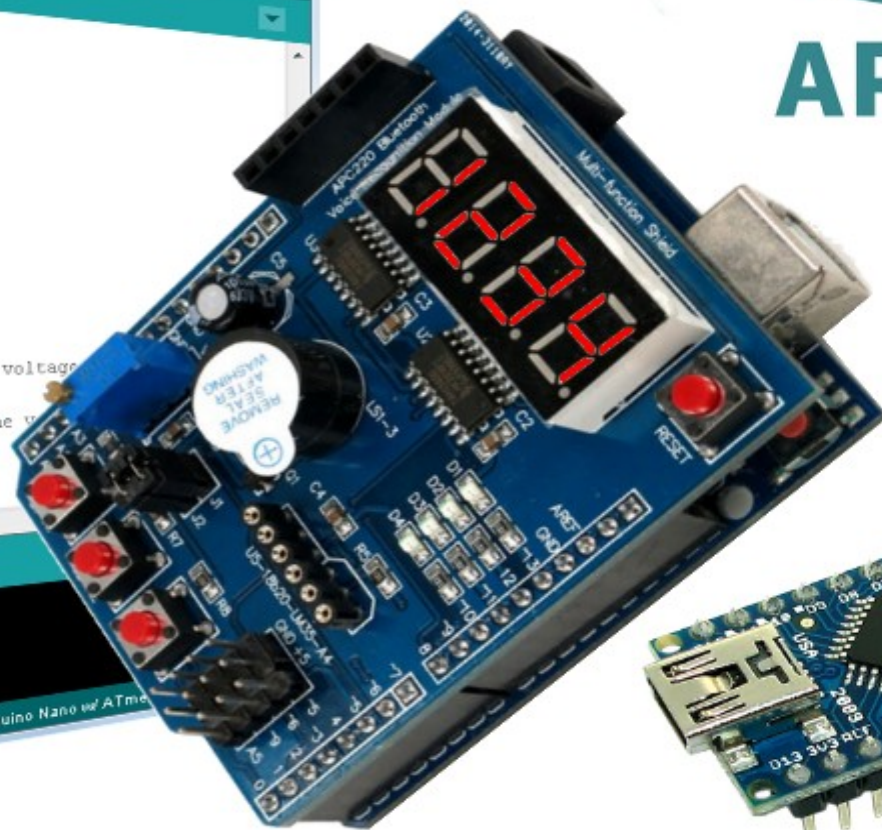
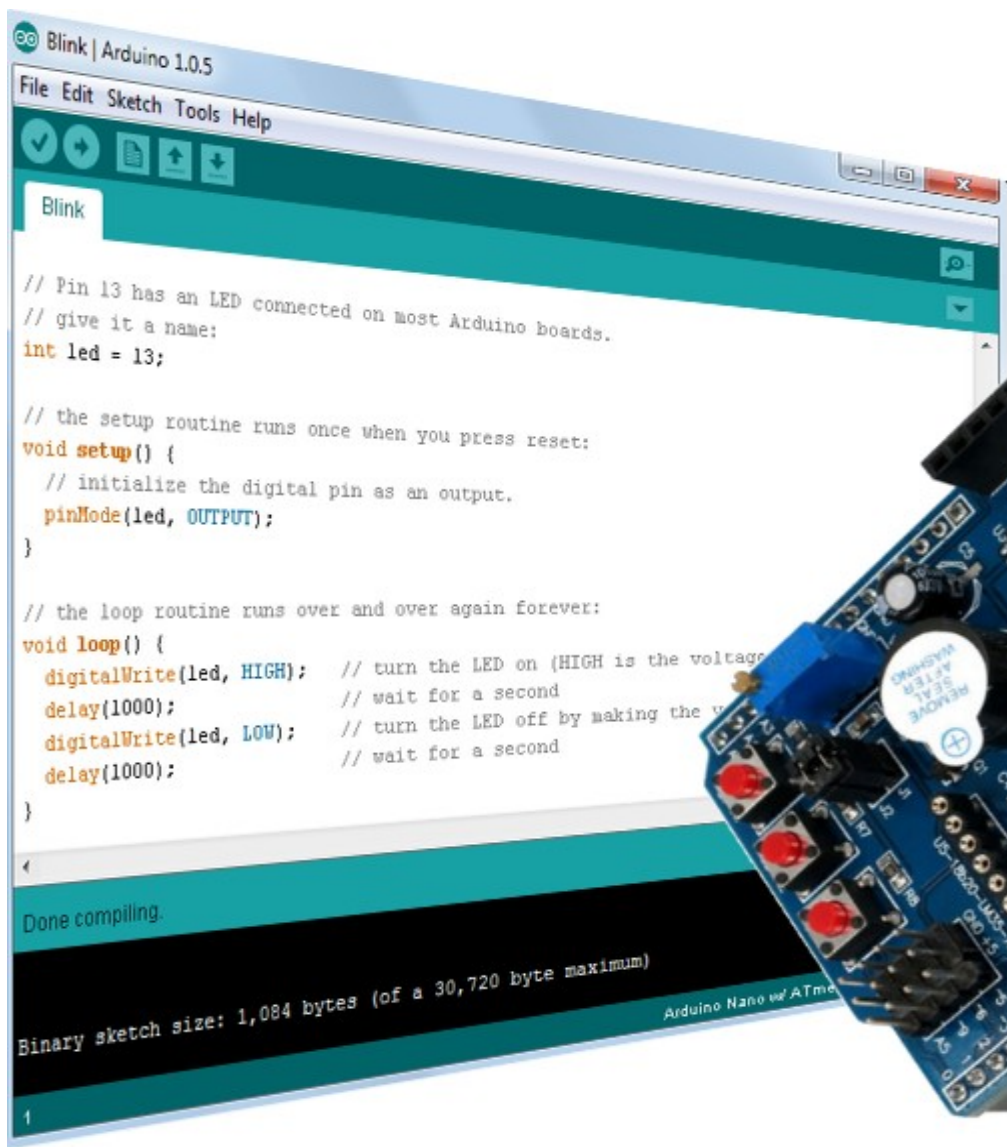


