y comercializada por la **compañía *Heltec Automation***.

Placa de desarrollo Heltec Wireless Stick Lite

Esta placa está basada en el chip **ESP32-PICO** que cuenta con dos **procesadores de 32-bits**, un **co-procesador** de ultra bajo consumo y **4 MiB de memoria de programa**, entre otras características típicas del los chips **ESP32**.

Está orientada al desarrollo de **aplicaciones de bajo consumo** y cuenta con comunicación **Wi-Fi**, **Bluetooth** y **LoRa**.

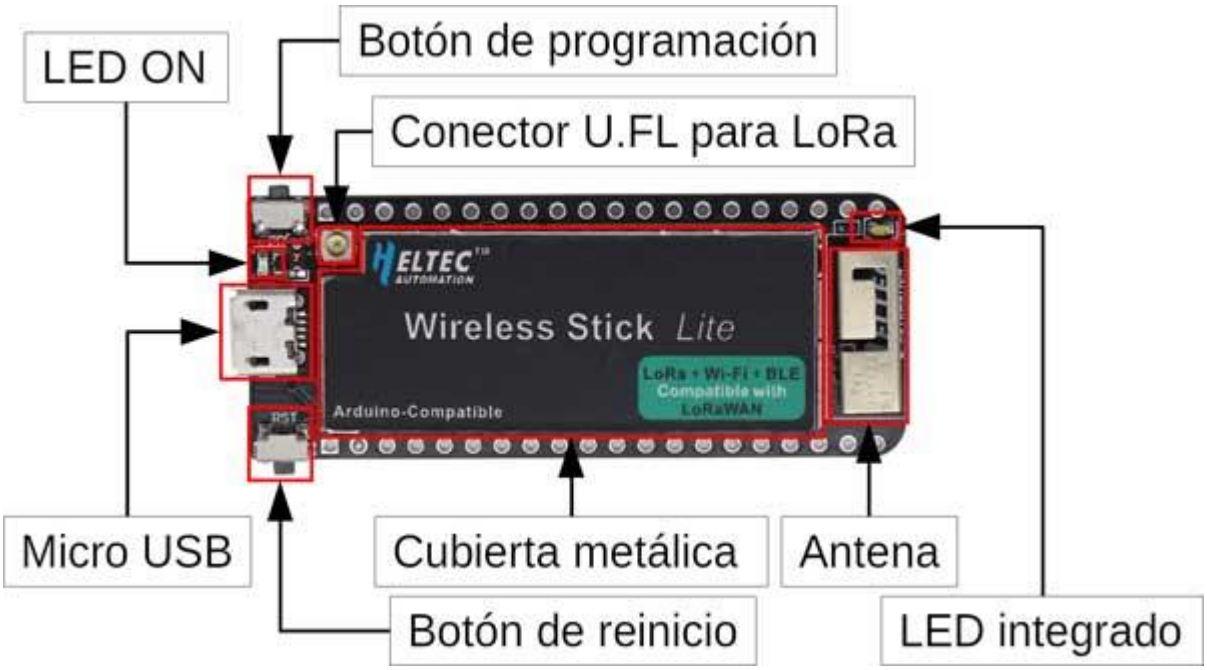
Vamos, que puedes integrar cualquier tipo de comunicación inalámbrica en tus aplicaciones con una sola placa.

Si quieres saber más sobre el **ESP32-PICO** puedes echar un vistazo al **artículo sobre ESP32** donde se muestran los detalles de este chip.

La placa cuenta con toda la circuitería necesaria para hacer funcionar al **ESP32**, incluyendo **un chip regulador de voltaje** que se encarga de suministrar al **ESP32** los 3.3 voltios necesarios para su operación. Dicho sistema **cuenta con protección contra corto-circuitos y** consumos elevados, como medida de **seguridad extra**.

Las **placas basadas en ESP32** funcionan con niveles lógicos a 3.3 voltios. Por lo tanto, **aplicar un voltaje superior a uno de los pines digitales puede dañar la placa**.

El chip principal y gran parte del circuito de esta placa se encuentran **protegidos por una cubierta metálica** (conocida usualmente como ***RF shield***) para reducir las interferencias.

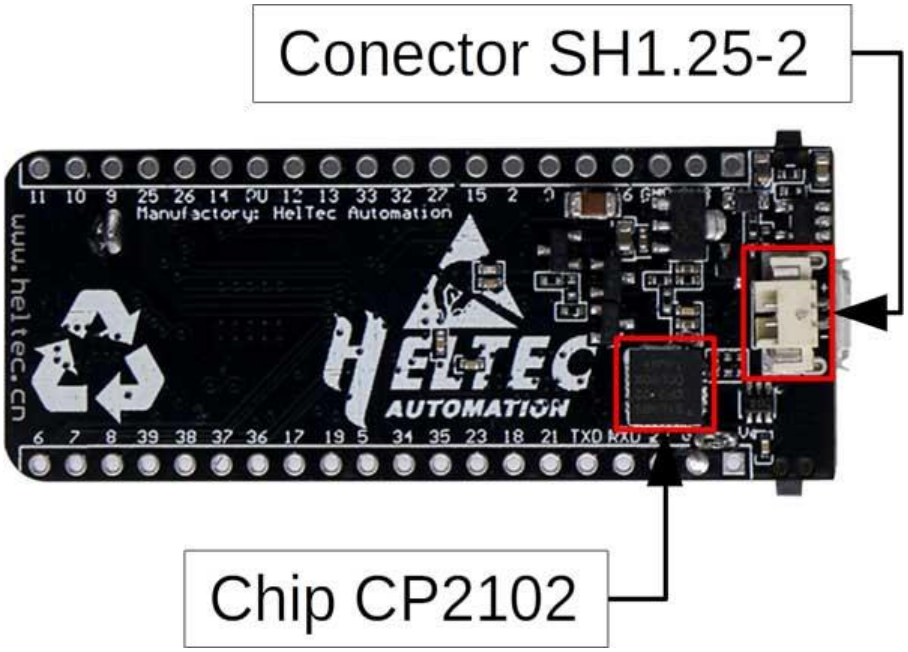


Además, en la parte superior de la placa se aprecian los siguientes componentes:

- **puerto micro USB**: listo para conectar, programar o alimentar la placa igual que un **Arduino UNO** o un **Arduino Nano**. Con la salvedad de que es un puerto USB diferente.
- **pulsador RST**: permite reiniciar el **ESP32**. Equivalente al pulsador de reinicio de las placas Arduino.
- **pulsador PROG**: utilizado a la hora de programar la placa con herramientas externas.
- **antena**: antena 3D metálica para Wi-Fi y Bluetooth.
- **conector U.FL**: este permite conectar una antena externa en caso de que se desees utilizar el **módulo LoRa** que incluye el sistema.
- **LED ON**: este LED se enciende al alimentar correctamente la placa.
- **LED integrado**: este LED está conectado al pin digital 25 y por lo tanto, puede ser controlado desde el código (más adelante se muestra un ejemplo).

No es necesario utilizar el pulsador **PROG** cuando se programa el **ESP32** desde el **IDE de Arduino**. Esto se debe, a que la propia interfaz USB se encarga de todos los procesos necesarios.

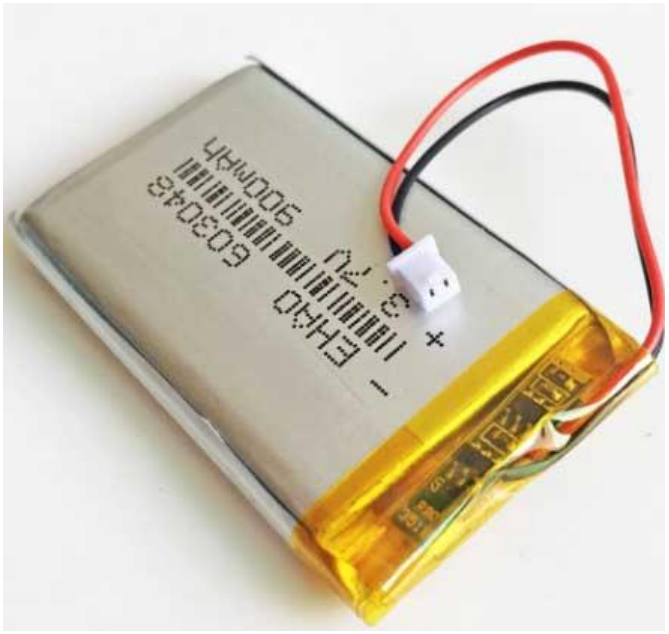
Por la parte inferior de la placa se puede apreciar dos componentes principales: el chip **CP2102** y el conector para baterías.



El **chip CP2102** actúa como puente entre **la computadora** y el **ESP32**, es decir que **permite la comunicación entre ellos** utilizando el **puerto micro USB de la placa**.

Dicho de otra manera, es el componente que facilita la vida y permite programar el **ESP32** sin necesidad de programadores u otras herramientas de hardware externas.

