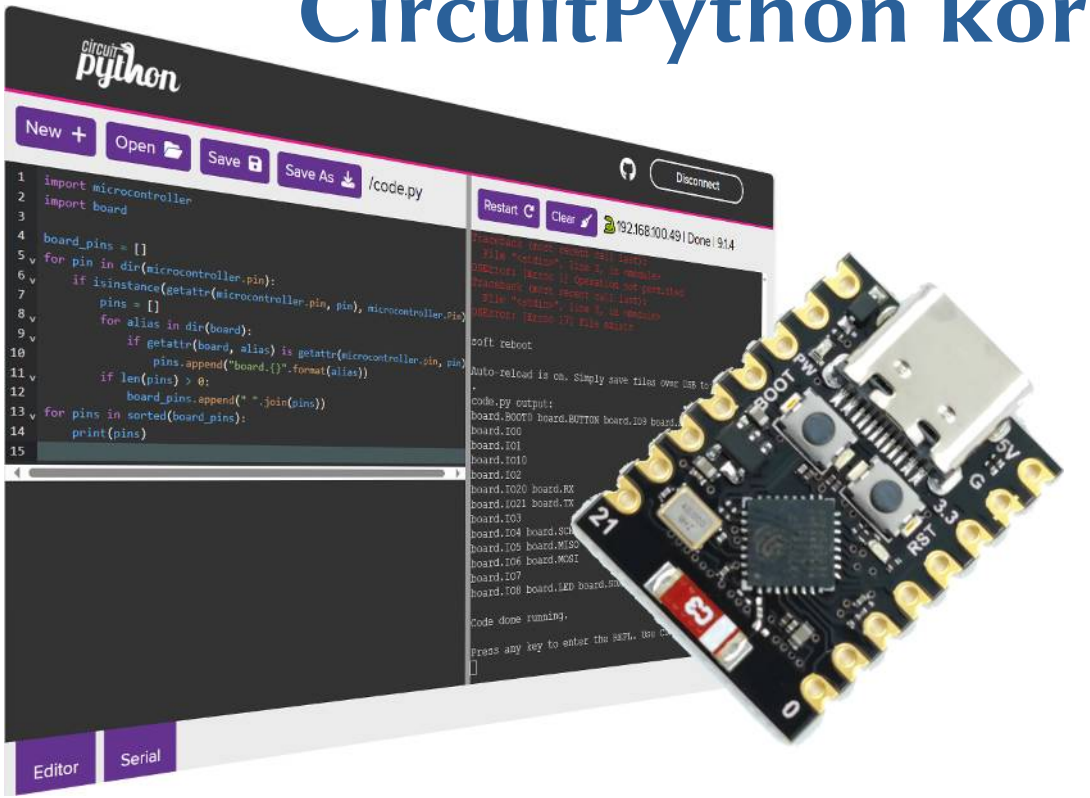
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



10. ESP32 webszerverek CircuitPythonban

Felhasznált és ajánlott irodalom

- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
  - P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- **CircuitPython:**
  - Adafruit: https://circuitpython.org/downloads
  - Learn Adafruit: Welcome to CircuitPython
  - Learn Adafruit: CircuitPython Essentials
  - Adafruit: Adafruit CircuitPython API Reference
  - Adafruit: github.com/adafruit/Adafruit CircuitPython Bundle

- **Online eszközök és támogatás:**
  - Learn Adafruit: CircuitPython on ESP32 Quick Start
  - Adafruit: Adafruit Web Serial ESPTool
  - Adafruit: CircuitPython Code Editor

Mi a 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, illetve a 2025. május 1-jei 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

ESP32-C3 webszerver STA módban

- Az ESP32 webszerver STA (station) módban kapcsolódik a helyi hálózatra
- A helyi hálózat eszközeivel (laptop, tablet, mobil) könnyebben „megtaláljuk”, ha a helyi routeren fix IP címet rendelünk hozzá

Adafruit HTTPServer

- **CircuitPythonban** webszerverhez többféle támogatói könyvtár is található, amelyek különböző megközelítést kívánnak és különböző előnyöket kínálnak:
  pl. Biplane, Ampule, CircuitPython-native-wsgiserver, Adafruit_HTTPServer
- Ebben az előadásban az **Adafruit HTTPServer** programkönyvtárat fogjuk használni, ami az Adafruit_CircuitPython_Bundle csomagból telepíthető, dokumentációja alapján a legegyszerűbb használatát röviden így összegezhetjük:
- Importálás:

```
from adafruit_httpserver import Server, Request, Response, GET, POST, stb.
```

- Inicializálás:

```
server = Server(pool) # Ahol pool egy Socketpool objektum
```

- Kiszolgálás:

```
@server.route("/")    # GET lekérés az alapértelmezett
def base(request: Request):
    return Response(request, "Hello world!")
```

- Indítás és kezelés:

```
server.serve_forever(str(wifi.radio.ipv4_address), 80)
```

