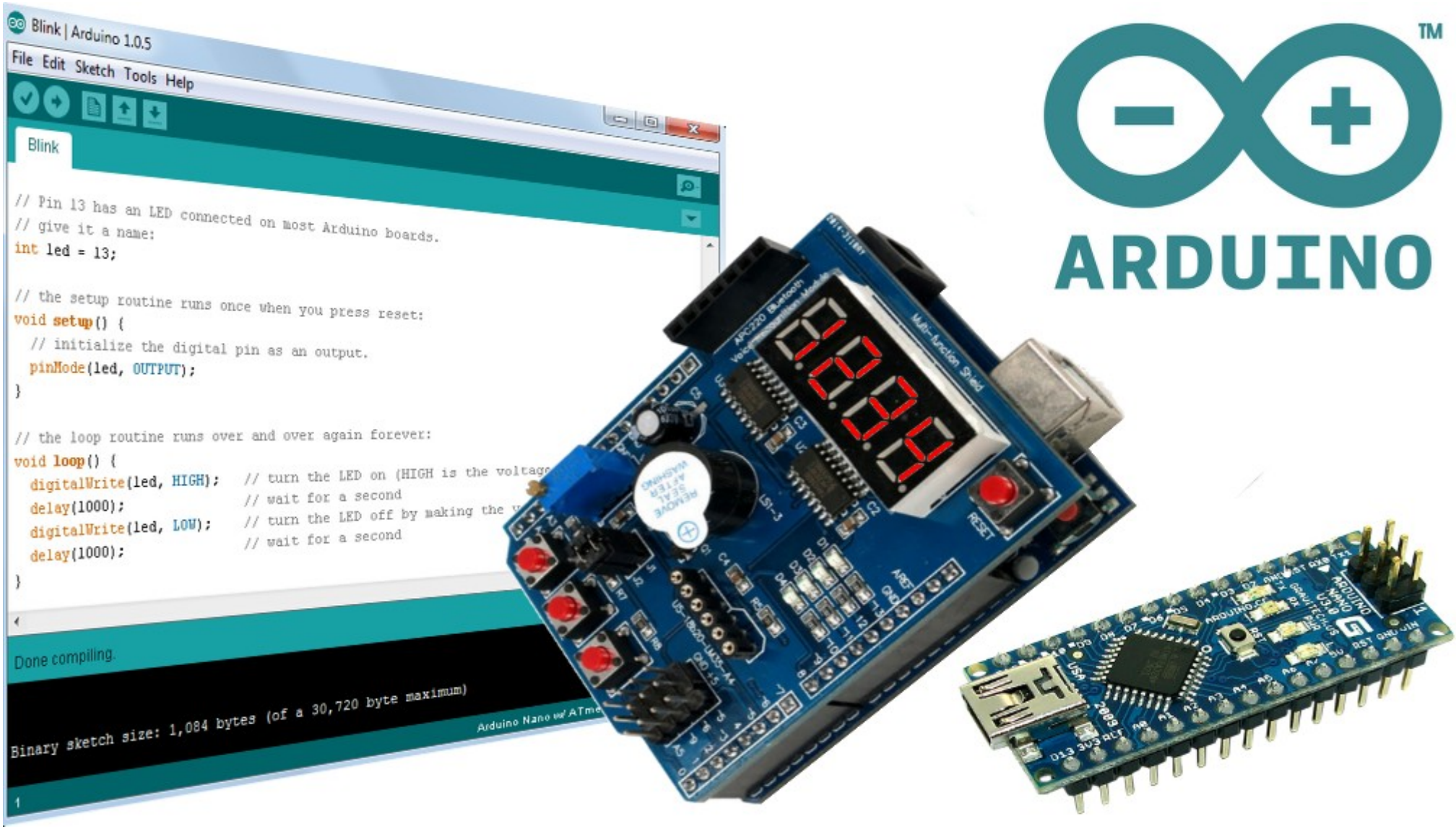
Arduino tanfolyam kezdőknek és haladóknak



16. ESP8266 alapú webszerver – 1. rész

Felhasznált és ajánlott irodalom

- Leírások, dokumentáció:
  - ESP8266 Community: ESP8266 Arduino Core's documentation
  - W3Schools: HTML Tutorial
  - W3Schools: CSS Tutorial
  - W3Schools: AJAX Tutorial
- Arduino programkönyvtárak:
  - Async TCP Library for ESP8266 Arduino
  - ESPAsyncWebServer
- Mintaprogramok:
  - Last Minute Engineers:
    Create A Simple ESP8266 Weather Station With BME280
  - Circuits4you:
    ESP8266 (ajax) update part of web page without refreshing

A csatlakozások titkos adatai

- Az **ESP8266**-t a helyi hálózatra kliensként csatlakoztatjuk, az ehhez szükséges személyes adatokat kiszerveztük egy **secrets.h** nevű fejléc állományba, amelyet a **Vázlatfüzet** (Sketchbook) mappa **libraries/secrets** almappájában helyeztünk el

```
#include <ESP8266WiFi.h>
#include "secrets.h"

void setup_wifi() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

```
#define WIFI_SSID  "MY_SSID"
#define WIFI_PASS  "MY_PASSWORD"
```

Webszerver

- **Webszerver:** olyan Internet kapcsolattal rendelkező eszköz, amely **HTTP kéréseket** fogad és szolgál ki *(a **GET** és **POST** kérések formátumát és összehasonlításukat lásd a 2021. ápr. 15-i előadásban)*
- **HTTP status kódok:** A HTTP kérésre küldött válasz egy állapotjelző kódot tartalmaz, amely az alábbiak egyike lehet

Status Code	Meaning	
200	OK: the request was successful	Sikeres lekérés
303	See Other: used to redirect to a different URI, after a POST request, for instance	Átirányítás
400	Bad Request: the server couldn't understand the request, because the syntax was incorrect	
401	Unauthorized: user authentication is required	Jogosulatlan kérés, azonosítás kell
403	Forbidden: the server refuses to execute the request, authorization won't help	Letiltva
404	Not Found: the requested URI was not found	A kért erőforrás nem található
500	Internal Server Error: The server encountered an unexpected condition and couldn't fulfill the request	

ESP8266 webszerver

- Három lehetőség közül választhatunk, amelyek különböző megközelítést kívánnak és különböző előnyöket kínálnak:
  - A **ESP8266WiFi** programkönyvtár és a **WiFiServer** osztály használata
    Előny: maximális szabadság az üzenetek és a fejlécek kezelésében
    Hátrány: a kommunikáció a klienssel és az adatok értelmezése ránk marad
    Példa: **ESP8266_webserver_complex**
  - Az **ESP8266WebServer** programkönyvtár használata
    Előny: kényelmes használat
    Hátrány: kötött fejléc generálás, egyidejűleg csak egy kliens kiszolgálása
    Példa: **ESP8266_webserver_simple**
  - Az ESPAsyncWebServer programkönyvtár (ESPAsyncTCP is kell hozzá)
    Előny: kényelmes használat, gyors, több kliens, gazdag szolgáltatáskínálat
    Hátrány: külön kell telepíteni a könyvtárakat (verzió inkompatibilitás?)
    Példa: **ESP8266_async_webserver**

WiFiserver – a rögösebb út

ESP8266_webserver_complex.ino - az alábbi minta alapján
arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/server-examples.html