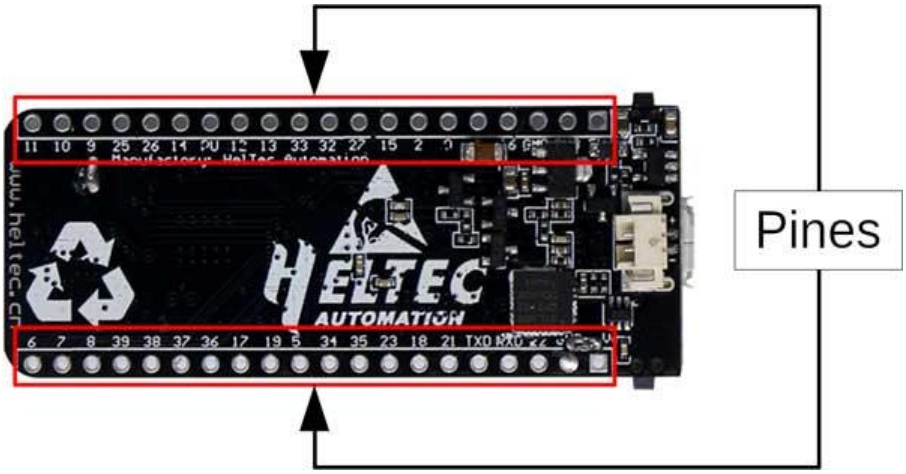
El **conector de baterías** es del tipo **SH1.25-2 para baterías Li-Po**. Conectando una batería al sistema es posible alimentar la placa sin necesidad de usar un cable USB u otra fuente de alimentación externa.



Además, la placa cuenta con el **sistema de carga para la batería**. Esto implica que no es necesario retirar la batería del sistema para cargarla. Basta conectar la placa por USB y la batería comienza a cargarse automáticamente.

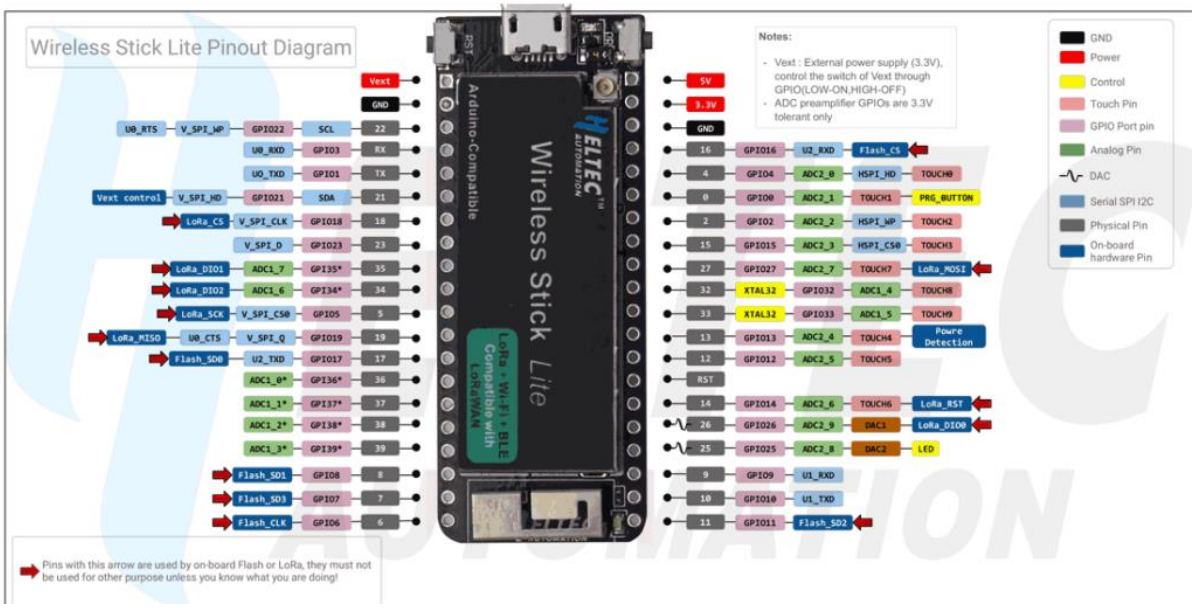
Pinout ESP32

La placa cuenta con dos filas de 20 pines cada una, ubicados a ambos lados tal y como se indica en la siguiente figura.



En la parte inferior de la placa se indica solo la numeración digital de los pines.

En la siguiente figura, ofrecida por Heltec en su documentación, se muestra la **distribución de pines** indicando todas las funcionalidades de cada uno.



Como se puede apreciar en **distribución de pines del ESP32**, la placa cuenta con **tres pines destinados a la alimentación**: vext, 5V y 3.3V. Además, tiene **dos pines de tierra** (*GND*) y **un pin de reinicio** (*RST*). Esto deja un total de 34 pines digitales que puedes utilizar en tus proyectos.

De estos pines digitales **18 pueden ser utilizados como entradas analógicas**, por lo tanto, es posible agregar un buen número de sensores analógicos a los proyectos. Los pines analógicos están indicados en verde en la imagen anterior.

Pines con otras funcionalidades

Los **pines señalados con flechas rojas** son utilizados para la **memoria de programa** y el **chip LoRa** (ambos incluidos en la placa). **Estos pines no deben ser utilizados** a no ser que sean absolutamente necesarios:

- **pines relacionados con la memoria flash**: 17, 8, 7, 6, 11 y 16.
- **pines relacionados con el chip LoRa**: 18, 35, 34, 5, 19, 26, 14, 27.

Es importante **evitar el uso de estos pines**, especialmente los que están relacionados con la memoria, ya que pueden provocar que el código no funcione correctamente.

Existen más pines que están relacionados con los componentes de la placa:

- el pin **25** está conectado al LED de la placa de forma tal que al poner un estado alto (HIGH) en este pin el LED se enciende y con un estado bajo (LOW) se apaga.
- el **pin 21** permite activar y desactivar el pin de alimentación **Vext**.
- el **pin 0** se encuentra conectado al pulsador de programación. Este puede ser utilizado como un pulsador normal en el código.
- el **pin 13** puede ser utilizado para sensar el voltaje de entrada. Esta característica es muy útil cuando el sistema es alimentado con baterías.

Pines «solo entradas» en un ESP32

Los pines 34, 35, 36, 37, 38 y 39 son pines de entrada. Esto significa que no pueden utilizarse como salida, es decir, si se intenta utilizar la función *digitalWrite()* con uno de estos pines no se obtendrá el resultado esperado.

Es recomendable reservar estos pines para dispositivos de entrada tales como: **pulsadores, teclados, sensores analógicos**, etc.

Cómo alimentar la placa de desarrollo ESP32

Existen cuatro formas de alimentar la **placa Heltec Wireless Stick Lite**:

- **conector micro USB**.
- **pin 5V**.
- **pin 3.3V**.
- **batería Li-Po**.

Alimentar ESP32 por conector micro USB

Alimentar la placa utilizando el puerto USB es lo más común cuando se está aprendiendo o en el momento de programar el **ESP32**.

En estos casos, **el pin 5V puede ser utilizado para alimentar otros componentes** que requieran una alimentación de 5 voltios. siempre teniendo en cuenta que un **consumo excesivo puede dañar el puerto del ordenador**.

De igual manera, es posible utilizar **el pin 3.3V para alimentar dispositivos** que operen a ese voltaje. Teniendo en cuenta que los 3.3 voltios son obtenidos a partir de un regulador integrado en la placa, es necesario que **el consumo no exceda los 500 mA**. De lo contrario, **el regulador, y** por tanto**, la placa, pueden resultar dañados**.

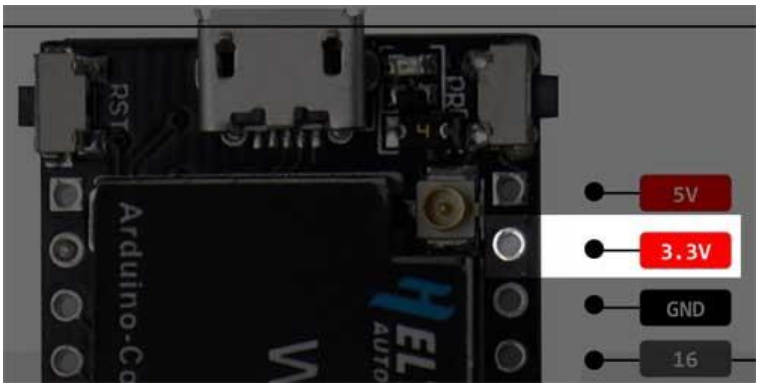
Alimentar ESP32 por Pin 5V

Por lo general, al terminar un proyecto, no es necesario conectarlo utilizando el puerto USB. En esos casos **el pin 5V puede ser utilizado para alimentar la placa**.

Es recomendable no alimentar la placa desde el **puerto USB** y el pin 5V al mismo tiempo.

En este caso también es posible utilizar el pin 3.3V para alimentar otros dispositivos externos, pero siempre teniendo en cuenta no exceder el consumo sobre los 500 mA.

Alimentar ESP32 por Pin 3.3V



Si no es necesario disponer de 5 voltios en el proyecto que estás desarrollando, **es posible utilizar una fuente de alimentación de 3.3 voltios y conectarla directamente al pin 3.3V**. En este caso, el pin 5V no puede ser utilizado para alimentar dispositivos de 5 voltios. Además, es

necesario garantizar que la fuente sea estable o de lo contrario el **ESP32** podría resultar dañado.

Alimentar ESP32 con batería Li-Po

En caso de que tu aplicación requiera ser alimentada por baterías, es posible agregar una mediante el conector de baterías. Esto permite utilizar el pin 3.3V para alimentar otros dispositivos, pero no el pin 5V.