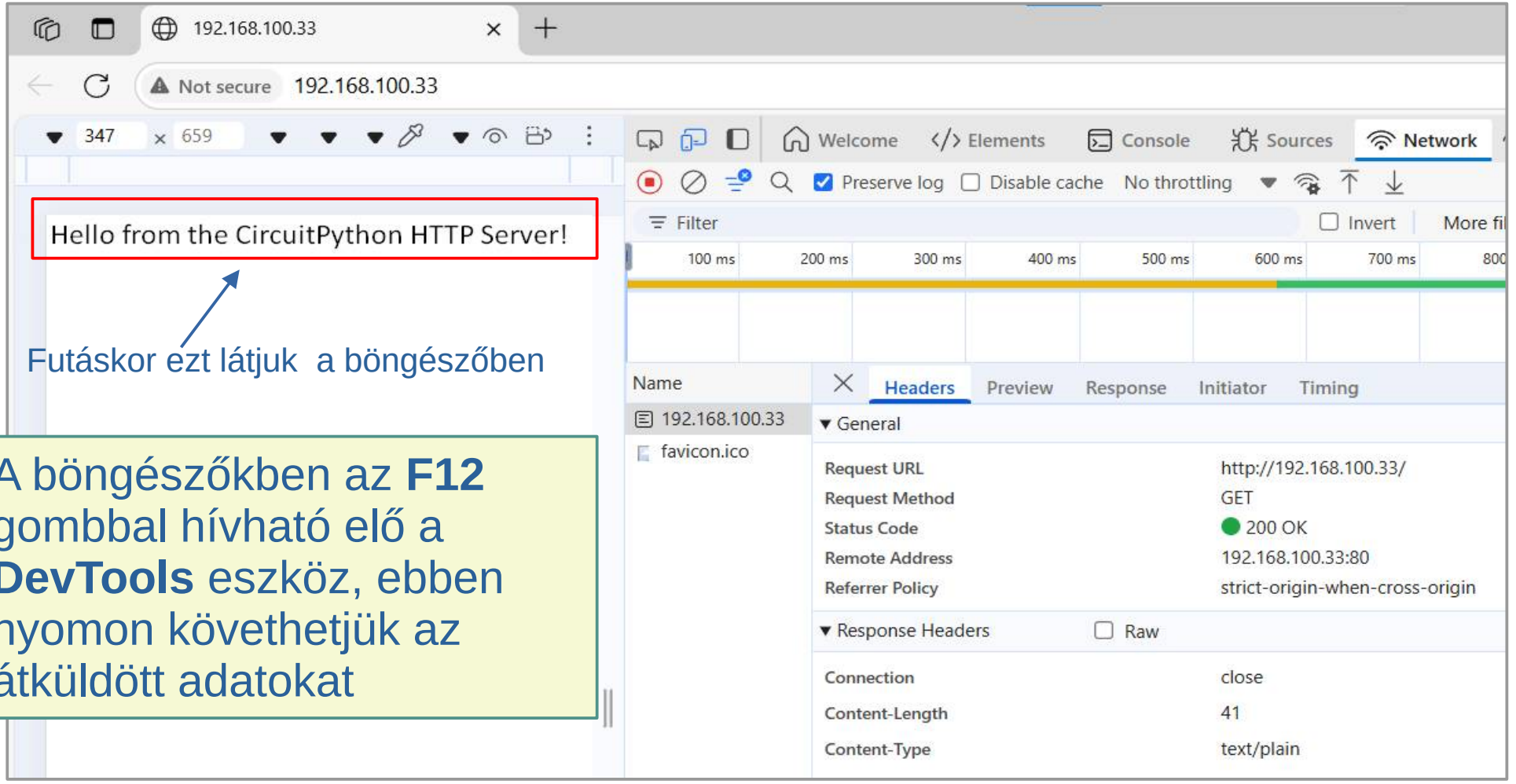
httpserver_helloworld.py

- A legegyszerűbb webszerver, amely csak egy üzenetet ír ki

```
import wifi
from os import getenv
import socketpool
from adafruit_httpserver import Server, Request, Response

WIFI_SSID = getenv("CIRCUITPY_WIFI_SSID")
WIFI_PASS = getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(WIFI_SSID, WIFI_PASS)
pool = socketpool.SocketPool(wifi.radio)
server = Server(pool,debug=True)

@server.route("/")
def base(request: Request):
    return Response(request, "Hello from the CircuitPython HTTP Server!")

@server.route("/favicon.ico")
def favicon(request: Request):
    return Response(request, "", content_type="image/x-icon")

server.serve_forever(str(wifi.radio.ipv4_address), 80)
```

A **@server.route()** dekorátorok egy-egy request URI kiszolgálójául rendelik a mögöttük álló függvényt

httpserver_helloworld.py futási eredménye



Futáskor ezt látjuk a böngészőben

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

favico.ico lekérés – egy kis probléma

```
Shell
>>> %Run -c $EDITOR_CONTENT

  Started development server on http://192.168.100.33:80
192.168.100.3 -- "GET /" 452 -- "200 OK" 125 -- 27ms
192.168.100.3 -- "GET /favicon.ico" 395 -- "200 OK" 85 -- 36ms
192.168.100.3 -- "GET /" 498 -- "200 OK" 125 -- 30ms
192.168.100.3 -- "GET /favicon.ico" 415 -- "200 OK" 85 -- 320ms
```

- A naplózás jól mutatja, hogy a HTML lekérés után a böngésző a (nemlétező) **favico.ico** ikont is lekéri automatikusan (ezt mi egy „üres ikonnal” szolgáltuk ki)
- HTML lapoknál ezt úgy tudjuk elkerülni, hogy a fejrészt bővítjük az alábbi sorral:

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

bővebben lásd itt: How to Prevent Automatic Favicon Requests

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>
```

Egy tipikus HTML lap szerkezete

<head> és <meta> elemek

- A HTML **<head>** elem egy konténer a következő elemek számára: **<title>, <style>, <meta>, <link>** és **<script>**
- 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

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 a **`<head>`** szekcióban is megadhatjuk

```
<!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

```
#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
```

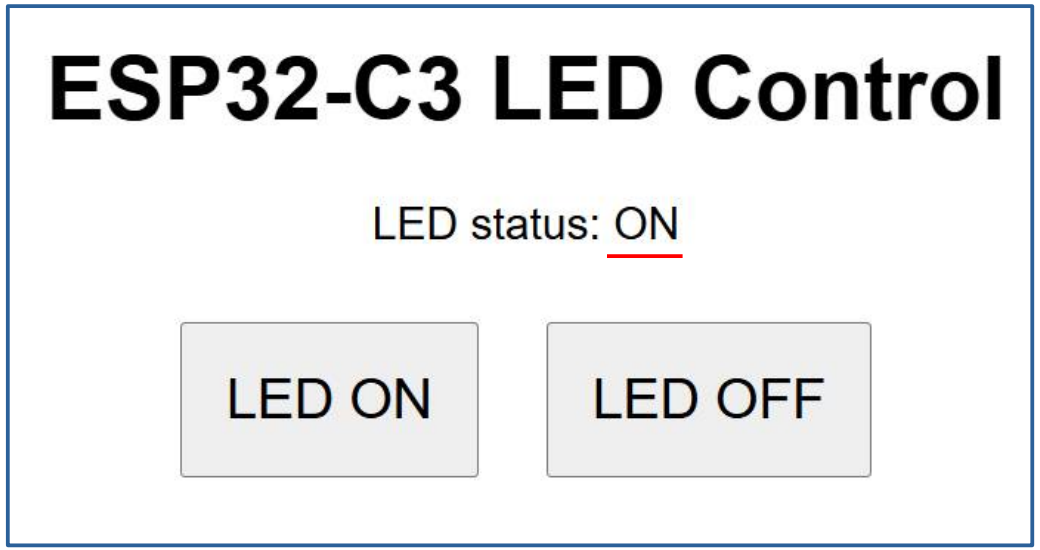
- Csoportosítás: az alábbi három stílusdefiníció összevonható

```
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;
}
```

LED vezérlés webszerverrel

- Ez a program egy egyszerű webszervert valósít meg ESP32-C3-on, amely lehetővé teszi a beépített LED (GPIO8) vezérlését egy böngészőből HTTP POST kérések segítségével
- A nyomógombok egy HTML Formalap elemei, a LED státusz visszajelzését pedig úgy oldjuk meg, ahogy a turistacsalogató arc-kivágásos táblák (face-in-hole board): van egy fenntartott hely, ahová küldéskor mindig beillesztjük a LED aktuális állapotát mutató szöveget.