# Arduino tanfolyam kezdőknek és haladóknak



## 12. ESP8266 programozása Arduino környezetben

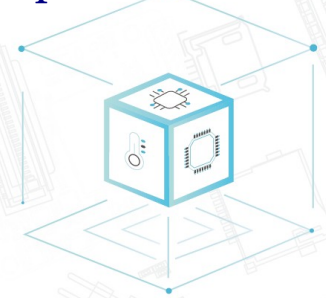
# Felhasznált és ajánlott irodalom

---

- Arduino Create: [Using Arduino IDE to Program NodeMCU](#)
- ESP8266 Community: [ESP8266 Arduino Core's documentation](#)
- Rui & Sara Santos: [Random Nerd Tutorials - ESP8266 projects](#)
- Last Minute Engineers: [Insight Into ESP8266 NodeMCU](#)

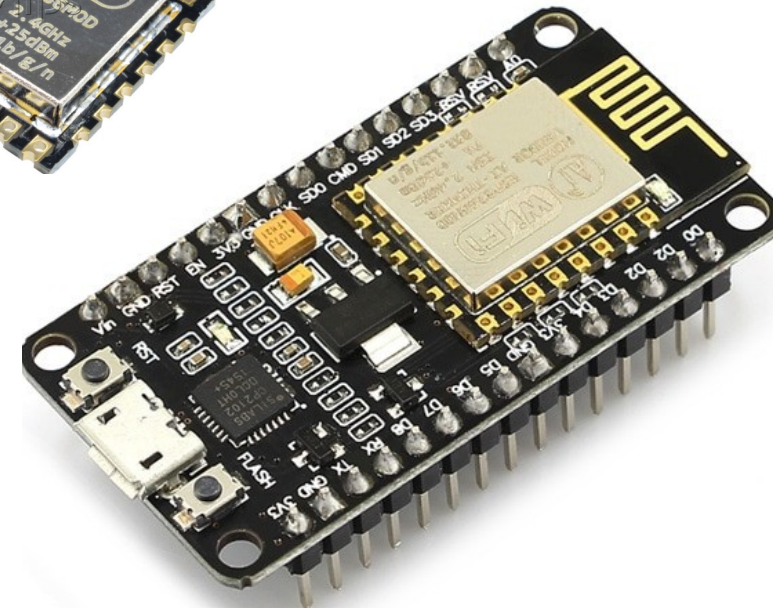
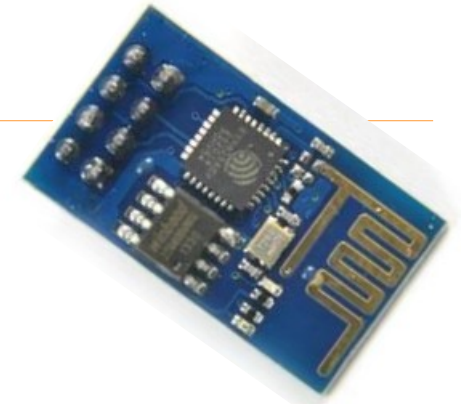
# Az ESP8266EX mikrovezérlő