En este caso, el conector USB o el pin 5V pueden ser utilizados para cargar la batería sin necesidad de extraerla. Basta conectar la placa a la fuente de alimentación para que el proceso de carga de la batería comience de manera automática.

Preparar Arduino IDE para programar un ESP32

Ya conoces la **distribución de pines** y **cómo alimentar tu placa de desarrollo ESP32**. Ahora solo queda aprender a cargar el código. Como verás, es muy parecido a cómo se configura un **ESP8266**. En concreto, ya vimos cómo programar una placa **NodeMCU a través del IDE de Arduino**.

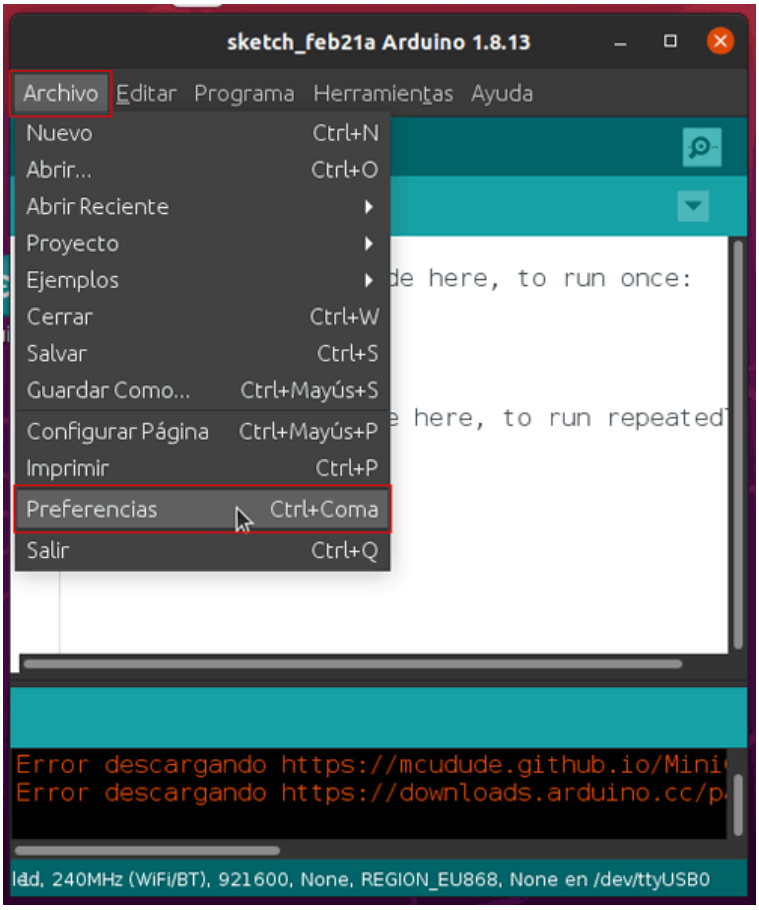
Por suerte, es posible también utilizar el **IDE de Arduino** para programar las **placas ESP32**. Pero antes de poder cargar tus códigos al **ESP32** es necesario **preparar el IDE** para esta tarea.

Ya verás como es muy parecido.

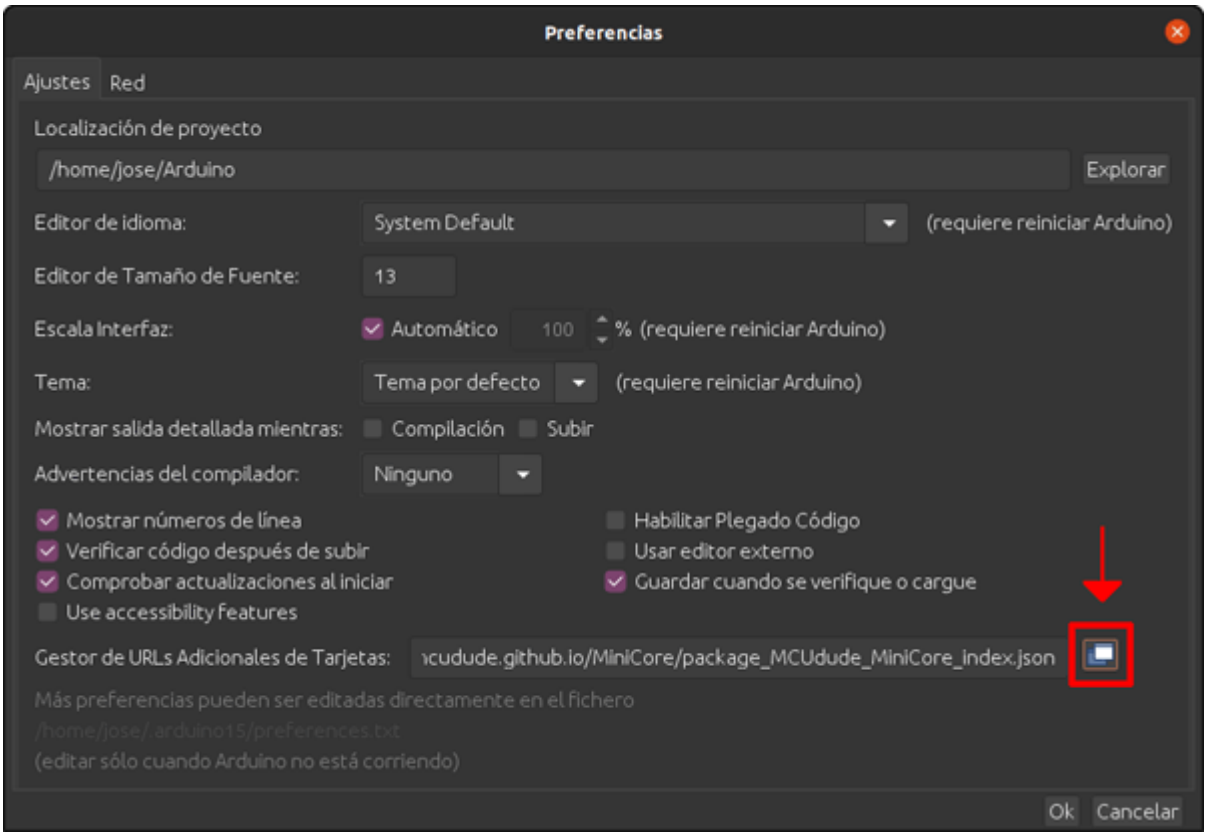
Paso 1. Adicionar las URLs para placas ESP32

Para programar un **ESP32** desde **Arduino** hay que agregar las URLs de las placas **ESP32** para poder descargar el **núcleo** (o *core*) de **ESP32** para **Arduino**.

Lo primero es ejecutar **Arduino IDE** y hacer clic en “Archivo>Preferencias”.



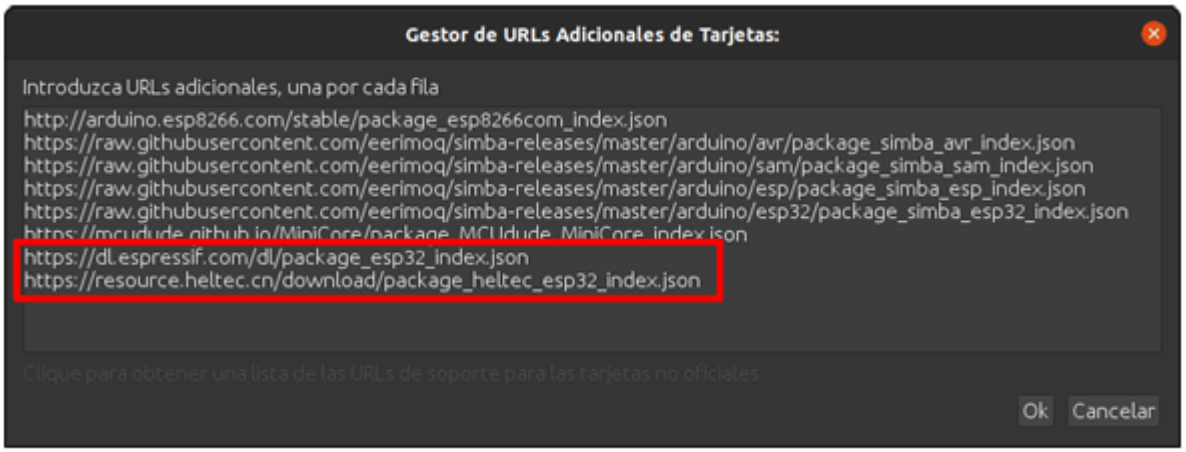
En la ventana de preferencias es necesario hacer clic en el botón "**Gestor de tarjetas adicionales**".



Ahora, en la nueva ventana se pegan las siguientes URLs:

- **https://dl.espressif.com/dl/package_esp32_index.json**: con esta dirección el gestor de placas tendrá acceso a un conjunto elevado de placas y módulos ESP32 de varios fabricantes.
- **https://resource.heltec.cn/download/package_heltec_esp32_index.json**: con esta otra el gestor de placas tendrá acceso a las placas de desarrollo ESP32 comercializadas por Heltec.