httpserver_ledcontrol.py – 3/1.

```
import wifi
from os import getenv
import socketpool
from adafruit_httpserver import Server, Request, Response, POST
from digitalio import DigitalInOut, Direction
import board

# WiFi csatlakozás
WIFI_SSID = getenv("CIRCUITPY_WIFI_SSID")
WIFI_PASS = getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(WIFI_SSID, WIFI_PASS)

# Socketpool inicializálása
pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, debug=True)

# LED konfigurálása (katódvezérelt)
led = DigitalInOut(board.LED)  # GPIO8
led.direction = Direction.OUTPUT
led.value = True  # Alapállapot: LED kikapcsolva (katódvezérelt)
```

httpserver_ledcontrol.py – 3/2.

```
# Webszerver útvonalak
@server.route("/")
def base(request: Request):
    return Response(request, webpage(led.value), content_type="text/html")

@server.route("/", POST)
def buttonpress(request: Request):
    raw_text = request.raw_request.decode("utf-8")
    if "ON" in raw_text:
        led.value = False  # Katódvezérelt, így False = BE
    elif "OFF" in raw_text:
        led.value = True   # True = KI
    return Response(request, webpage(led.value), content_type="text/html")

@server.route("/favicon.ico")
def favicon(request: Request):
    return Response(request, "", content_type="image/x-icon")

server.serve_forever(str(wifi.radio.ipv4_address), 80)
```

httpserver_ledcontrol.py – 3/3.

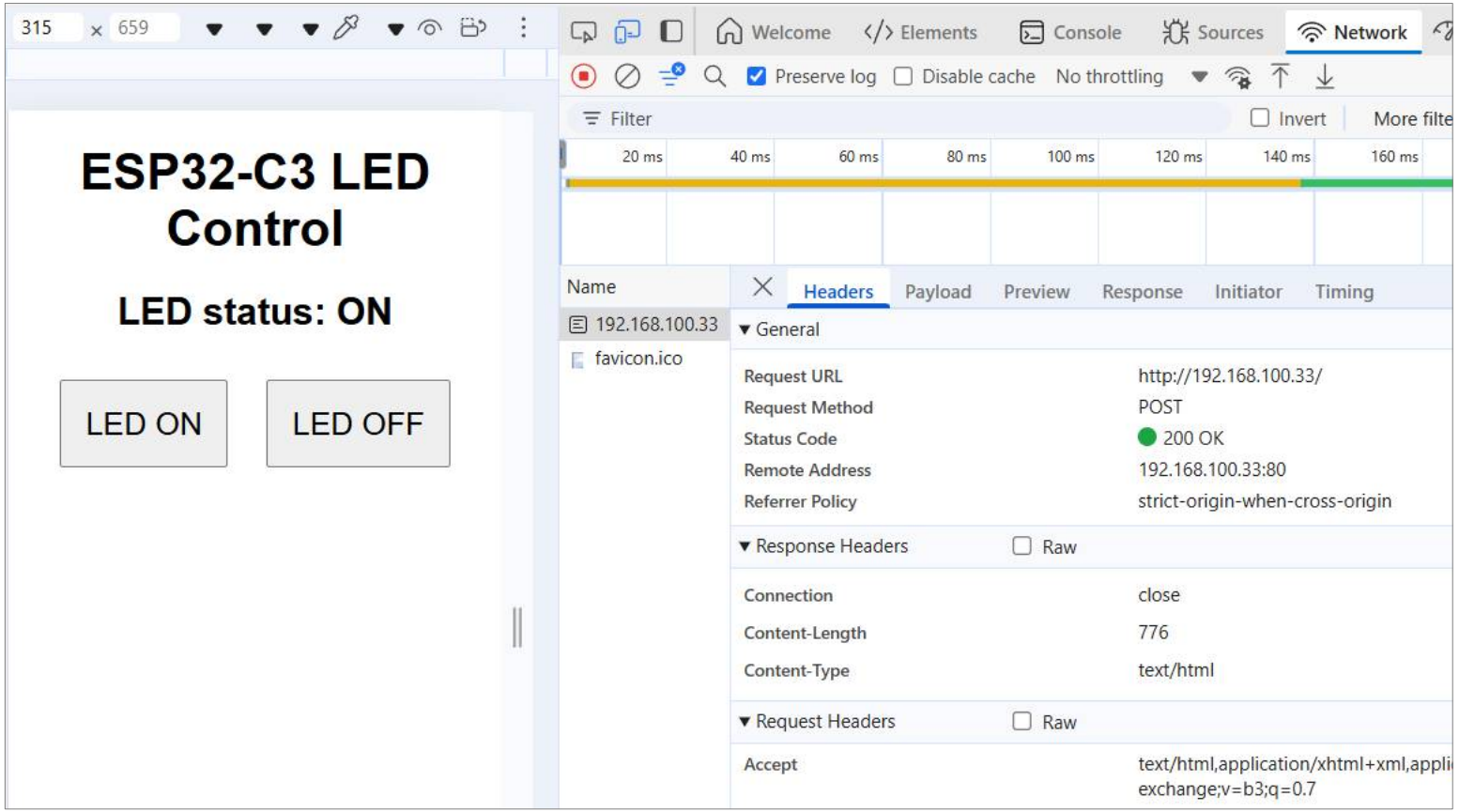
```
def webpage(led_state):
    return f"""
    <!DOCTYPE html>
    <html lang="hu">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
        <title>ESP32-C3 LED Control</title>
        <style>
            body {{ font-family: Arial; text-align: center; }}
            .button {{ padding: 15px; font-size: 20px; margin: 10px; }}
        </style>
    </head>
    <body>
        <h1>ESP32-C3 LED Control</h1>
        <h2>LED status: {"ON" if not led_state else "OFF"}</h2>
        <form method="POST">
            <button class="button" name="LED" value="ON" type="submit">LED ON</button>
            <button class="button" name="LED" value="OFF" type="submit">LED OFF</button>
        </form>
    </body>
    </html>
    """
```

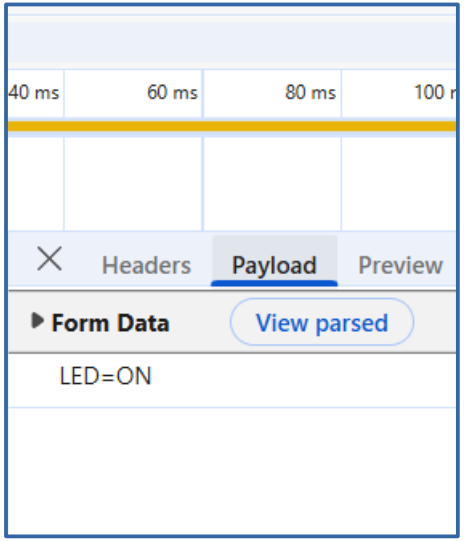
Az f”””…..””” típusú szövegben a { és } karaktereket meg kell duplázni!