```
#include <ESP8266WiFi.h>
#include "secrets.h"
WiFiServer server(80);

void setup(void){
  setup_wifi();                           // Csatlakozás a helyi hálózatra
  server.begin();                         // Indítjuk a webszervert
}

void loop() {
  WiFiClient client = server.available();
  if (client) {                           // wait for a client to connect
    while (client.connected()) {
      if (client.available()) {
        String line = client.readStringUntil('\r');  // read request line by line
        if (line.length() == 1 && line[0] == '\n') { // wait for end of client's request
          client.println(prepareHtmlPage());
          break;
        }
      }
    }
    while (client.available()) {
      client.read();                      // let client finish its request
    }
    client.stop();                        // close the connection:
  }
}
```

Itt küldjük a kliensnek a választ

ESP8266_webserver_complex.ino

```
String prepareHtmlPage() {
  String htmlPage;
  htmlPage.reserve(1024);  // prevent ram fragmentation
  htmlPage = F("HTTP/1.1 200 OK\r\n"
               "Content-Type: text/html\r\n"
               "Connection: close\r\n"
               "Refresh: 5\r\n"  // Refresh every 5s
               "\r\n"
               "<!DOCTYPE HTML>"
               "<html>"
               "Analog input:  ");
  htmlPage += analogRead(A0);
  htmlPage += F("</html>"
                "\r\n");
  return htmlPage;
}
```

Ez a rész ne a RAM-ban fogja a helyet!

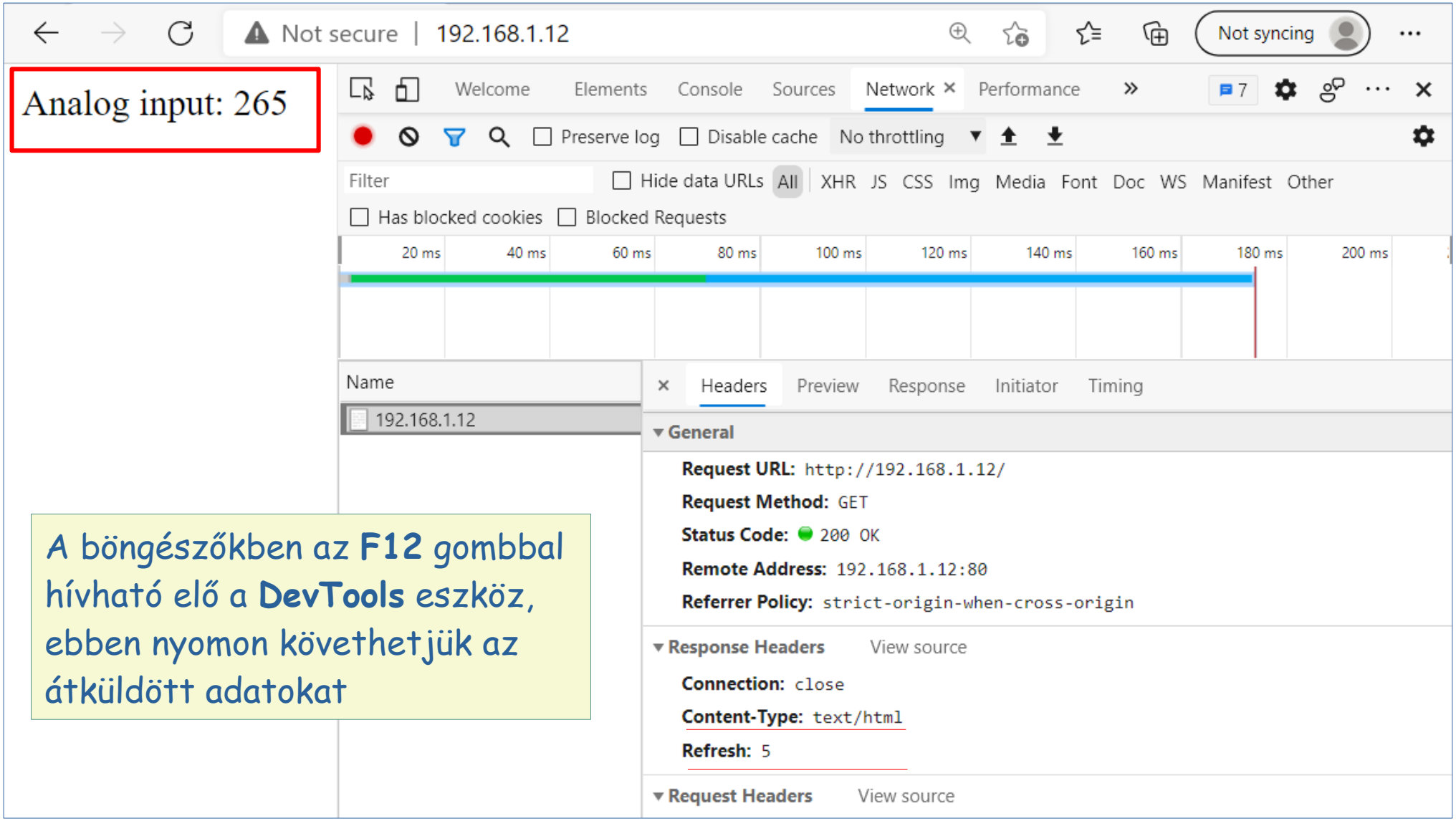
Üzenet fejrésze

Üzenet törzse

- A weblap összeállítását a fenti függvény végzi
- Az ADC kiolvasásával minden lekéréskor aktualizáljuk a lapot
- A kliens kapcsolatot a kiszolgálás végén lezárjuk, s a kliens 5 s elteltével új lekéréssel frissíti a böngészőben a lapot

ESP8266_webserver_complex.ino

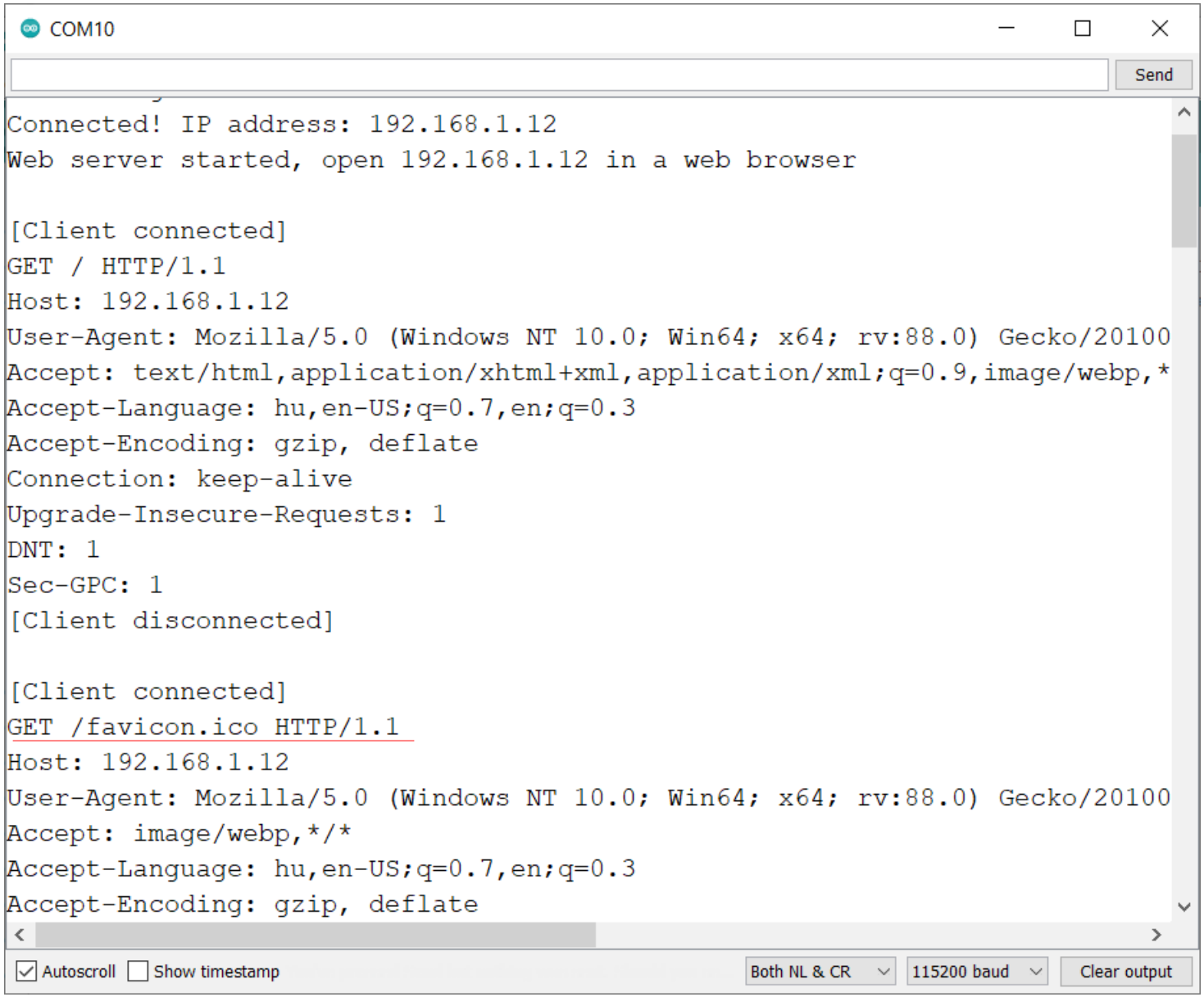
- Az alábbi ábrán a kijelzett eredmény és az üzenetváltás látható



A böngészőkben az **F12** gombbal hívható elő a **DevTools** eszköz, ebben nyomon követhetjük az átküldött adatokat

favico.ico lekérés – egy kis probléma

- A naplózás jól mutatja, hogy a HTML lekérés után a böngésző a (nemlétező) **favico.ico** ikont is lekéri automatikusan
- A következő oldalon megmutatjuk, hogyan lehet ezt megakadályozni



favico.ico lekérés – probléma megoldva