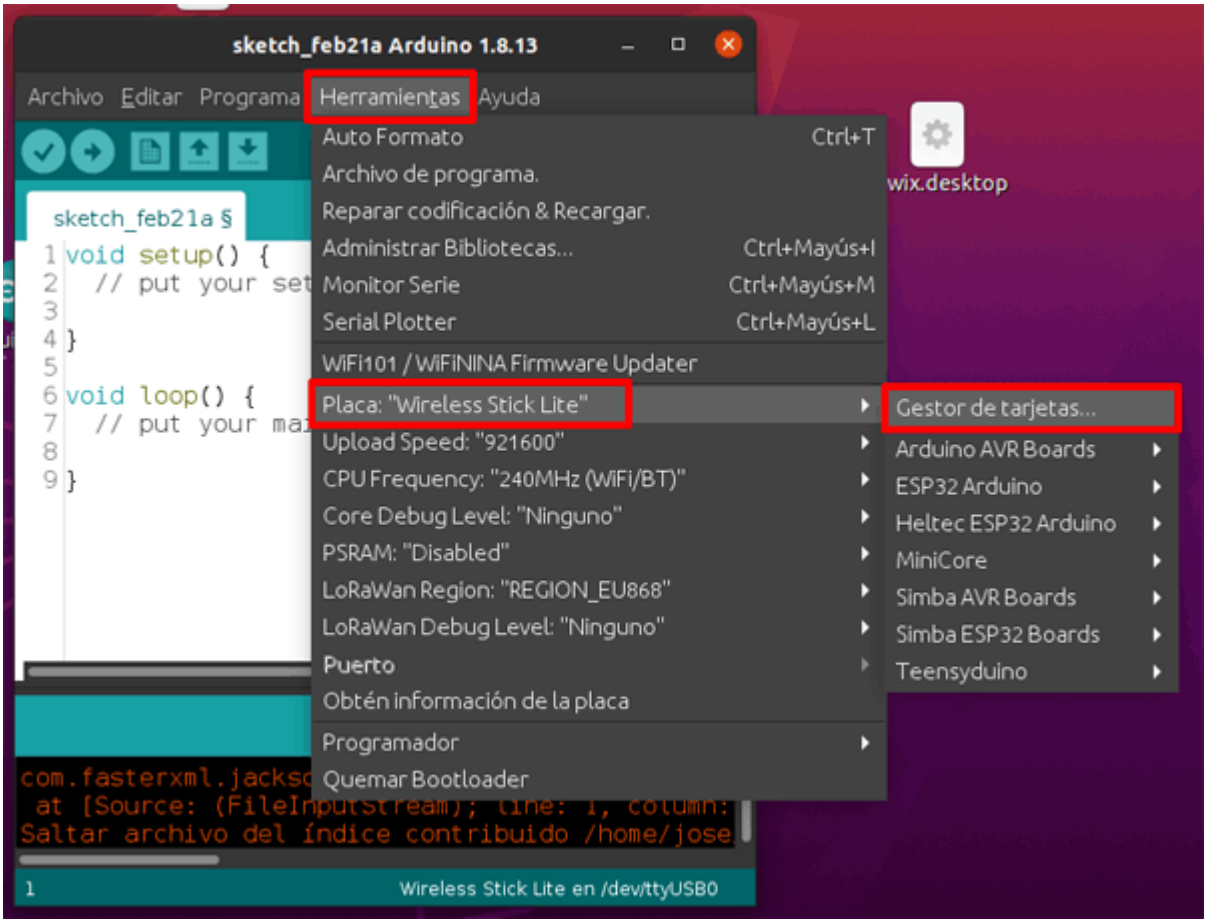
Dependiendo de la placa de desarrollo que uses, puedes agregar una u otra dirección, pero mi consejo es agregar ambas.



Una vez agregadas las URLs es necesario hacer clic en el botón “OK” de la ventana. Esto te devuelve a la ventana de preferencias, donde también tienes que hacer clic en el botón “OK”.

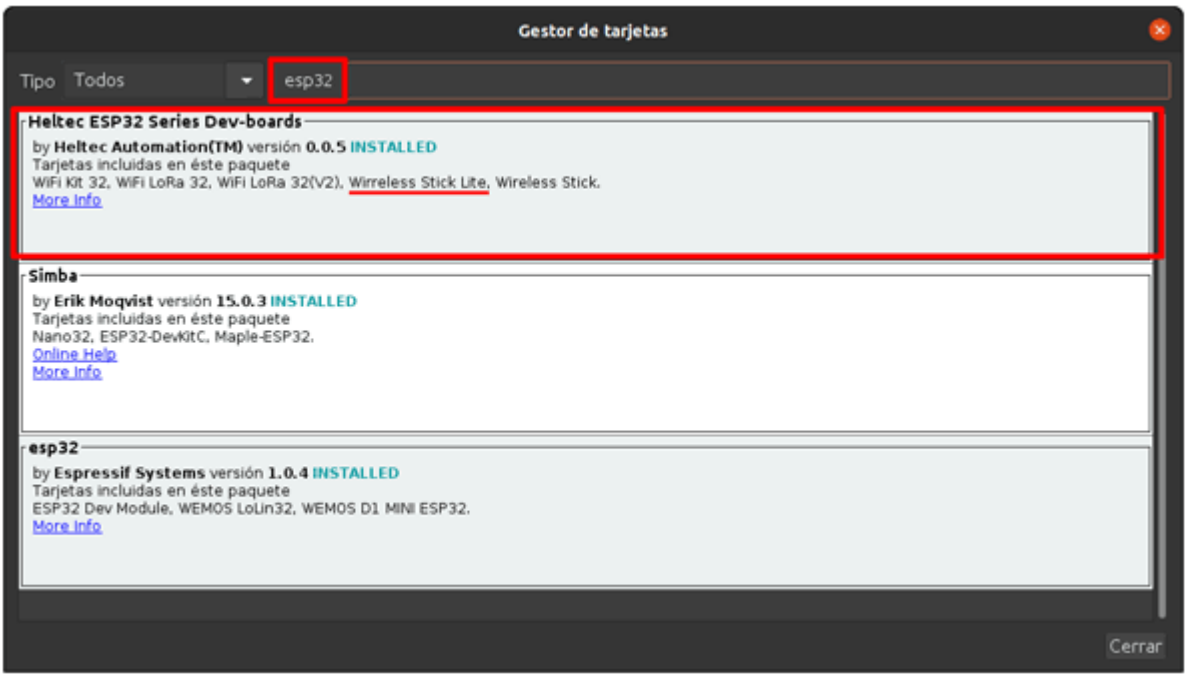
Paso 2. Instalar core y placa ESP32

Para instalar el soporte para **ESP32** y las placas de desarrollo hay que ir a “Herramientas>Placas>Gestor de Tarjetas”.



Esto abrirá el gestor de placas o tarjetas. En cuanto se inicie, comenzará a actualizar su base de datos, utilizando las URLs que se agregaron anteriormente en preferencias.

Una vez termine, hay que escribir “ **esp32** ” en la barra de búsqueda para filtrar las placas disponibles.



Ahora solo resta instalar el paquete que contiene la **placa de desarrollo** a utilizar. En este caso sería la primera opción. Ten en cuenta que este proceso puede tardar un tiempo, ya que el software tiene que descargar todos los archivos necesarios para programar el **ESP32**.

Cómo programar un ESP32 con el IDE Arduino

Una vez instalados todos los archivos necesarios es hora de **programar tu placa ESP32** .

Requisitos previos

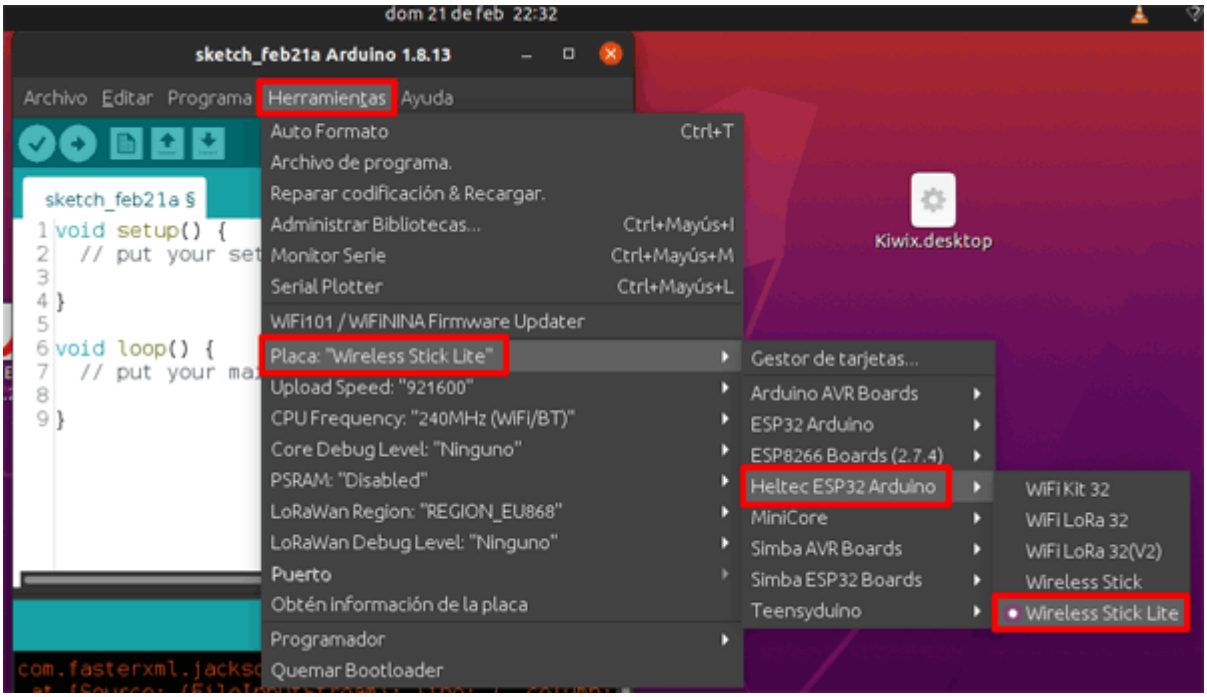
- **Arduino IDE**: es necesario tener el software instalado en tu sistema. En caso de que no lo tengas y presentes dudas con respecto a su instalación puedes consultar un artículo anterior donde se muestra **cómo instalar Arduino IDE paso a paso**.
- **Placa de desarrollo ESP32**: por supuesto, es necesario tener una placa de desarrollo basada en **ESP-32** . Aunque, el proceso es similar para todas las placas, en este caso se utiliza la ***Heltec Wireless Stick Lite***.
- **Cable USB**: aunque puede ser diferente en algunos casos, casi todas las placas de desarrollo basadas en **ESP-32** requieren un cable micro USB en un extremo (placa) y USB tipo A en el otro (ordenador). Vamos, que es el **típico cable que utilizan la mayoría de los móviles**.