led_state értékétől függően itt "ON" vagy "OFF" lesz kiküldve

httpserver_ledcontrol.py futási eredménye

- A böngészőben látható, hogy **POST** kérés ment ki, melynek törzse: **LED=ON**



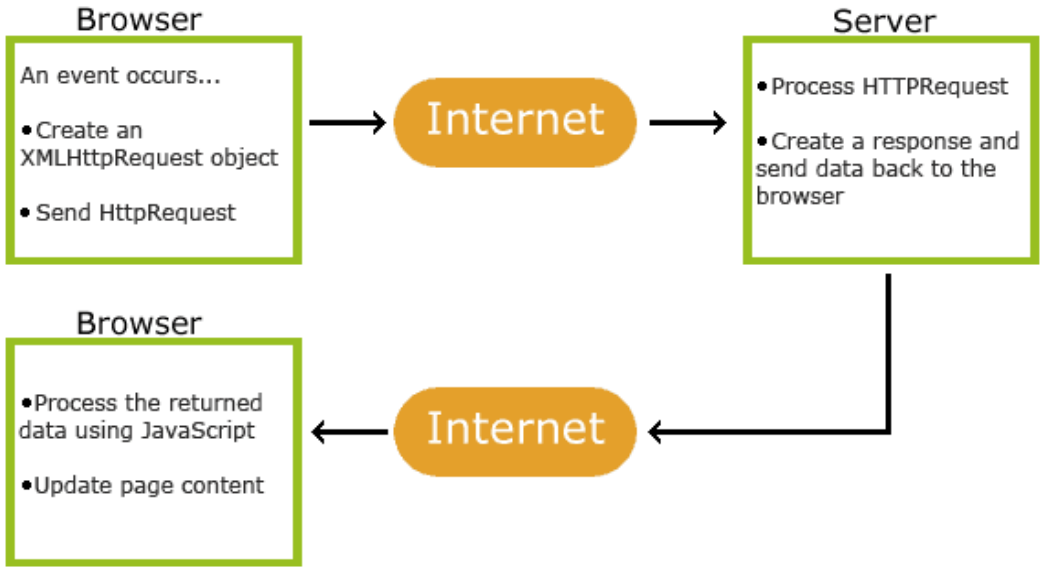


Hogyan frissítsük a weblapot?

- Az előző LED vezérlő programnál a böngészőben megjelenített tartalom csak akkor változott, amikor a felhasználó megnyomta a gombot. Ilyenkor a szerver minden **POST kérésre** válaszul kiküldte a **frissített HTML lapot**
- Amikor azonban külső forrásból származó adatot jelenítünk meg (pl. ADC mérések, szenzoradatok), akkor az érzékelő folyamatosan új értékeket mér, ezért automatikusan kell frissítenünk a böngésző ablakban megjelenített tartalmat.
- **Az automatikus frissítés lehetőségei**
  - **Meta Refresh:** Ha a weblap fejrészében szerepel a `<meta http-equiv="refresh" content="5">` sor, akkor a böngésző ötmásodpercenként automatikusan újra letölti a weblapot
  - **JavaScript** időzítő (ezt is a weblap fejrészében kell elhelyezni, s ez is újratölti a lapot) `<script>setInterval(() => location.reload(), 5000);</script>`
  - **AJAX technika** – lehetővé teszi, hogy csak a szükséges adatokat kérjük le. A HTML nem frissül teljesen, csak az a rész, ahol az új ADC értéket vagy LED állapotot meg kell jeleníteni. Ez gyorsabb és kevesebb adatforgalmat generál, így hatékonyabb

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



Hogyan működik az AJAX?

- Egy JavaScript függvény küld egy HTTP kérést a szerver felé
- A szerver feldolgozza a kérést, majd választ küld vissza (JSON vagy egyszerű szöveg)
- JavaScript fv. a kapott adatokat alapján csak az érintett HTML elemek tartalmát frissíti
- Például az alábbi kód 2 másodpercenként lekéri a **`/readADC`** URI-t és a kapott választ beszúrja a honlapon a **`<span id="ADCValue">0</span>`**-vel megjelölt helyre

```
<script>
setInterval(function() { getData(); }, 2000); // 2 s-onként hívja a getData() fv-t
function getData() {
    var xhttp = new XMLHttpRequest();          // Új objektum példányosítás
    xhttp.onreadystatechange = function() {    // Inline visszahívási függvény
        if (this.readyState == 4 && this.status == 200) {
            document.getElementById("ADCValue").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "/readADC", true);       // Új Request beállítása
    xhttp.send();                              // A lekérés indítása
}
</script>
```

Aszinkron módú legyen a lekérés végrehajtása

LED vezérlés és ADC kiolvasás AJAX technikával

- A következő program egy **ESP32-C3** alapú webszervert valósít meg, ahol a böngészőből vezérelhetjük a LED állapotát, és lekérhetjük egy ADC értékét
- A programban bemutatjuk az **AJAX** technika alkalmazását, ami lehetővé teszi, hogy a frissítést ne az egész oldal újratöltésével oldjuk meg, hanem a háttérben folyó **POST** kérésekkel frissítsük a LED állapotát és az ADC mérésből kapott adatokat

JavaScript események és frissülő elemek

- Gombnyomás esemény
  - a `<button>` elemekben található `onclick="setLED('ON')"` és `onclick="setLED('OFF')"` események hívják meg a `setLED(state)` függvényt
  - Ez egy **POST** kérés segítségével elküldi az új LED állapotot a szervernek.
  - A szerver válasza alapján frissítjük a NAelemet

Automatikus frissítés időzítéssel

- A **setInterval()** 2 másodpercenként meghívja a **fetch("/readADC")** függvényt, amely **GET** kérés segítségével lekéri az aktuális ADC értéket a szerverről.
- A szerver válaszát beírjuk a `<span id="ADCValue">0</span>` elembe

ledswitch_ajax_post.py – 4/1.

```
index_html = """<!DOCTYPE html>
<html>
<head> <meta http-equiv="content-type" content="text/html; charset=UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
<title>LED Control</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;}
   button {display: block; width: 120px; background-color: #3498db; border: none; color: white;
           padding: 13px 20px; text-decoration: none; font-size: 25px; margin: 0px auto 35px;
           cursor: pointer; border-radius: 4px;}
</style>
</head>
<body>
<h1>ESP32-C3 WebServer</h1>
<h3>Update web page without refresh</h3>
<button type="button" onclick="setLED('ON')">LED ON</button>
<button type="button" onclick="setLED('OFF')">LED OFF</button><br>
<div><p>ADC Value: <span id="ADCValue"> 0</span> V<br>
        LED State: <span id="LEDState">NA</span></p></div>
```