```
String prepareHtmlPage() {
  String htmlPage;
  htmlPage.reserve(1024);  // prevent ram fragmentation
  htmlPage = F("HTTP/1.1 200 OK\r\n"
               "Content-Type: text/html\r\n"
               "Connection: close\r\n"
               "Refresh: 5\r\n"  // Refresh every 5s
               "\r\n"
               "<!DOCTYPE HTML>"
               "<html><head><title>ESP8266</title>”
               "<link rel=\"icon\" href=\"data:,\"></head><body>"
               "Analog input:  ");
  htmlPage += analogRead(A0);
  htmlPage += F("</body></html>"
                "\r\n");
  return htmlPage;
}
```

- Bővítsük a HTML lap fejrészét az alábbi sorral:

  `<link rel="icon" href="data:,">`

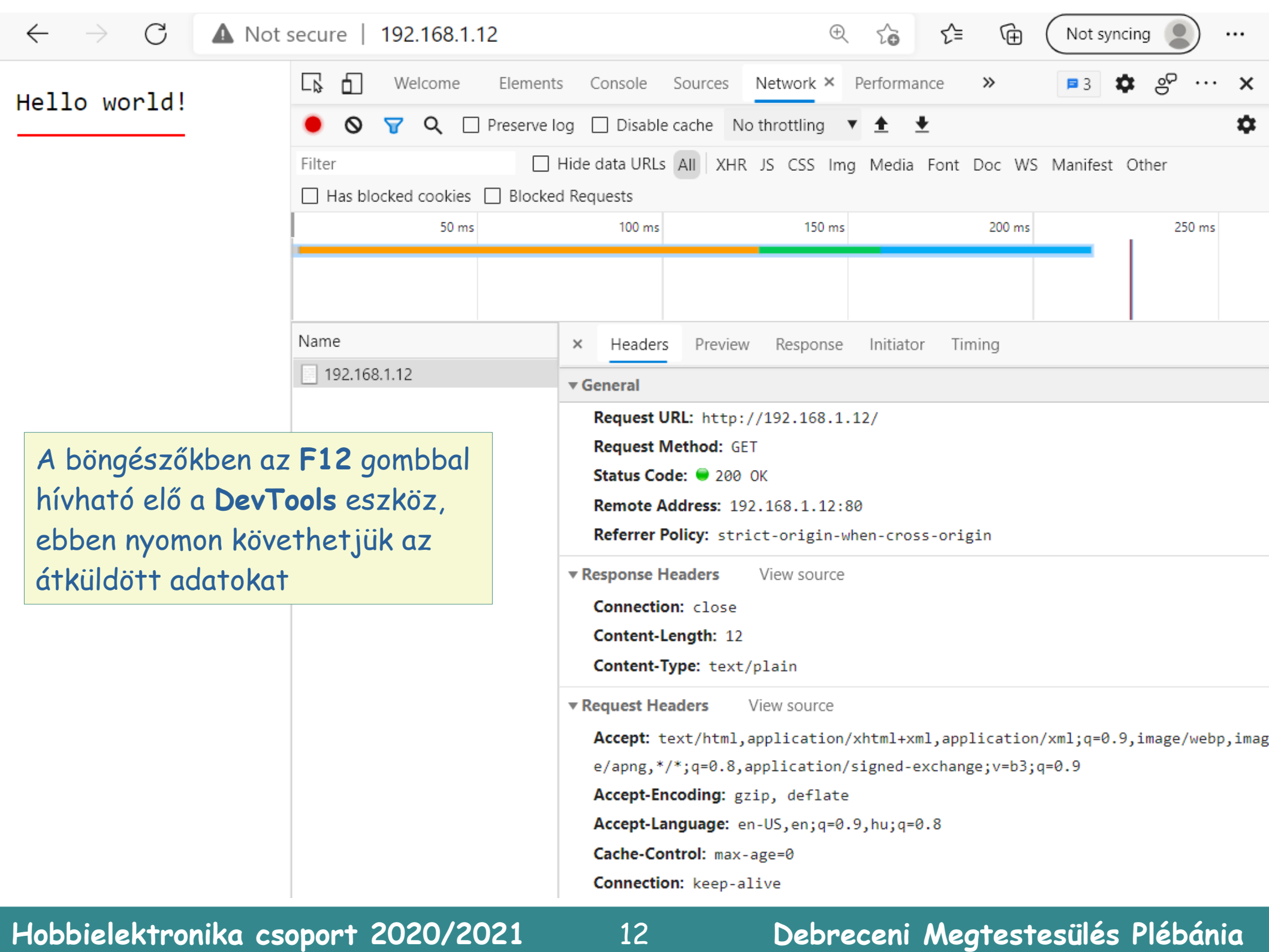
  bővebben lásd itt: How to Prevent Automatic Favicon Requests

Az ESP8266WebServer programkönyvtár

- Az **ESP8266 Arduino Core** részeként az **ESP8266WebServer** könyvtárat is megkaptuk, ezzel kényelmesebb dolgozni

ESP8266_webserver_simple.ino - a kényelmes út

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include "secrets.h"
ESP8266WebServer server(80);                    // Webszerver objektumpéldány létrehozása

void setup(void){
  setup_wifi();                                 // Csatlakozás a helyi hálózatra
  server.on("/", handleRoot);                   // Visszahívási függvény hozzárendelés
  server.onNotFound(handleNotFound);            // Visszahívási függvény hozzárendelés
  server.begin();                               // Indítjuk a webszervert
}

void loop(void){
  server.handleClient();                        // A kliens kapcsolatok kezelése
}

void handleRoot() {                             // A "/" URI lekérés kiszolgálása
  server.send(200, "text/plain", "Hello world!");
}

void handleNotFound(){                          // A sikertelen lekérések kiszolgálása
  server.send(404, "text/plain", "404: Not found");
}
```