Paso 1. Conectar la placa al ordenador

Para conectar tu placa **ESP32** al ordenador se utiliza el cable USB. Una vez conectado, el LED de encendido se activará indicando que la placa está correctamente alimentada. En caso de que el LED no se encienda es posible que el cable o el puerto USB estén dañados.

Paso 2. Seleccionar la placa ESP32 a utilizar

Ahora es necesario **indicar al software Arduino IDE la placa a utilizar**. Para esto es necesario ir a “Herramientas>Placas” y seleccionar la opción correspondiente. En este caso “Heltec ESP32 Arduino>Wireless Stick Lite”.



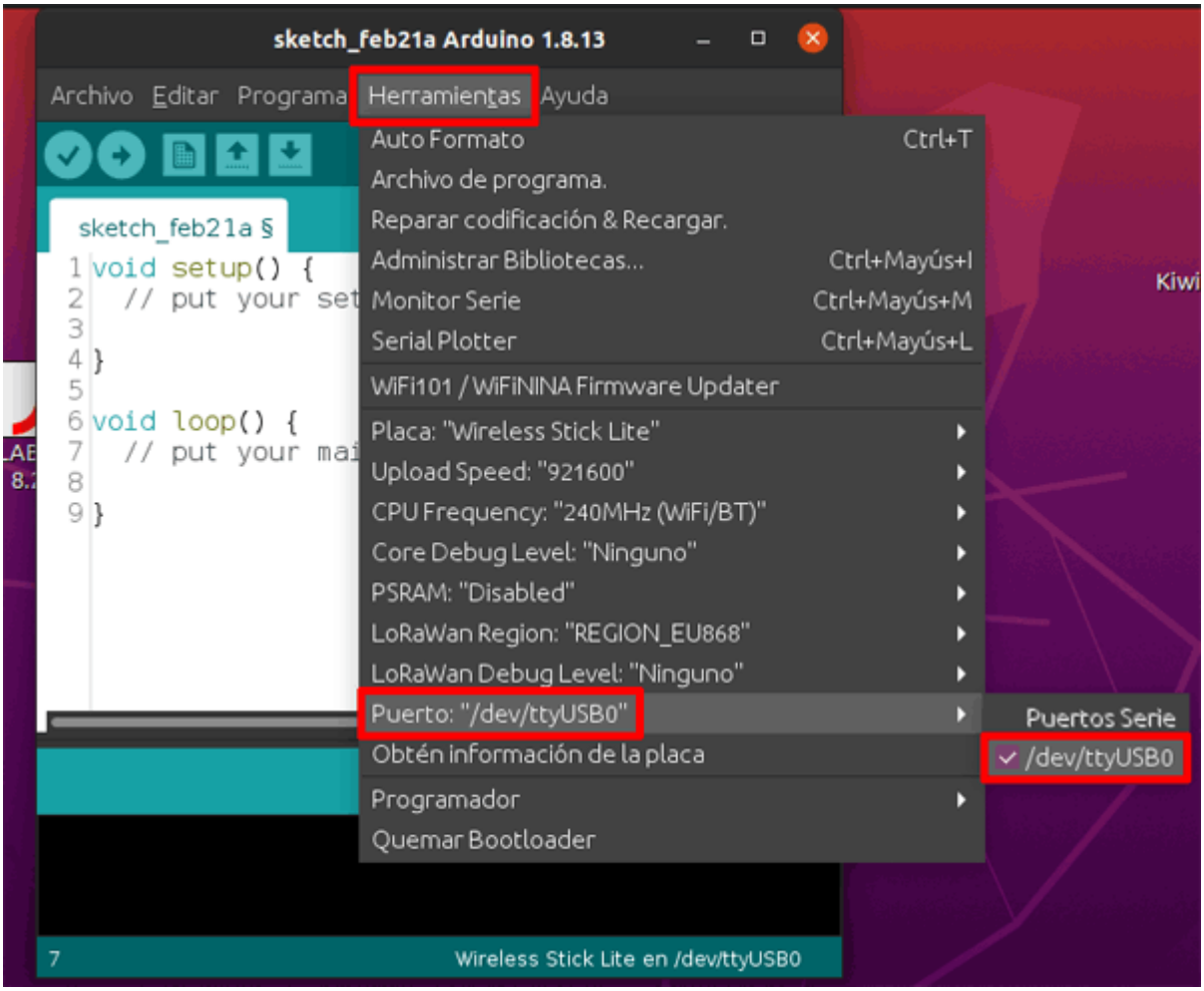
Ten en cuenta que en el menú “Herramientas>Placas>ESP32 Arduino” se encuentran un gran número de placas basadas en **ESP32**, así como variantes de estas.

Paso 3. Seleccionar el puerto

Aunque en muchas ocasiones el propio **IDE** determina el puerto empleado por la placa, en otras es necesario especificarlo. Para esto, tienes que ir a “Herramientas>Puerto”, una vez aquí es necesario seleccionar el puerto correspondiente a la placa.

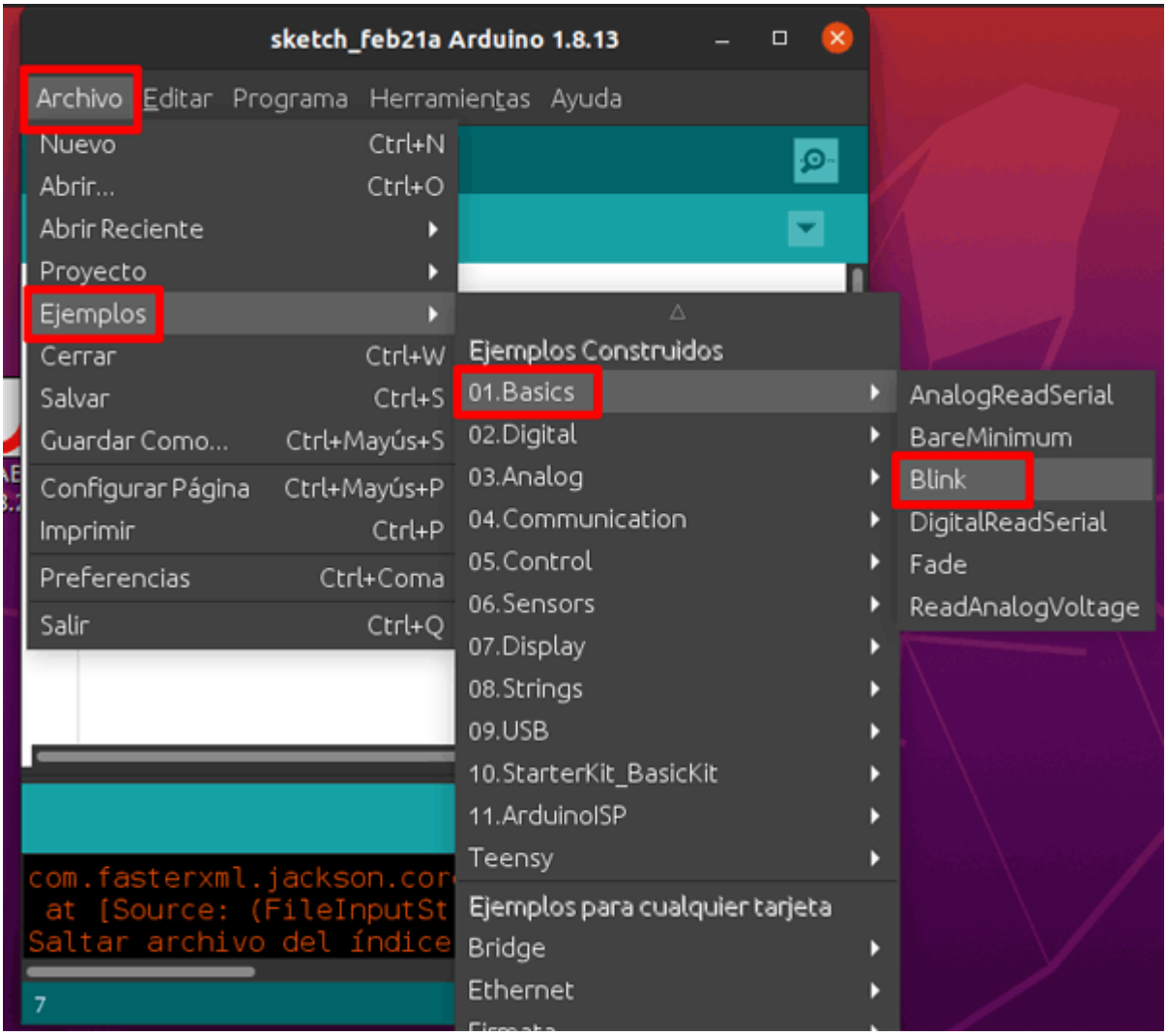
El nombre del puerto puede variar en dependencia del sistema:

- en **Windows** estará compuesto por la palabra **COM** seguida de un número, por ejemplo: COM4;
- en **Linux** comienza con **tty** y es seguida generalmente de las palabras **ACM** o **USB** con un número de orden, por ejemplo: ttyUSB0.