A cserére kijelölt elemek

Ezek a weblap látható elemei

ledswitch_ajax_post.py – 4/2.

```
<script>
function setLED(state) {        // A LED állapot kiküldése és az állapot kijelzése
    fetch("/setLED", {          // Ez a függvény gombnyomáskor aktiválódik
        method: "POST",
        body: JSON.stringify({ "LED": state }),
        headers: { "Content-Type": "application/json" }
    })
    .then(response => response.text())
    .then(data => {
        document.getElementById("LEDState").innerText = data;
    });
}

setInterval(function() {        // ADC érték frissítése 2 másodpercenként
    fetch("/readADC", {
        method: "GET",
        headers: { "Accept": "text/plain" }
    })
    .then(response => response.text())
    .then(data => { document.getElementById("ADCValue").innerText = data; });
}, 2000);
</script>
</body></html>"""
```

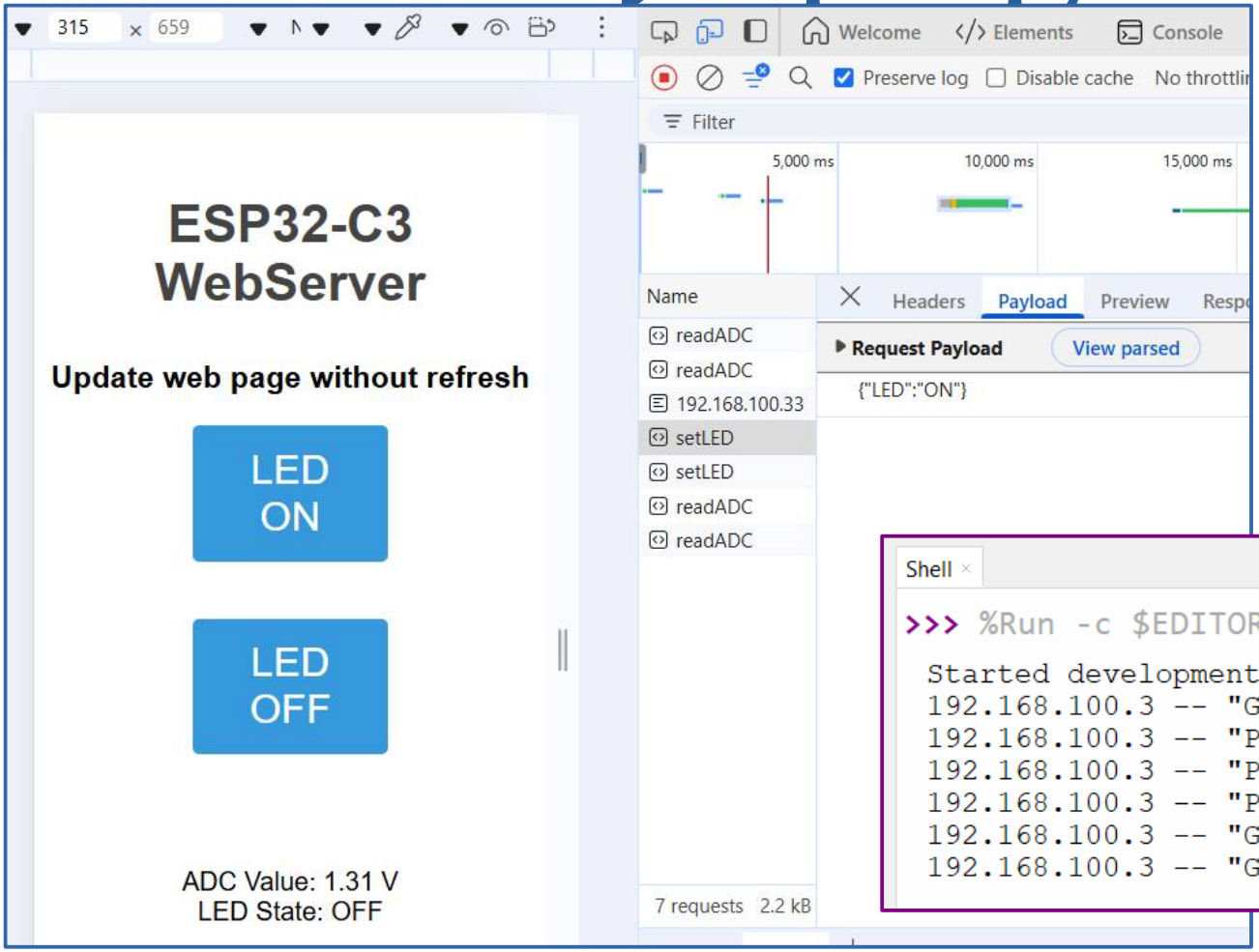
ledswitch_ajax_post.py – 4/3.

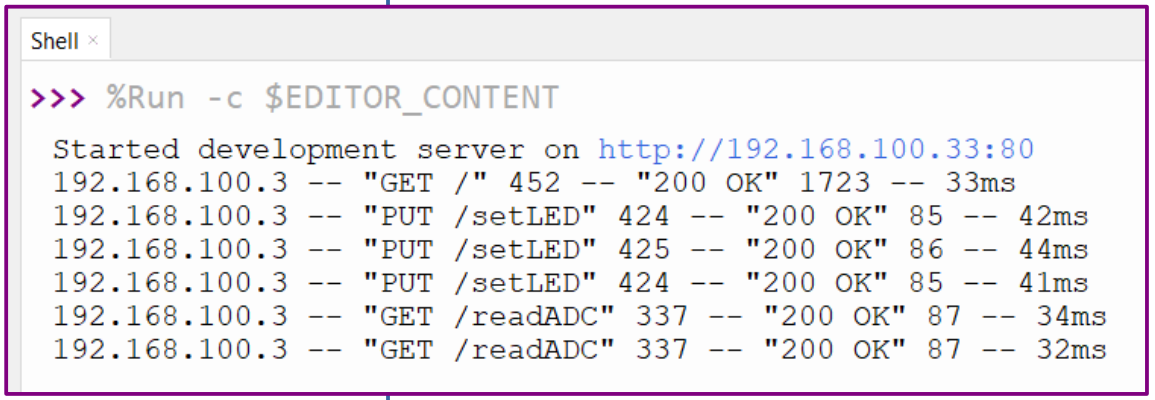
```
import board, digitalio, analogio, wifi, socketpool, json
from os import getenv
from adafruit_httpserver import Server, Request, Response, GET, POST, Route
led = digitalio.DigitalInOut(board.LED)     # LED és ADC konfigurálás
led.direction = digitalio.Direction.OUTPUT
adc = analogio.AnalogIn(board.IO0)

WIFI_SSID = getenv("CIRCUITPY_WIFI_SSID")  # Wi-Fi inicializálás a settings.toml-ból
WIFI_PASS = getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(WIFI_SSID, WIFI_PASS)
pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, debug=True)           # HTTPServer konfigurálás

def set_led(request: Request):              # A /set_led URI request kiszolgálása
    try:
        data = request.json()               # JSON beolvasása
        led_state = data.get("LED")
        if led_state == "ON":
            led.value = False               # LED BE
        elif led_state == "OFF":
            led.value = True                # LED KI
        return Response(request, led_state, content_type="text/plain")
    except ValueError:
        return Response(request, "Hibás kérés!", content_type="text/plain", status=400)
```