A böngészőkben az **F12** gombbal hívható elő a **DevTools** eszköz, ebben nyomon követhetjük az átküldött adatokat

ESP8266_async_webserver.ino

- Néhány eltéréstől eltekintve az **ESPAsyncWebserver** programkönyvtárnak is hasonló a használata, mint az **ESP8266WebServer** könyvtáré

```
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include "secrets.h"

AsyncWebServer server(80);                  // Webszerver objektumpéldány létrehozása

void setup(void) {
  setup_wifi();                             // Csatlakozás a helyi hálózatra
  server.on("/", HTTP_GET, handleRoot);     // Visszahívási függvény hozzárendelés
  server.onNotFound(handleNotFound);        // Visszahívási függvény hozzárendelés
  server.begin();                           // Indítjuk a webszervert
}

void loop(void) { }

void handleRoot(AsyncWebServerRequest *request) {
  request->send(200, "text/plain", "Hello world!");
}

void handleNotFound(AsyncWebServerRequest *request) { // sikertelen lekérések
  request->send(404, "text/plain", "404: Not found");
}
```

A módszer megadása opcionális

Nincs tennivaló!

A korábbiakhoz képest eltérés, hogy most a visszahívási függvényekben **request** objektumokat kezelünk

HTML alapok

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>
<h1>This is a Heading</h1>
<p>This is a paragraph.</p>
<a href="https://www.w3schools.com">This is a
link</a>

<h2>This is a paragraph with image</h2>
<p><img src="w3schools.jpg"
alt="W3Schools.com" width="104" height="142"
align="right">Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Nam pretium orci
non ultricies faucibus. Donec a quam placerat,
tempor ex in, maximus ligula. Sed vitae sapien
in quam pulvinar accumsan. Phasellus quis
rutrum mi. Nunc pretium nisi at risus
pellentesque, sit amet luctus libero
scelerisque. Nam accumsan euismod dictum. Ut
ac orci dignissim, commodo lorem sed,
scelerisque dolor. </p>
</body>
</html>
```

<head> és <meta> elemek

- The HTML **<head>** elem egy konténer a következő elemek számára: **<title>, <style>, <meta>, <link>, <script>** és **<base>**
- A **<meta>** elemek kísérő információkat adnak át a böngészőnek
- A **<link>** elemek külső erőforrások viszonyát/helyét adja meg a dokumentumhoz képest (stíluslapok, favico.ico). Globális Attribútumok megadását is támogatja

```
<!DOCTYPE html>
<html lang="hu">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="demopage">
  <title>Hobbielektronika csoport</title>
  <link rel="stylesheet" href="css/main.css">
  <link rel="icon" href="data:,">
</head>
<body>

</body>
</html>
```

Karakterkódolás

Reszponzivitás

Stíluslap betöltés

favico.ico lekérés tiltás

Favicon

- A **favicon** (*favorite icon* vagy *shortcut icon*) egy, a webszerverre feltöltött fájl, ami egy vagy több kis képet (ikont) tartalmaz, ami a webhelyhez vagy weboldalhoz rendelhető. Az ikon a böngésző címsorában, az előzményekben, vagy a könyvjelzőknél jelenik meg
- Például egy weblaphoz így rendelhető hozzá (a <head> szekcióban):
  `<link rel="icon" type="image/png" href="image/favicon.png">`
- Az egyes böngészők az alábbi favicon formátumokat támogatják:

Böngésző	Kép formátum						
	ICO	PNG	GIF	Animált GIF	JPEG	APNG	SVG
Edge	igen	igen	igen	nem	ismeretlen	ismeretlen	ismeretlen
Firefox	igen	igen	igen	igen	igen	3.0	41.0
Chrome	igen	igen	4.0	nem	igen	nem	80
Int. Explorer	5.0	11.0	11.0	nem	nem	nem	nem
Opera	7.0	7.0	7.0	7.0	7.0	9.5	44.0
Safari	igen	4.0	4.0	nem	4.0	nem	nem szabv.

HTML stílusok

1. A stílust attribútumként használva egy elem megjelenését befolyásolja, formája:

```
<tagname style="property:value;">
```

```
<h2 style="color:blue;">This is a heading</h2>
<p style="color:red;">This is a paragraph.</p>
```

This is a heading

This is a paragraph.