Paso 4. Cargar programa al ESP32

Ya solo queda cargar el código implementado al **ESP32**. Para comprobar que el proceso de grabado sea correcto, una buena opción es utilizar el ejemplo *Blink,* ya que no requieren ningún componente externo.



Una vez el código esté listo es necesario hacer clic en el botón verificar del **IDE** para comprobar que no exista ningún error en él. Luego, haciendo clic en el botón cargar, el código es grabado al **ESP32**.

Una vez termine de cargar el código el LED de la placa comenzará a pestañear.

Ejemplo de servidor web con ESP32

Veamos ahora un ejemplo más práctico, utilizar el **ESP32 como punto de acceso Wifi** e implementar un pequeño **servidor Web**. Para este proyecto necesitarás:

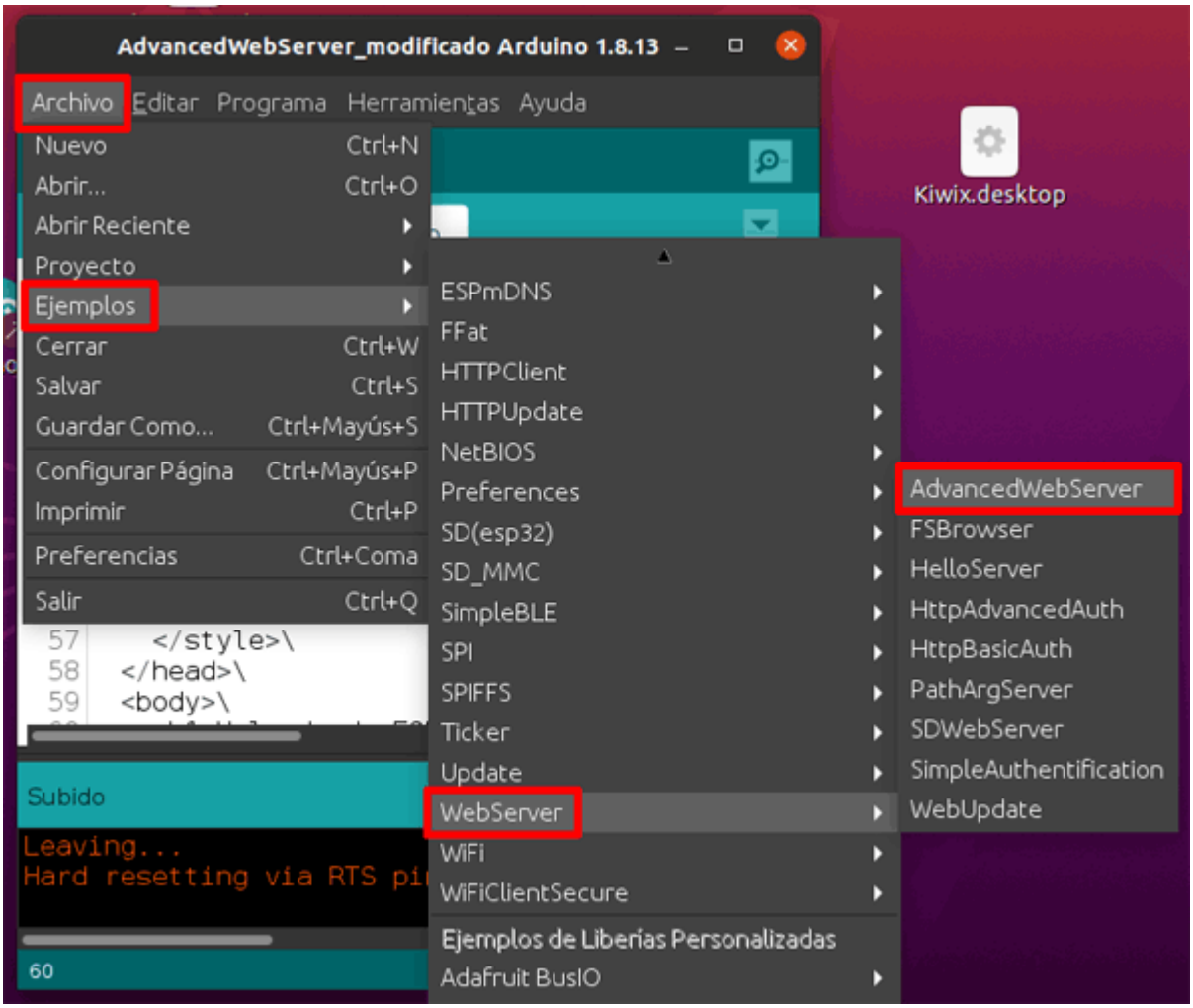
- 1x **placa de desarrollo Heltec Wireless Stick Lite** *
- 1x **cable USB**

**** Puedes utilizar cualquier otra placa basada en ESP32**, pero no olvides indicarlo correctamente en Arduino IDE.*

Para este proyecto se toma como base el ejemplo *AdvancedWebServer* disponible en **Arduino IDE**. En este ejemplo se muestra cómo programar el **ESP32** para conectarse a una red Wifi y

actuar un servidor Web. Por supuesto, la idea no es utilizar el código tal cual, sino que se le harán algunas modificaciones.

Lo primero es abrir el ejemplo. Para esto es necesario ir al menú Archivo>Ejemplos>WebServer y hacer clic en la opción *AdvancedWebServer*.



Lo primero es incluir las cabeceras necesarias. En este caso se incluyen **las cabeceras para utilizar la red Wifi** y para **implementar el servidor Web**. Aunque en el ejemplo se incluyen otras cabeceras, con la modificación realizada, no son necesarias.

```
1 // Cabeceras

2 #include <WiFi.h>

3 #include <WebServer.h>
```

Lo próximo es definir el nombre de la **red Wifi** que creará el **ESP32** y un pin con un LED conectado. En este caso yo he decidido nombrar la red “Wifi-**ESP32**” y utilizar el propio LED que trae integrado la placa.

```
1 const char *ssid = "Wifi-ESP32";
```

```
2 const int led = LED_BUILTIN;
```

Es necesario crear un **objeto de tipo WebServer** para implementar el servidor. Este se declara con un paréntesis indicando el puerto de comunicación a emplear. Se ha utilizado el puerto 80, ya que es el estandarizado para comunicaciones Web.

```
1 WebServer server(80);
```

En la **función *setup()*** inicialmente se configura el pin correspondiente al LED como salida y se pone a estado bajo para apagarlo. Además, se configura el Serial para establecer comunicación con el **monitor serie** a 115200 baudios.

```
1 void setup(void) {
2   pinMode(led, OUTPUT);
3   digitalWrite(led, LOW);
4
5   Serial.begin(115200);
```

Ahora se notifica al integrado que la **Wifi** va a ser configurada. Para esto se utiliza la función ***WiFi.softAP()*** a la que se le pasa como parámetro el **SSID de la red**.

Posteriormente se obtiene la **dirección IP** correspondiente al **ESP32** y se envía al **monitor serie**.

```
1   Serial.println("Configurando punto de acceso...");
2
3   WiFi.softAP(ssid);
4
5   IPAddress myIP = WiFi.softAPIP();
6   Serial.print("AP IP address: ");
7   Serial.println(myIP);
```

Estando la **Wifi** configurada es hora de preparar el **servidor Web**. Para esto es necesario indicar qué debe hacer cuando un cliente intente acceder a cada recurso.

Esto se realiza mediante la instrucción ***server.on()*** indicando primero el recurso y después la rutina a ejecutar. En este caso se han asignado rutinas a tres recursos:

- **“/”: handleRoot()**
- **«/test.svg»: drawGraph()**

- **“/inline”: handleInline()**

Más adelante se analiza para que sirve cada una de estas funciones.

Además, también se utiliza la instrucción ***server.onNotFound()*** para especificar qué acción realizar cuando se intente acceder a un recurso no válido.

```
1  server.on("/", handleRoot);
2  server.on("/test.svg", drawGraph);
3  server.on("/inline", handleInline);
4  server.onNotFound(handleNotFound);
```

La **función *setup()*** finaliza activando el **servidor Web**. Lo indica mediante el **monitor serie**.

```
1  server.begin();
2
3  Serial.println("HTTP server started");
4 }
```

En la **función *loop()*** simplemente se ejecuta la instrucción ***server.handleClient()***. Esta función es la que se encarga de comprobar las peticiones provenientes de los clientes.

```
1 void loop(void) {
2  server.handleClient();
3 }
```

La **función *handleRoot()*** es ejecutada cuando un cliente intenta acceder a la raíz del servidor, es decir, cuando en el navegador se escriba solo la **dirección IP de la placa**.