ledswitch_ajax_post.py – 4/4.

```
# ADC lekérése ütemezetten
def read_adc(request: Request):
    voltage = adc.value * (3.3 / 65535)  # ADC érték átszámítása feszültségre
    return Response(request, f"{voltage:0.2f}", content_type="text/plain")

# Weboldal kiszolgálása
def serve_root(request: Request):
    return Response(request, index_html, content_type="text/html")

# Route-ok regisztrálása
server.add_routes([
    Route("/", [GET, POST], serve_root),
    Route("/setLED", [POST], set_led),
    Route("/readADC", [GET], read_adc),
])

# Szerver indítása a 80-as porton
server.serve_forever(str(wifi.radio.ipv4_address), port=80)
```

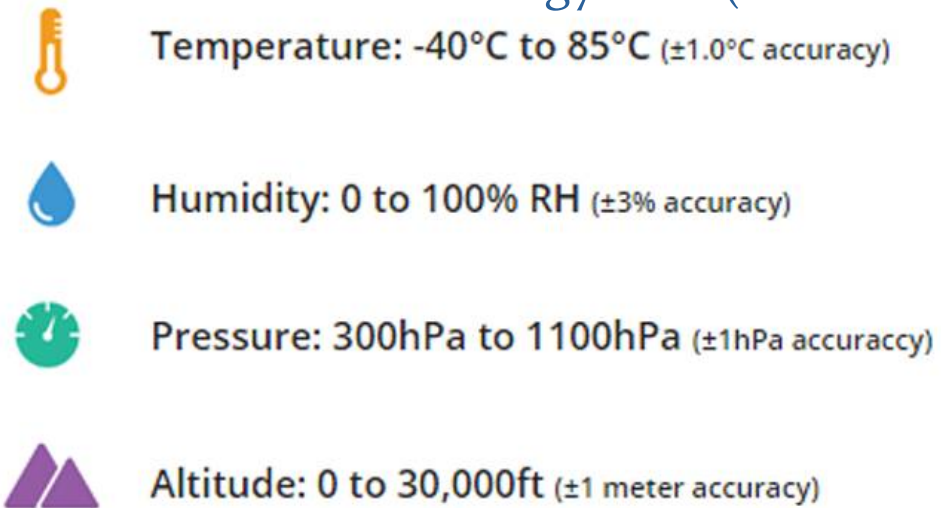
ledswitch_ajax_post.py futási eredménye

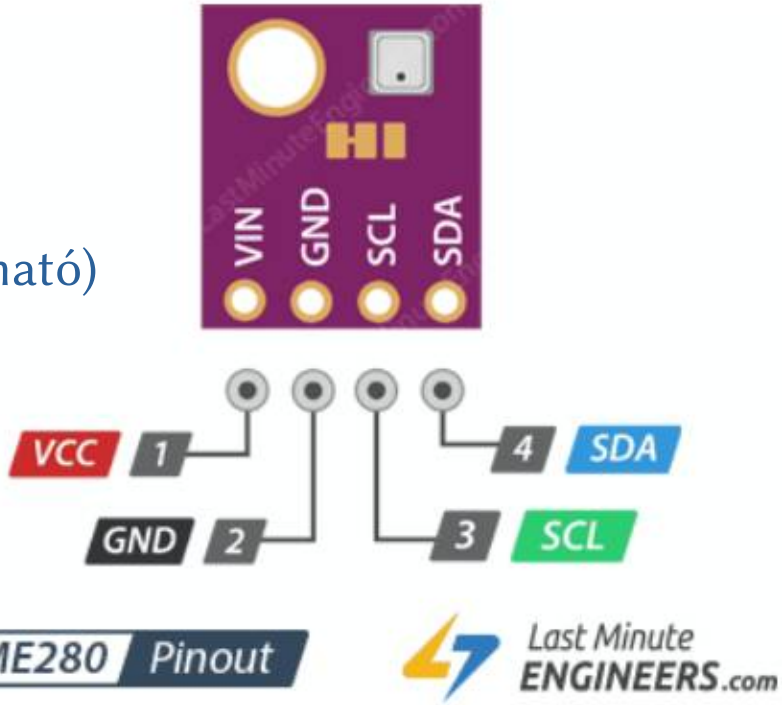




A BME280 I2C szenzor

- A Bosch által gyártott **BME280** egy környezeti érzékelő, amely hőmérsékletet, relatív páratartalmat és légnyomást mér. Az elődjeitől (**BMP85, BMP180, BMP280**) eltérően a **BME280** relatív páratartalom mérést is támogat
- **Főbb jellemzők**
  - Interfész: I2C és SPI (mi I2C-t használunk)
  - Tápfeszültség: 3,3V vagy 5V (modultól függően)
  - I2C cím: 0x76 vagy 0x77 (átforrasztással választható)





Az ábrák forrása: Last Minute Engineers

A BME280 I2C szenzor használata CircuitPythonban

- A **BME280** szenzor **CircuitPythonban** történő használatához az Adafruit_CircuitPython_Bundle csomagból telepíthető**Adafruit_BM280** programkönyvtárat használjuk
- Arra ügyeljünk, hogy ez a programkönyvtár „szigorúbb”, csak a **busio**-val konfigurált I2C buszt tudja használni (lásd lentebb)

```
import board
import busio
from adafruit_bme280 import basic as adafruit_bme280

i2c = busio.I2C(board.SCL, board.SDA)    # Itt nem jó a board.I2C()!
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c, address=0x76)

print(f"Hőmérséklet: {bme280.temperature:.1f} °C")
print(f"Páratartalom: {bme280.humidity:.1f} %")
print(f"Légnyomás: {bme280.pressure+14.4:.1f} hPa")
```