```
<h1 style="font-family:Arial;">This is a heading</h1>

<p style="font-family:courier;">This is a paragraph.</p>

<p style="font-size:160%;">This is a paragraph.</p>

<p style="text-align:center;">Centered paragraph.</p>
```

This is a heading

This is a paragraph.

This is a paragraph.

Centered paragraph.

2. A stílust `<style>` elemként is megadhatjuk a `<head>` szekcióban

```
<!DOCTYPE html><html> <head>
<style>
  h2 {color:red;}
  p {color:blue;}
</style>
</head><body>
<h1>Big Title</h1>
<h2>This is a level 2 heading</h2>
<p>This is a paragraph.</p>
<h2>Another level 2 heading</h2>
<p>This is another paragraph</p>
</body></html>
```

Big Title

This is a level 2 heading

This is a paragraph.

Another level 2 heading

This is another paragraph

HTML stílusok

3. A HTML stílust külső fájlban is megadhatjuk, ekkor a **<head>** szekcióban egy **<link>** elem segítségével kell becsatolni: **<link rel="stylesheet" href="styles.css">**

- A HTML stílus leíró **CSS** (Cascaded Style Sheets) nyelv erősségét az ún. szelektorok adják, ezek segítségével dönti el a böngésző, hogy egy adott elemere melyik stílusdefiníció vonatkozik (vagy melyek vonatkoznak)

```
#para1 { text-align: center; color: red; }

.center { text-align: center; color: red; }

p.center { text-align: center; color: red; }
```

```
<p id="para1">Bekezdés</p>

Minden class=center elemre

<p class="center">Bekezdés</p>
```

```
<p class="center large">Bekezdés</p>   A center és a large osztály is vonatkozik rá
```

- Csoportosítás: az alábbi három stílusdefiníció összevonható

```
h1 { text-align: center; color: red; }

h2 { text-align: center; color: red; }

p { text-align: center; color: red; }
```

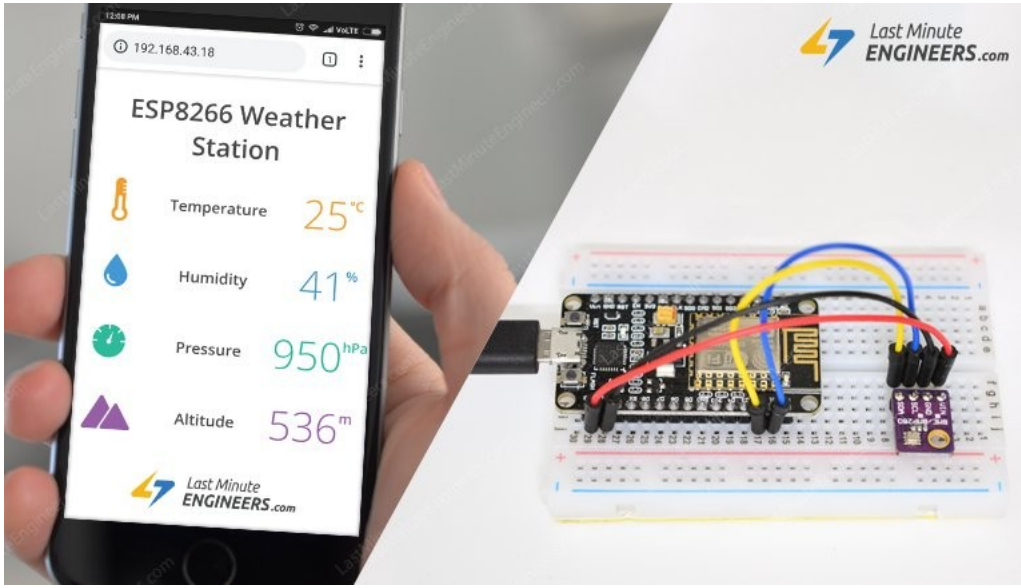
```
h1, h2, p {
    text-align: center;
    color: red;
}
```

ESP8266_BME280_01

- A Last Minute Engineers ESP8266+BME280 időjárás állomásával már találkoztunk korábban, most nézzük meg közelebbről is!

```
#include <ESP8266WebServer.h>
#include "secrets.h"
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme;
float temperature, humidity, pressure, altitude;

ESP8266WebServer server(80);

void setup() {
  setup_wifi();
  bme.begin(0x76);
  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);
  server.begin();
}

void loop() {
  server.handleClient();
}
```

Megjegyzés: Ez itt a fenti cikk kiindulási programja, ami (még) nincs úgy kicicomázva, mint a képen látható "végtermék"



ESP8266_BME280_01

```
void handle_OnConnect() {
  temperature = bme.readTemperature();
  humidity = bme.readHumidity();
  pressure = (bme.readPressure() + 1440) / 100.0F;
  altitude = bme.readAltitude(SEALEVELPRESSURE_HPA);
  server.send(200, "text/html", SendHTML(temperature,humidity,pressure,altitude));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

void setup_wifi() {
  Serial.begin(115200);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected! IP address: ");
  Serial.println(WiFi.localIP());
}
```

ESP8266_BME280_01

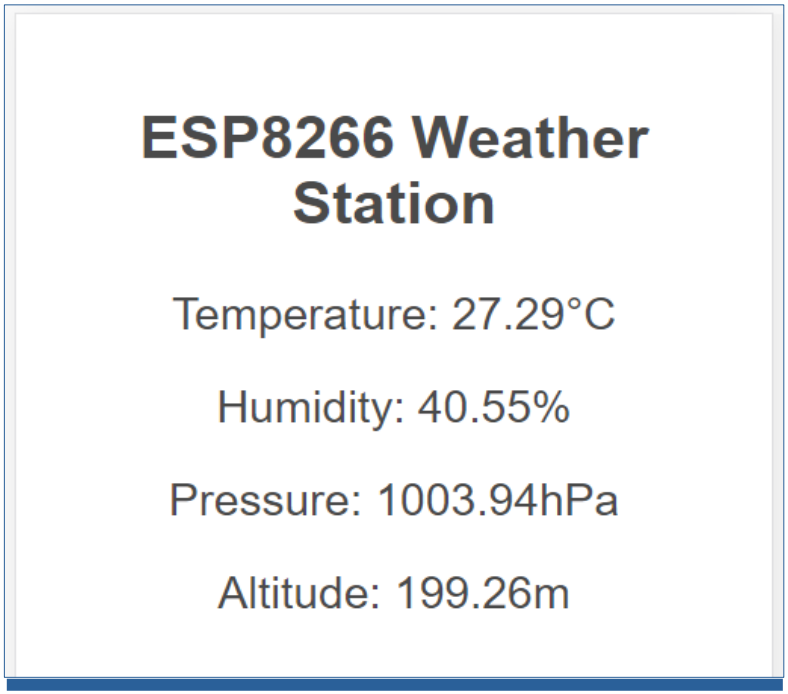
```
String SendHTML(float temperature,float humidity,float pressure,float altitude){
  String ptr = "<!DOCTYPE html> <html><head>\n";
  ptr +="<meta name=\"viewport\" content=\"width=device-width,initial-scale=1.0,\">\n";
  ptr +="<title>ESP8266 Weather Station</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto;
text-align: center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;}\n";
  ptr +="p {font-size: 24px;color: #444444;margin-bottom: 10px;}\n";
  ptr +="</style></head>\n";
  ptr +="<body><div id=\"webpage\">\n";
  ptr +="<h1>ESP8266 Weather Station</h1>\n";
  ptr +="<p>Temperature: ";
  ptr +=temperature;
  ptr +="&deg;C</p>";
  ptr +="<p>Humidity: ";
  ptr +=humidity;
  ptr +="%</p>";
  ptr +="<p>Pressure: ";
  ptr +=pressure;
  ptr +="hPa</p>";
  ptr +="<p>Altitude: ";
  ptr +=altitude;
  ptr +="m</p>";
  ptr +="</div></body></html>\n";
  return ptr;
}
```

A mérési adatokból „röptében” állítjuk össze ezt a részt

ESP8266_BME280_01 kimenete