Lo primero que se realiza en esta función es encender el LED para indicar que se está procesando una petición. Seguidamente se declaran todas las variables necesarias:

- ***temp***: es un arreglo de caracteres que se utiliza para almacenar la respuesta a enviar al cliente.
- ***sec***: es utilizada para almacenar los segundos transcurridos desde que la placa fue energizada.
- ***min***: es utilizada para almacenar los minutos transcurridos desde que la placa fue energizada.
- ***hora***: es empleada para almacenar las horas transcurridas desde que la placa fue energizada.

```
1 void handleRoot() {
2   digitalWrite(led, HIGH); // encender LED
3   char temp[400];
4   int sec = millis() / 1000;
5   int min = sec / 60;
6   int hr = min / 60;
```

Lo siguiente es almacenar el **código HTML** a enviar al cliente en la variable ***temp***. Para esto se utiliza la función ***snprintf()*** que trabaja de forma similar a la clásica función ***printf()*** de lenguaje C, con la salvedad de que la salida es enviada a una **cadena de caracteres**, en este caso ***temp***.

```
1   snprintf(temp, 400,
2            "<html>\
3   <head>\
4     <meta http-equiv='refresh' content='5'/>\
5     <title>ESP32 Demo</title>\
6     <style>\
7       body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color:
    #000088; }\
8     </style>\
9   </head>\
10  <body>\
11    <h1>Hola desde ESP32!</h1>\
12    <p>Uptime: %02d:%02d:%02d</p>\
13    <img src=\"/test.svg\" />\
14  </body>\
15  </html>",
16           hr, min % 60, sec % 60);
```

Por último, la función envía una respuesta al cliente indicando:

- la solicitud fue procesada correctamente (código 200)

- el contenido de la respuesta (*text/html*)
- la respuesta (*temp*).

Antes de terminar, el LED es apagado nuevamente, indicando que ya la respuesta fue enviada al cliente.

```
1  server.send(200, "text/html", temp);
2  digitalWrite(led, LOW);
3 }
```

La **función *drawGraph()*** es ejecutada cuando se intenta acceder al recurso "***/test.svg***".

Esta comienza declarando las variables a utilizar:

- ***out*****:** utilizada para almacenar la respuesta a enviar al cliente.
- ***temp*****:** utilizada como almacenamiento temporal a la hora de generar la respuesta a enviar al cliente.

```
1 void drawGraph() {
2  String out = "";
3  char temp[100];
```

A continuación, se genera una gráfica **SVG** utilizando valores aleatorios.

```
1   out += "<svg xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\" width=\"400\"
height=\"150\">\n";
2
3   out += "<rect width=\"400\" height=\"150\" fill=\"rgb(250, 230, 210)\" stroke-
width=\"1\" stroke=\"rgb(0, 0, 0)\" />\n";
4   out += "<g stroke=\"black\">\n";
5   int y = rand() % 130;
6   for (int x = 10; x < 390; x += 10) {
7     int y2 = rand() % 130;
8     sprintf(temp, "<line x1=\"%d\" y1=\"%d\" x2=\"%d\" y2=\"%d\" stroke-width=\"1\"
9  />\n", x, 140 - y, x + 10, 140 - y2);
10    out += temp;
11    y = y2;
```

```
    }

    out += "</g>\n</svg>\n";
```

Por último, la función envía la respuesta al cliente mediante la instrucción ***server.send()*** indicando:

- la solicitud fue procesada correctamente (código 200)
- el contenido de la respuesta (*image/svg+xml*)
- la respuesta (*out*).

```
1  server.send(200, "image/svg+xml", out);
2 }
```

La **función *handleInline()*** es más bien simple. Lo único que realiza es enviar una respuesta en texto plano indicando "**Usted ha accedido al recurso inline» "** .

```
1 void handleInline(){
2  server.send(200, "text/plain", "Usted a accedido al recurso ``inline''");
3 }
```

Por último, **la función *handleNotFound()*** es ejecutada cuando el recurso solicitado por el navegador o cliente no coincide con ninguno de los anteriores.

Lo primero que realiza la función es encender el LED indicando que está procesando una petición. Luego, comienza a construir el mensaje de respuesta al cliente. Dicho mensaje contiene los parámetros fundamentales de la **petición HTTP** realizada por el cliente.

Finalmente la respuesta es enviada y el LED apagado.

```
1  void handleNotFound() {
2    digitalWrite(led, HIGH);
3    String message = "Archivo no encontrado\n\n";
4    message += "URI: ";
5    message += server.uri();
6    message += "\nMethod: ";
7    message += (server.method() == HTTP_GET) ? "GET" : "POST";
8    message += "\nArguments: ";
```

```
9    message += server.args();
10   message += "\n";
11
12   for (uint8_t i = 0; i < server.args(); i++) {
13    message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
14   }
15
16   server.send(404, "text/plain", message);
17   digitalWrite(led, LOW);
18 }
```

Y ese es todo el código. Aquí te lo dejo completo para que lo puedas cargar a tu placa o incluso modificarlo a tu antojo.

```
1    /*
2     * Ejemplo de servidor Web para ESP32
3     */
4    #include <WiFi.h>
5    #include <WebServer.h>
6
7    const char *ssid = "Wifi-ESP32";
8    const int led = LED_BUILTIN;
9
10   WebServer server(80);
11
12   void handleRoot() {
13    digitalWrite(led, HIGH); // encender LED
14    char temp[400];
15    int sec = millis() / 1000;
```

```
16   int min = sec / 60;
17   int hr = min / 60;
18
19   snprintf(temp, 400,
20            "<html>\
21   <head>\
22     <meta http-equiv='refresh' content='5'/>\
23     <title>ESP32 Demo</title>\
24     <style>\
25       body { background-color: #cccccc; font-family: Arial, Helvetica, Sans-Serif; Color:
     #000088; }\
26     </style>\
27   </head>\
28   <body>\
29     <h1>Hola desde ESP32!</h1>\
30     <p>Uptime: %02d:%02d:%02d</p>\
31     <img src=\"/test.svg\" />\
32   </body>\
33 </html>",
34           hr, min % 60, sec % 60
35          );
36   server.send(200, "text/html", temp);
37   digitalWrite(led, LOW);
38 }
39
40 void handleNotFound() {
41   digitalWrite(led, HIGH);
42
```

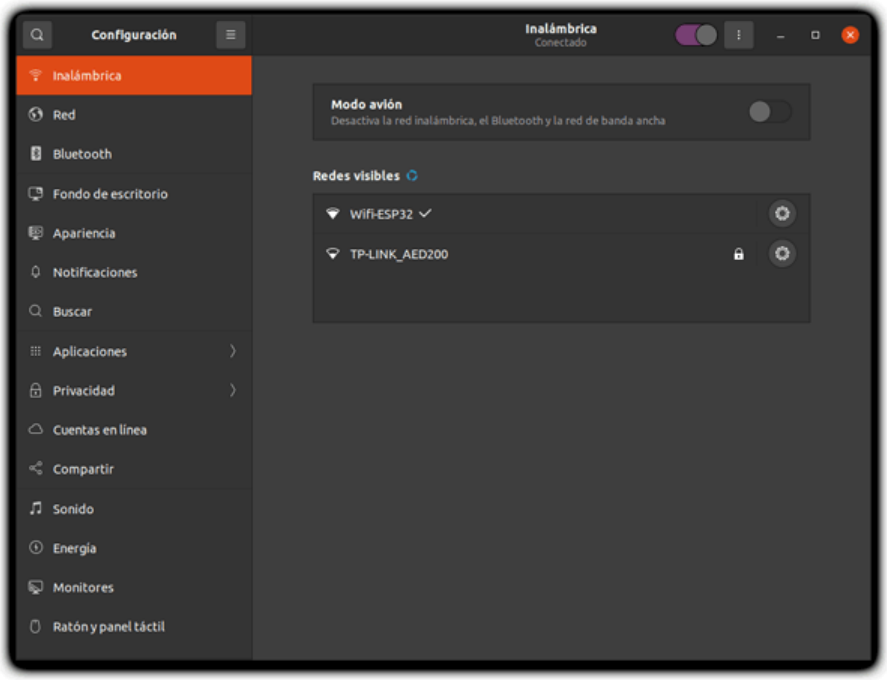
```
43    String message = "Archivo no encontrado\n\n";
44    message += "URI: ";
45    message += server.uri();
46    message += "\nMethod: ";
47    message += (server.method() == HTTP_GET) ? "GET" : "POST";
48    message += "\nArguments: ";
49    message += server.args();
50    message += "\n";
51
52    for (uint8_t i = 0; i < server.args(); i++) {
53      message += " " + server.argName(i) + ": " + server.arg(i) + "\n";
54    }
55
56    server.send(404, "text/plain", message);
57    digitalWrite(led, LOW);
58  }
59
60  void handleInline(){
61    server.send(200, "text/plain", "Usted a accedido al recurso ``inline''");
62  }
63
64  void setup(void) {
65    pinMode(led, OUTPUT);
66    digitalWrite(led, LOW);
67
68    Serial.begin(115200);
69    Serial.println();
```

```
70    Serial.println("Configurando punto de acceso...");
71
72    WiFi.softAP(ssid);
73
74    IPAddress myIP = WiFi.softAPIP();
75    Serial.print("AP IP address: ");
76    Serial.println(myIP);
77
78    server.on("/", handleRoot);
79    server.on("/test.svg", drawGraph);
80    server.on("/inline", handleInline);
81    server.onNotFound(handleNotFound);
82    server.begin();
83
84    Serial.println("HTTP server started");
85  }
86
87  void loop(void) {
88    server.handleClient();
89  }
90
91  void drawGraph() {
92    String out = "";
93    char temp[100];
94    out += "<svg xmlns=\"http://www.w3.org/2000/svg\" version=\"1.1\" width=\"400\"
      height=\"150\">\n";
95
96    out += "<rect width=\"400\" height=\"150\" fill=\"rgb(250, 230, 210)\" stroke-
      width=\"1\" stroke=\"rgb(0, 0, 0)\" />\n";
```

```
97    out += "<g stroke=\"black\">\n";
98    int y = rand() % 130;
99    for (int x = 10; x < 390; x += 10) {
100     int y2 = rand() % 130;
101     sprintf(temp, "<line x1=\"%d\" y1=\"%d\" x2=\"%d\" y2=\"%d\" stroke-width=\"1\"
      />\n", x, 140 - y, x + 10, 140 - y2);
102     out += temp;
103     y = y2;
104    }
105    out += "</g>\n</svg>\n";
106
       server.send(200, "image/svg+xml", out);
      }
```