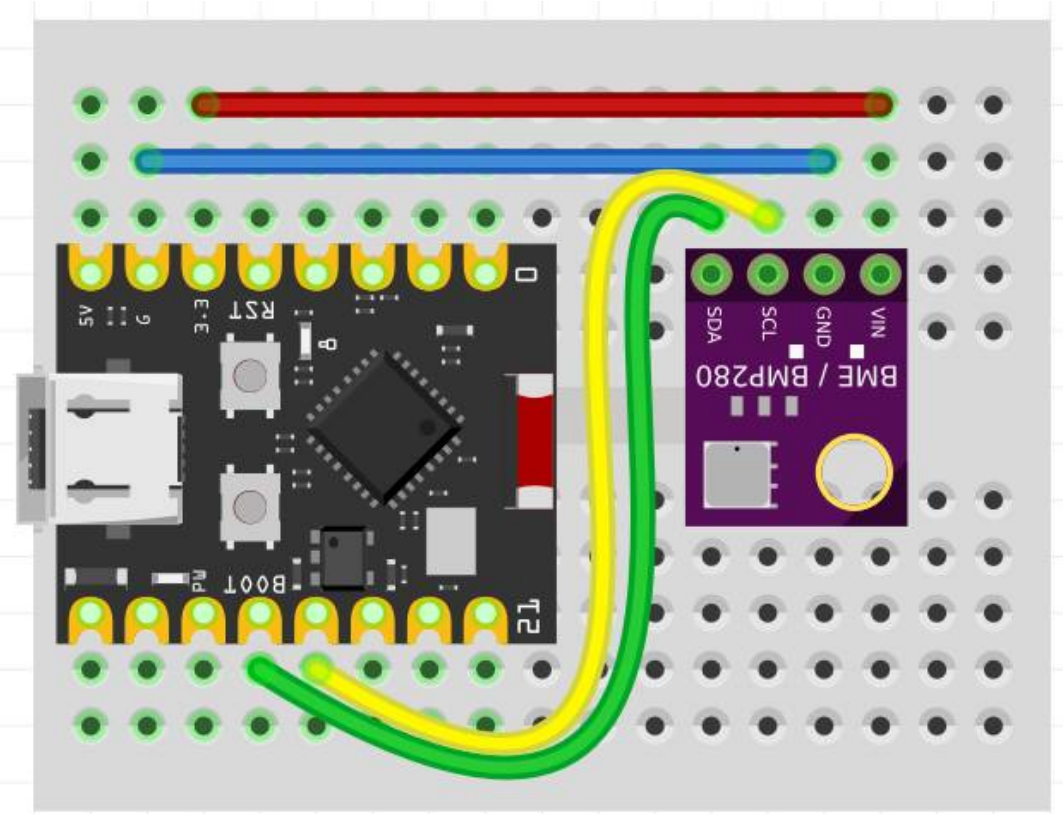
ESP32-C3 Időjárásállomás BME280 szenzorral

- Ez a projekt egy IoT alapú időjárás állomás, amely az ESP32-C3 mikrovezérlőt és a **BME280** szenzort használja a hőmérséklet, páratartalom és légnyomás mérésére. A mért adatok egy WiFi-n keresztül elérhető webes felületen jeleníthetők meg, így bármilyen eszközről könnyen megtekinthetők.
- A LastMinuteEngineers eredetileg ESP8266-re Arduino környezetben készült projektjét ESP32-C3 és CircuitPython környezetre adaptáltunk, az alábbi módosításokkal:
- A HTML és a CSS kódot fájlokba szerveztük és a **/html** mappába töltöttük:
  - **index.html** – a főoldal
  - **style.css** – a stíluslap
  - **404.html** – egyedi hibaoldal

  a program így áttekinthetőbb és könnyebben karbantartható
- A HTTP kérések kiszolgálása három ágon folyik:
  - "/" – a httpserver beépített kiszolgálója a **/html/index.html** fájlt küldi ki
  - "/**<file>**" – a **/html** mappa fájljainak kiküldése, ill a **404-es hiba** kezelése
  - "/**readBME280**" – AJAX üzenetváltás a szenzor adatainak automatikus lekéréséhez

ESP32-C3 Időjárásállomás BME280 szenzorral

- A **BME280** szenzor az alapértelmezett I2C interfészen keresztül csatlakozik az **ESP32-C3**-hoz:
  - VIN ⟶ 3.3V (ESP32-C3)
  - GND ⟶ GND (ESP32-C3)
  - SDA ⟶ GPIO8 (ESP32-C3)
  - SCL ⟶ GPIO9 (ESP32-C3)
- A **BME280** szenzor adatait JSON formátumban adjuk át a böngészőnek, így kényelmesebb az adatok kezelése. Például:




```
{"humidity": 49, "temperature": "24.6", "pressure": "1012", "altitude": 133}
```

httpserver_bme280_ajax.py – 2/1.

```
import os, board, busio, wifi, socketpool, json
from adafruit_bme280 import basic as adafruit_bme280
from adafruit_httpserver import Server, Request, Response, GET
from adafruit_httpserver import MIMETypes, FileResponse, NOT_FOUND_404

# MIME típusok konfigurálása
MIMETypes.configure(
    default_to="text/plain",
    keep_for=[".html", ".css", ".js", ".png", ".jpg", ".jpeg", ".gif", ".ico"],
)

# BME280 inicializálása
i2c = busio.I2C(board.SCL, board.SDA)
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c, address=0x76)
bme280.sea_level_pressure = 1013.25

# Wi-Fi csatlakozás
WIFI_SSID = os.getenv("CIRCUITPY_WIFI_SSID")
WIFI_PASS = os.getenv("CIRCUITPY_WIFI_PASSWORD")
wifi.radio.connect(WIFI_SSID, WIFI_PASS)
pool = socketpool.SocketPool(wifi.radio)
server = Server(pool, "/html", debug=True)     # Automatikus fileserver a /html mappából
```

httpserver_bme280_ajax.py – 2/2.

```
# JSON generálás a BME280 adatokhoz (Debrecenre korrigálva)
def generate_json():
    return json.dumps({
        "temperature": f"{bme280.temperature:.1f}",
        "humidity": int(bme280.humidity),
        "pressure": f"{round(bme280.pressure + 14.4)}",
        "altitude": int(bme280.altitude)
    })

@server.route("/readBME280")
def send_sensor_data(request: Request):
    return Response(request, content_type="application/json", body=generate_json())

@server.route("/<path>")
def file_handler(request: Request, path: str):
    if path in os.listdir("/html"):
        return FileResponse(request, filename=path)
    else:
        return FileResponse(request, filename="404.html", status=NOT_FOUND_404)

# Szerver indítása
server.serve_forever(str(wifi.radio.ipv4_address), 80)
```

Az index.html állomány tartalma 2/1.