```
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>ESP8266 Weather Station</title>
<style>
  html { font-family: Helvetica; display: inline-block;
    margin: 0px auto; text-align: center;
  }
  body {margin-top: 50px;}
  h1 {color: #444444; margin: 50px auto 30px;}
  p {font-size: 24px; color: #444444; margin-bottom: 10px;}
</style>
</head>
<body>
  <div id="webpage">
    <h1>ESP8266 Weather Station</h1>
    <p>Temperature: 27.29&deg;C</p>
    <p>Humidity: 40.55%</p>
    <p>Pressure: 1003.94hPa</p>
    <p>Altitude: 199.26m</p>
  </div>
</body>
</html>
```

Ez a HTML kód, amit megkap a böngésző



ESP8266_BME280_02

- Néhány praktikus változtatást érdemes tennünk: automatikus frissítés, favicon lekérés tiltás, számkiírás formátumának szabályozása, hosszú szöveg flash memóriában tárolása

```
String SendHTML(float temperature, float humidity, float pressure, float altitude) {
  String ptr = F("<!DOCTYPE html> <html>"
                 "<head><meta name=\"viewport\" content=\"width=device-width,
                  initial-scale=1.0\">"
                 "<meta http-equiv=\"refresh\" content=\"5\" >"
                 "<title>ESP8266 Weather Station</title>"
                 "<link rel=\"icon\" href=\"data:,\">"
                 "<style>html { font-family: Helvetica; display: inline-block;"
                 "margin: 0px auto; text-align: center;}"
                 "body{margin-top: 50px;} h1 {color: #730841;margin: 50px auto 30px;}"
                 "p {font-size: 24px;color: #444444;margin-bottom: 10px;}"
                 "</style></head><body>"
                 "<div id=\"webpage\">"
                 "<h1>ESP8266 Weather Station</h1>");
  ptr += "<p>Temperature: " + String(temperature,1) + " &deg;C</p>";
  ptr += "<p>Humidity: " + String(humidity,0) + " %</p>";
  ptr += "<p>Pressure: " + String(pressure,1) + " hPa</p>";
  ptr += "<p>Altitude: " + String(altitude,0) + " m</p>";
  ptr += "</div></body></html>\n";
  return ptr;
}
```

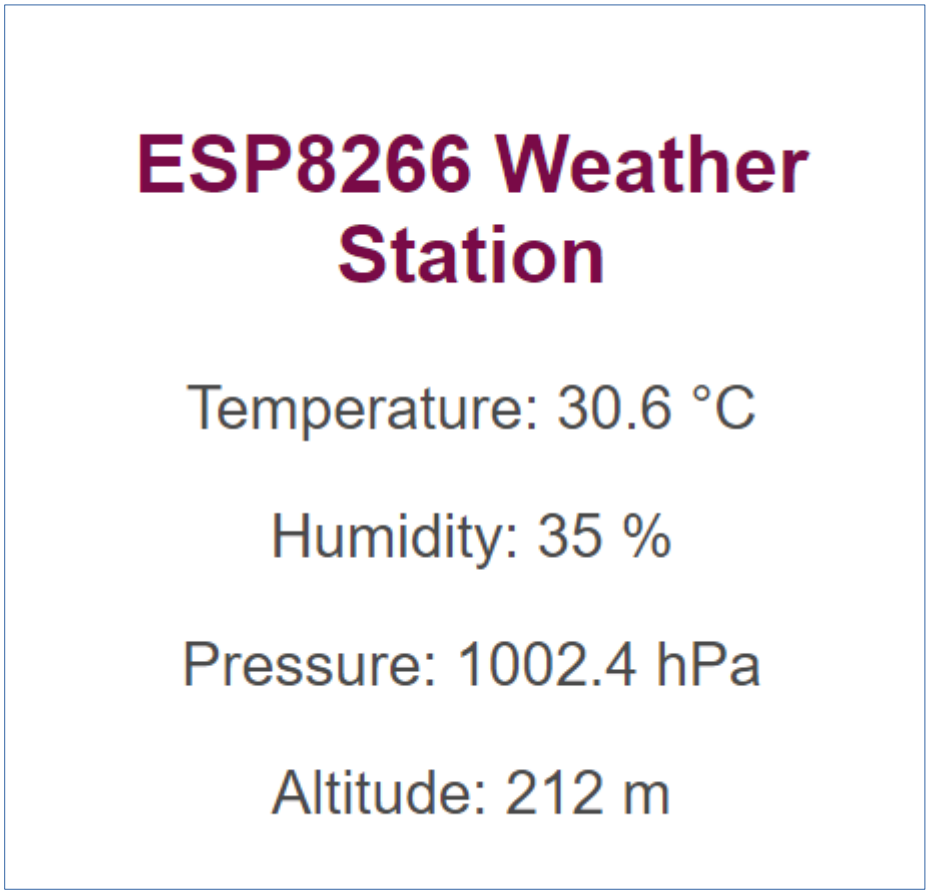
frissítés

favicon lekérés tiltás

Ez a rész ne a RAM-ban fogja a helyet!

ESP8266_BME280_02

- Az előző oldali változtatásokkal így módosult a kinézet
- A kidekorált végleges változat **ESP8288_WS_bmp280** néven megtalálható a **2021. márc. 11**-i előadás mintaprogramjai között, vagy a Last Minute Engineers: **Create A Simple ESP8266 Weather Station With BME280** c. leírásban

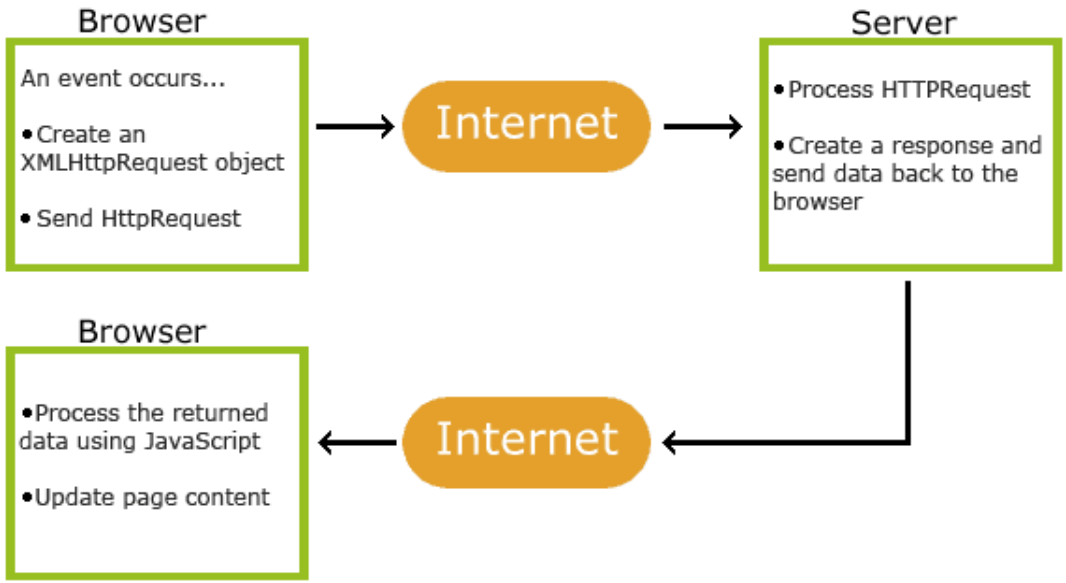


AJAX - Asynchronous JavaScript And XML

- Az **AJAX** nem programnyelv, hanem egy technika, amellyel
  - Frissíthető egy weblap újratöltés nélkül
  - Adatot kérhetünk le a szerverről – a weblap letöltése után
  - Adatot fogadhatunk a szervertől – a weblap letöltése után
  - Adatot küldhetünk a szervernek – a háttérben
- Az **AJAX** technika kombinálja a böngésző beépített **XMLHttpRequest** objektumát (az adatok lekérésére) és a **JavaScriptet** az adatok dinamikus megjelenítésére

- Bővebb információ:

  W3Schools: XML AJAX



ESP8266_BME280_03

- A Last Minute Engineers ESP8266+BME280 időjárás állomás projektjében is bemutatták az AJAX technika használatát, lecserélve az automatikus frissítést előíró sort egy **JavaScript** függvényre:
- A `SendHTML()` függvényben a HTML fejrészéből hagyjuk el ezt:
  `<meta http-equiv="refresh" content="5">`
- Ugyanott (a </head> címke előtt) helyezzük el az az alábbi **JavaScript** kódot:

```
<script>
  setInterval(loadDoc,5000);                    // 5000 ms-onként indíja a loadDoc() fv-t
  function loadDoc() {
    var xhttp = new XMLHttpRequest();                    // Új objektum példányosítás
    xhttp.onreadystatechange = function() {              // Inline visszahívási fv.
      if (this.readyState == 4 && this.status == 200) {
        document.body.innerHTML = this.responseText;     // A weblap frissítése
      }
    };
    xhttp.open("GET", "/", true);                        // Új Request beállítása
    xhttp.send();                                        // A lekérés indítása
  }
</script>
```

Ezzel így nem sokkal jutottunk előbbre, mert így is a teljes lap frissül

AJAX – finomítjuk a technikát

- A `<div id=”demo>` megjelöl egy szakaszt, s a `getElementsById()` metódus segítségével csak a megnevezett szakaszt cseréljük le a lekérésre válaszul kapott tartalomra. Ezt fogjuk alkalmazni…

```
<!DOCTYPE html>
<html>
<body>
<h1>AJAX demo</h2>
<p>Ez a bekezdés nem változik</p>
<div id="demo">
<h2>Ezt fogjuk lecserélni</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>
<script>
function loadDoc() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt", true);
  xhttp.send();
}
</script>
</body>
</html>
```

Előtte

AJAX demo

Ez a bekezdés nem változik

Ezt fogjuk lecserélni

Change Content

Utána

AJAX demo

Ez a bekezdés nem változik

AJAX

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

ESP8266_ledswitch_ajax

- A circuits4you.com portálon található egy ideillő mintapélda: ESP8266 (ajax) update part of web page without refreshing
- Ebben a webszerver nyitólapja statikus szövegként a flash memóriában tárolódik, a `GET / HTTP/1.1` lekérésre ezt küldjük ki
- A nyitólapba beágyazott `getData()` JavaScript függvény két másodpercenként lekéri az **ADC** által mért adatot és a weblap megjelölt helyére beszúrja
- A weblap két nyomógombot is definiál, ezek a beágyazott `sendData()` JavaScript függvényt hívják meg, különböző paraméterrel. Ez a függvény **XmlHttpRequest** segítségével adatot küld a szervernek a beépített LED kapcsolgatására.

index.h

- A weblapunkat egy fejléc állományban helyezzük el

PROGMEM használata ESP8266-vel és Arduino IDE-vel

```
const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<body>
<div id="demo">
<h1>The ESP8266 NodeMCU Update web page without refresh</h1>
  <button type="button" onclick="sendData(1)">LED ON</button>
  <button type="button" onclick="sendData(0)">LED OFF</button><br>
</div>

<div>
  ADC Value is : <span id="ADCValue">0</span><br>
    LED State is : <span id="LEDState">NA</span>
</div>
<script>
function sendData(led) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("LEDState").innerHTML = this.responseText;
    }
  };
  xhttp.open("GET", "setLED?LEDstate="+led, true);
  xhttp.send();
}
```

Index.h

```
setInterval(function() {
  // Call a function repetatively with 2 Second interval
  getData();
}, 2000); //2000mSeconds update rate

function getData() {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("ADCValue").innerHTML =
      this.responseText;
    }
  };
  xhttp.open("GET", "readADC", true);
  xhttp.send();
}

</script>
<br><br><a href="https://circuits4you.com">Circuits4you.com</a>
</body>
</html>
)=====";
```

ESP8266_ledswitch_ajax.ino

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include "secrets.h"
#include "index.h"                          // Our HTML webpage contents with javascripts

#define LED 2                               // On board LED

ESP8266WebServer server(80);                // Server on port 80

void setup(void) {
  setup_wifi();
  pinMode(LED, OUTPUT);                     // Onboard LED port Direction output
  server.on("/", handleRoot);               // This is display page
  server.on("/setLED", handleLED);          // Serve LED requests
  server.on("/readADC", handleADC);         // Serve ADC requests
  server.begin();                           // Start server
  Serial.println("HTTP server started");
}

void loop(void) {
  server.handleClient();                    // Handle client requests
}
```

Amint látjuk, a webszervert működtető program egyszerű, mint a faék...