Una vez cargues el código a la placa es hora de comprobar que funciona correctamente. Lo primero es abrir el monitor serie para obtener la dirección IP del **ESP32**. Luego es necesario utilizar tu teléfono móvil u ordenador y conectarte a la red Wifi creada por el **ESP32**.

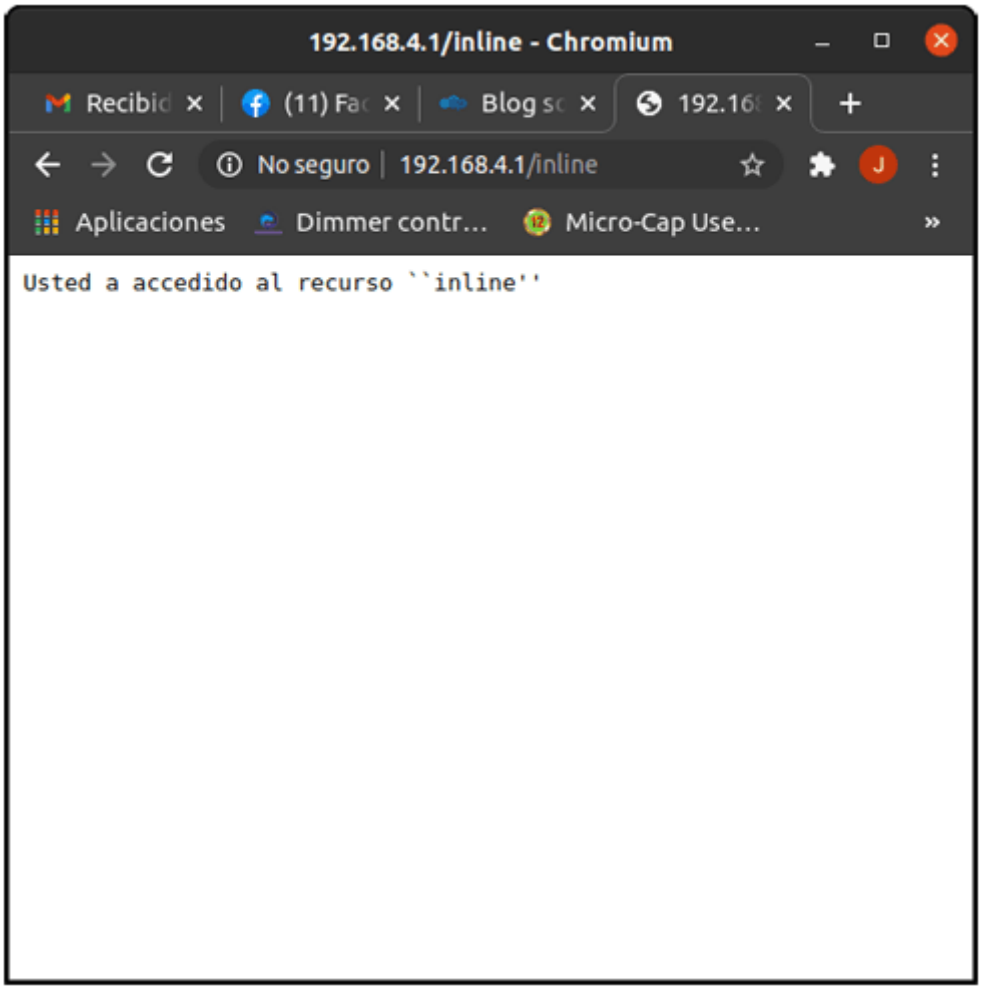
Al revisar las redes disponibles deberías obtener algo similar a lo mostrado en la siguiente imagen.



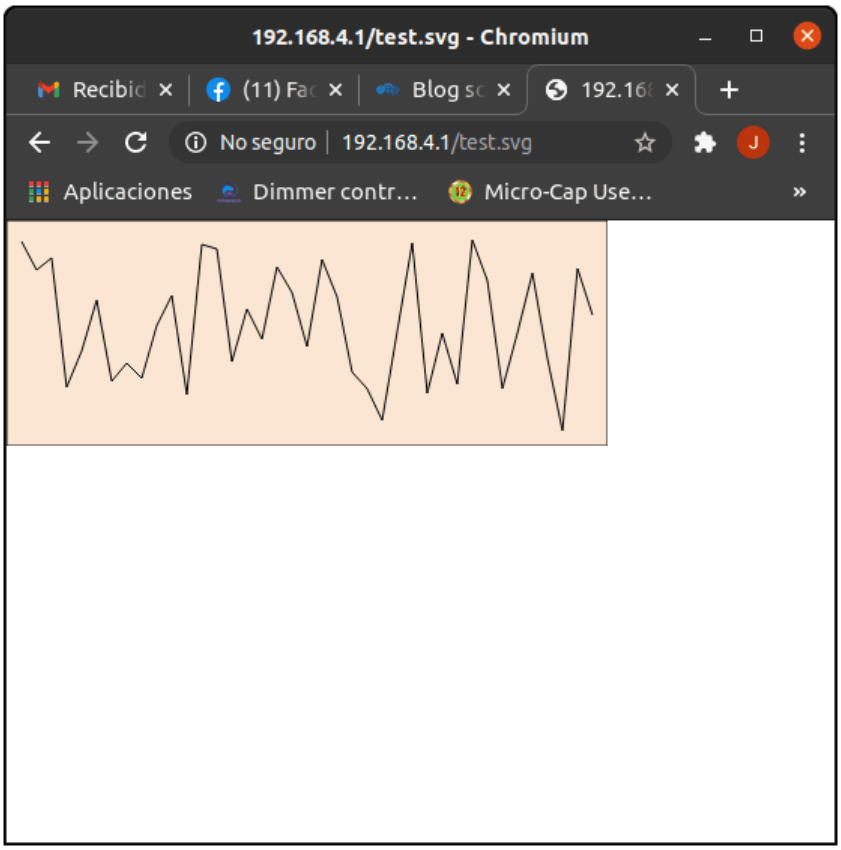
Ahora solo tienes que conectarte a la red "***Wifi-ESP32***" y abrir un navegador. No es necesario ninguno en particular, aunque te recomiendo **Firefox** o **Chrome**.

Una vez tengas el navegador abierto teclea en la barra de búsqueda la dirección IP del **ESP32**. Luego presiona *Enter* y obtendrás el siguiente resultado.

Ahora, si le agregas al final "/inline" el resultado será el siguiente.



Si en cambio utilizas “/test.svg” solo obtienes la gráfica, tal y como puedes observar en la figura.



Si te animas, puedes probar a modificar el código para que muestre otros mensajes o incluso habilitar otros recursos.

Si quieres obtener algunos conocimientos básicos sobre la **arquitectura cliente servidor** y las peticiones HTTP te recomiendo que des un vistazo a los artículos **parte 1** y **parte 2** **sobre Ethernet Shield** presentes en el blog.

Proyectos destacados con ESP32

Ahora ya sabes todo lo necesario para comenzar a desarrollar tus propios **proyectos con ESP32**. De cualquier manera, aquí te dejo algunas ideas que pueden servir de inspiración:

- **publicar lecturas de sensores en plataformas con MQTT**: en este tutorial se muestra cómo publicar lecturas obtenidas desde un **ESP32** en cualquier **plataforma compatible con MQTT**.
- **control Web con ESP32**: En este artículo se muestra cómo controlar un **ESP32** desde tu móvil u ordenador utilizando el navegador web.
- **control de ESP32 a través de Telegram**: Si buscas una alternativa de control remoto con tus **ESP32**, quizá esta sea una de las mejores opciones.
- **estación meteorológica basada en ESP32**:Si quieres construir tu propia estación meteorológica basada en **ESP32** este tutorial puede ayudarte.