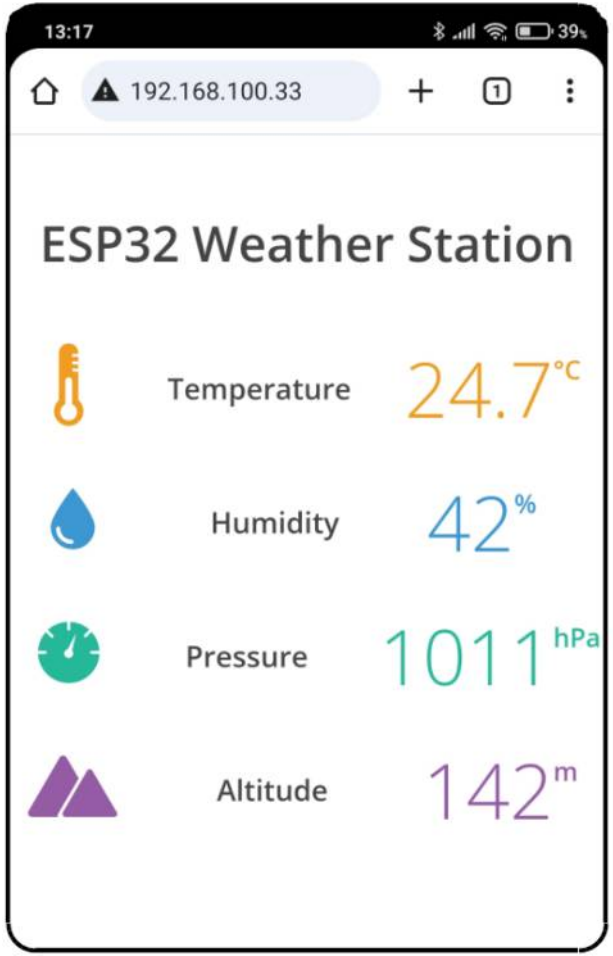
```
<html><head>
  <title>ESP32 Weather Station</title>
  <meta name='viewport' content='width=device-width, initial-scale=1.0'>
  <link rel="stylesheet" type="text/css" href="style.css">
  <link rel="icon" href="data:,">
  <link href='https://fonts.googleapis.com/css?family=Open+Sans:300,400,600' rel='stylesheet'>
  <script>
    setInterval(loadDoc, 2000);
    function loadDoc() {
      var xhttp = new XMLHttpRequest();
      xhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
          var obj =JSON.parse(this.responseText);
          document.getElementById("bme_temperature").innerHTML = obj.temperature;
          document.getElementById("bme_humidity").innerHTML = obj.humidity;
          document.getElementById("bme_pressure").innerHTML = obj.pressure;
          document.getElementById("bme_altitude").innerHTML = obj.altitude;
        }
      };
      xhttp.open("GET", "/readBME280", true);
      xhttp.send();
    }
  </script>
</head>
```

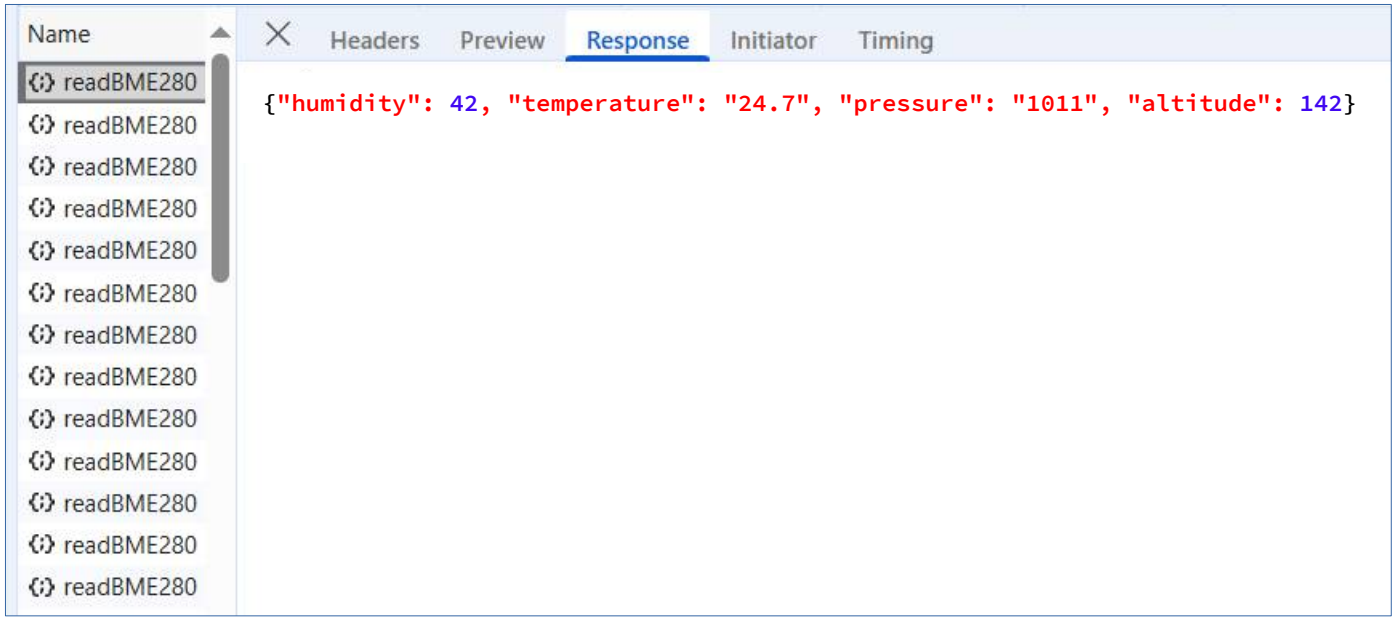
Deserialize JSON

Az index.html állomány tartalma 2/2.

```
<body><h1>ESP32 Weather Station</h1>
  <div class='container'>
    <div class='data temperature'>
      <div class='side-by-side icon'> <svg>... itt nem részletezzük ...</svg> </div>
      <div class='side-by-side text'>Temperature </div>
      <div class='side-by-side reading'><span id="bme_temperature">0</span>
        <span class='superscript'>&deg;C</span></div>
    </div>
    <div class='data humidity'>
      <div class='side-by-side icon'> <svg>... itt nem részletezzük ...</svg> </div>
      <div class='side-by-side text'>Humidity </div>
      <div class='side-by-side reading'><span id="bme_humidity">0</span>
         <span class='superscript'>%</span></div>
    </div>
    <div class='data pressure'>
      <div class='side-by-side icon'> <svg>... itt nem részletezzük ...</svg> </div>
      <div class='side-by-side text'>Pressure </div>
      <div class='side-by-side reading'><span id="bme_pressure">0</span>
         <span class='superscript'>hPa</span></div>
    </div>
    <div class='data altitude'>
      <div class='side-by-side icon'><svg>... itt nem részletezzük ...</svg> </div>
      <div class='side-by-side text'>Altitude </div>
      <div class='side-by-side reading'><span id="bme_altitude">0</span>
        <span class='superscript'>m</span></div>
    </div>
  </div></body></html>
```

httpserver_bme280_ajax.py futási eredménye





Az ESP32 C3 Super Mini kártya kivezetései