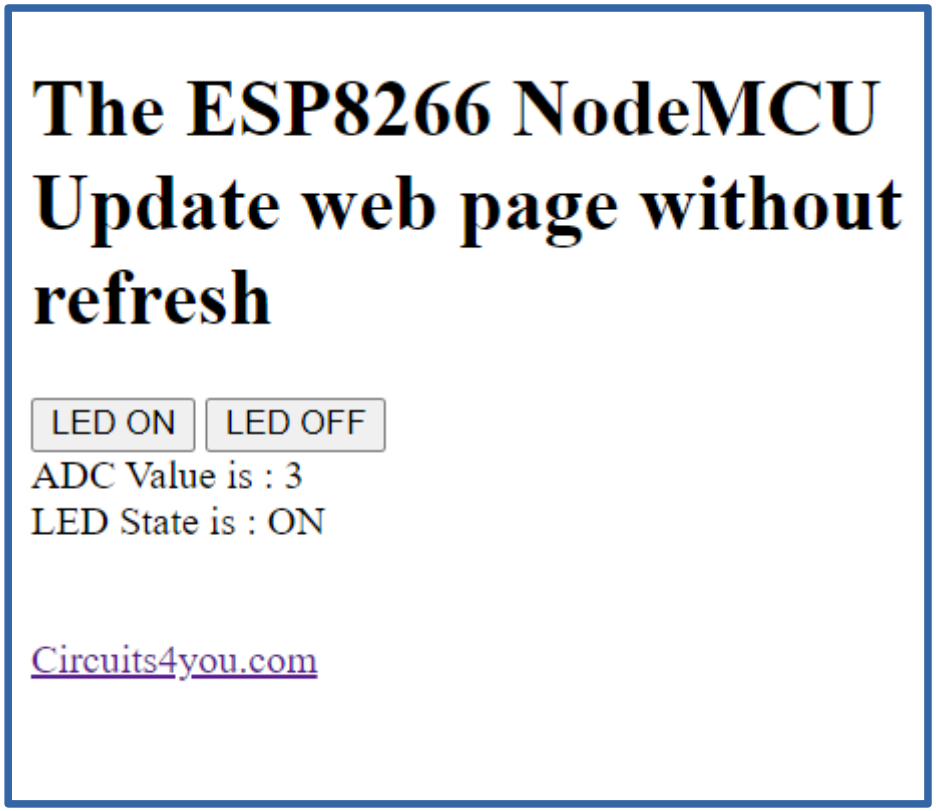
ESP8266_ledswitch_ajax.ino

```
void handleRoot() {
  String s = FPSTR(MAIN_page);                  //Read HTML contents from flash
  server.send(200, "text/html", s);             //Send web page
}

void handleADC() {
  int a = analogRead(A0);
  String adcValue = String(a);
  server.send(200, "text/plane", adcValue); //Send ADC value only to client ajax
request
}

void handleLED() {
  String ledState = "OFF";
  String t_state = server.arg("LEDstate");  // Get arg from "setLED?LEDstate="+led
  Serial.println(t_state);
  if (t_state == "1") {
    digitalWrite(LED, LOW);                 // LED ON (negative logic)
    ledState = "ON";                        // Feedback parameter
  }
  else  {
    digitalWrite(LED, HIGH);                // LED OFF
    ledState = "OFF";                       // Feedback parameter
  }
  server.send(200, "text/plane", ledState); // Send web page
}
```

A kiszolgáló függvények sem bonyolultak...

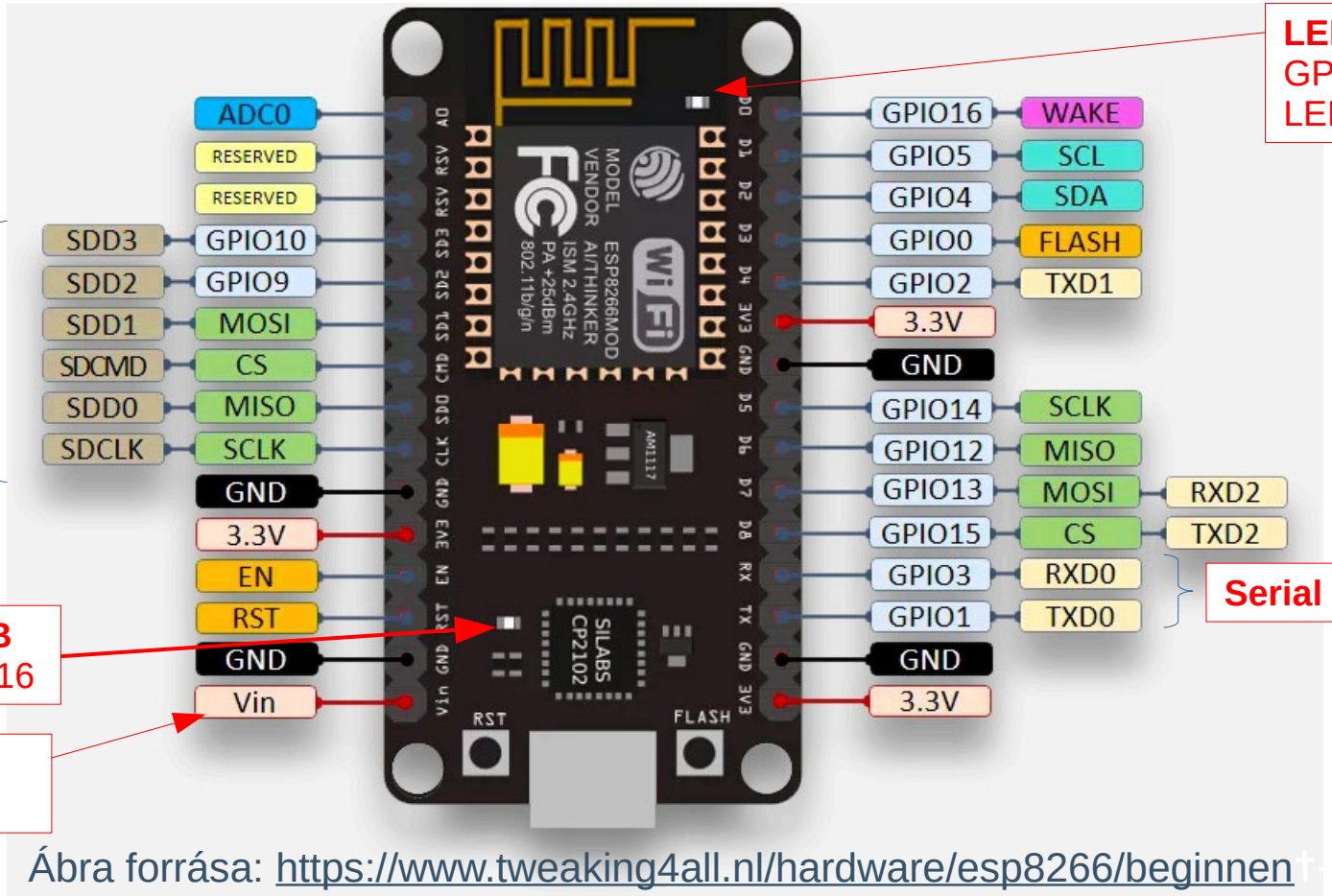
ESP8266_ledswitch_ajax.ino

- Az alábbi ábrán a program futási eredménye látható
- Az ADC lekérdezése automatikusan lefut két másodpercenként
- A LED állapota csak valamelyik nyomógomb lenyomásakor frissül



NodeMCU GPIO kivezetések

- A ki- és bemenetek jelszintje 3,3 V (ADC0 esetén 1 V lenne, de a **NodeMCU** kártyán van egy 200 k + 100 k előosztó)
- FLASH a jobboldali nyomógombhoz csatlakozik
- Az Arduino számozás a **GPIO*n*** jelölésből leválasztott ***n*** szám



Ábra forrása: https://www.tweaking4all.nl/hardware/esp8266/beginnen

Név	GPIO
D0	16
D1	5
D2	4
D3	0
D4	2
D5	14
D6	12
D7	13
D8	15
A0	19