- Tensilica L106 32-bit RISC CPU (80/160 MHz)
- WiFi modul IEEE 802.11 b/g/n
  - ❖ 32 KiB instruction RAM
  - ❖ 32 KiB instruction cache RAM
  - ❖ 80 KiB user-data RAM
  - ❖ 16 KiB ETS system-data RAM
- Külső QSPI flash: tipikusan 512 KiB to 4 MiB
- 16 GPIO kivezetés
- I2C, SPI, UART
- 10 bit ADC
- Beépített bootloader
- A termék honlapja: [www.espressif.com/en/products/socs/esp8266](http://www.espressif.com/en/products/socs/esp8266)



# ESP8266 kártyák

- **ESP-01:** 512 KiB memóriával, PCB antennával és korlátozott számú kivezetéssel
- **ESP-01:** 1 MiB memóriával, PCB antennával (a RESET láb nincs kivezetve, a memória csak DIO módban kezelhető)
- **ESP12-E:** 4 MiB memóriával, PCB antennával, 2 mm-es lábtávolsággal
- **NodeMCU kártya:** ESP12-E modul, 3,3 V-os stabilizátor, CP1202 (vagy CH340) USB-UART konverter, Reset és Boot nyomógombok



# Fejlesztőeszközök ESP8266-hoz

---

- Expressif SDK: [ESP-SDK](#)
- Arduino IDE + [Arduino core for ESP8266](#)
- NodeMCU Lua: [nodemcu.com/index\\_en.html](http://nodemcu.com/index_en.html)
- MicroPython: [micropython.org/download/](http://micropython.org/download/), [tutorial](#)
- Zephyr projekt: [docs.zephyrproject.org](http://docs.zephyrproject.org)
- Mongoose OS: [mongoose-os.com/mos.html](http://mongoose-os.com/mos.html)
- Blynk: [blynk.io/en/developers](http://blynk.io/en/developers)
- Simba: <http://simba-os.readthedocs.io/>, [github.com/eerimoq/simba](https://github.com/eerimoq/simba)
- Visual Studio Code + PlatformIO: [docs.platformio.org](http://docs.platformio.org)
- Lets Control It: [ESPEasy](#)

# ESP8266 kiegészítés telepítése Arduino környezetbe

---

- Az Arduino IDE **Board Manager** funkciója lehetővé teszi, hogy más hardver eszközökhöz (ESP8266, ESP32, STM32, Arduino Due stb) is használhassuk a fejlesztői környezetet
- Amit telepítünk: új fordító/linker, fejléc állományok, programkönyvtárak, programletöltő segédprogramok
- A telepítés után az ESP8266 kártyák (esetünkben a NodeMCU kártya) az Arduino UNO/Nano kártyákhoz hasonlóan önállóan programozható/használható
- **A telepítés előfeltételei:**
  - ❖ **Arduino IDE** (jelenleg az 1.8.13 verzió) telepítve legyen
  - ❖ **Python 2.7** vagy újabb verzió (nálam a v3.7 van telepítve)
  - ❖ A **Board Manager** listáját bővíteni kell az **ESP8266 Arduino Core** elérhetőségével, hogy az **ESP8266** kártyákat is felvehessük a választékba és telepíthessük a hozzájuk tartozó szoftver csomagot

# ESP8266 kiegészítés telepítése Arduino környezetbe

- A **File/Preferences** menüpontra kattintva a felugró lap *Additional Board Manager URLs* rovatába másoljuk be:

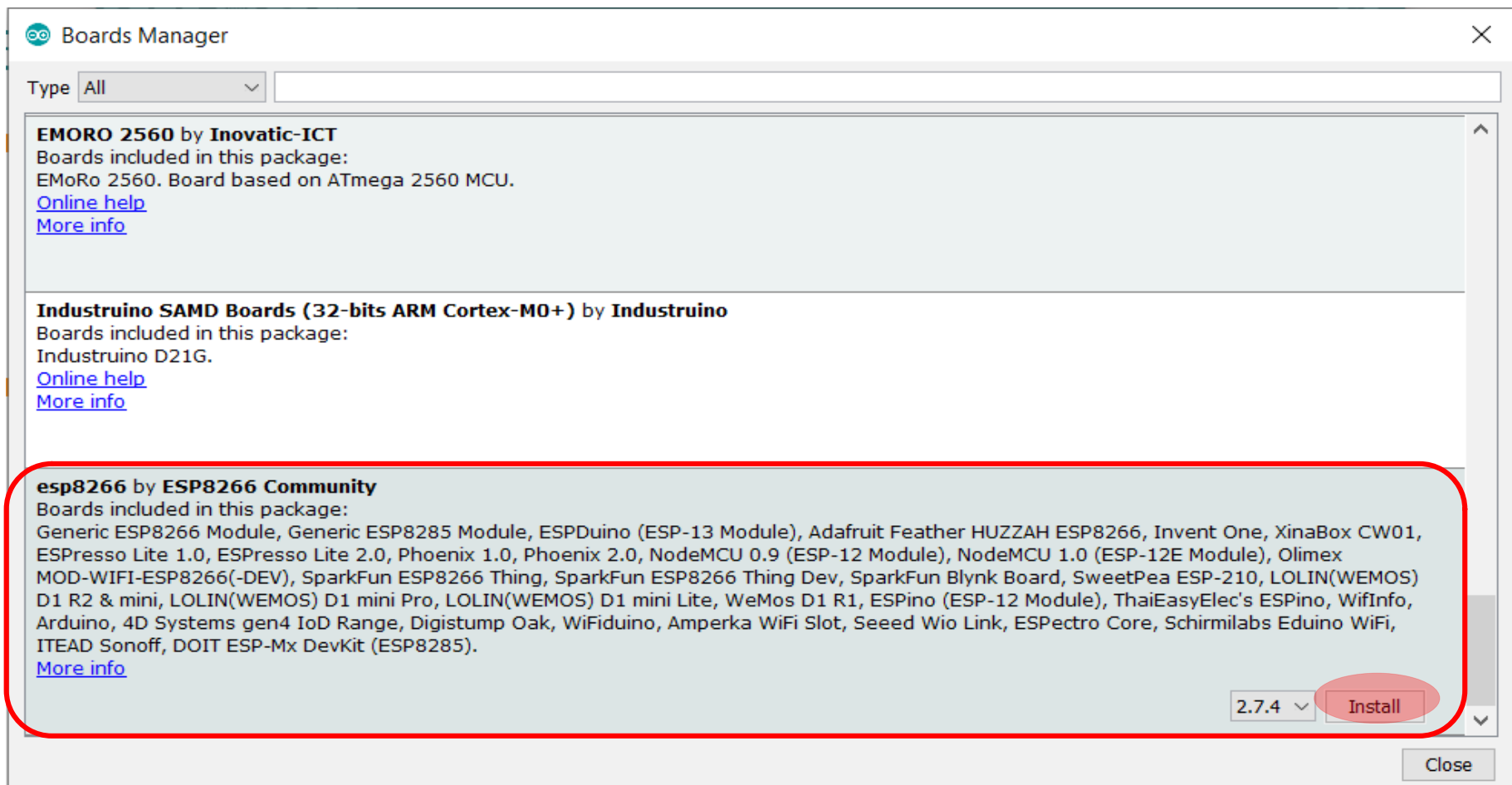
[http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json)

(több bejegyzés esetén mindegyik URL külön sorba kerüljön!)

The screenshot shows the Arduino IDE interface. On the left, the 'File' menu is open, and the 'Preferences' option is highlighted. The main window displays the 'Preferences' dialog box, which is currently on the 'Network' tab. The 'Additional Boards Manager URLs' field contains the URL [http://arduino.esp8266.com/stable/package\\_esp8266com\\_index.json](http://arduino.esp8266.com/stable/package_esp8266com_index.json), which is underlined in red. Other settings visible include the sketchbook location, editor language, font size, interface scale, theme, and various checkboxes for compiler warnings and code folding.

# ESP8266 kiegészítés telepítése Arduino környezetbe

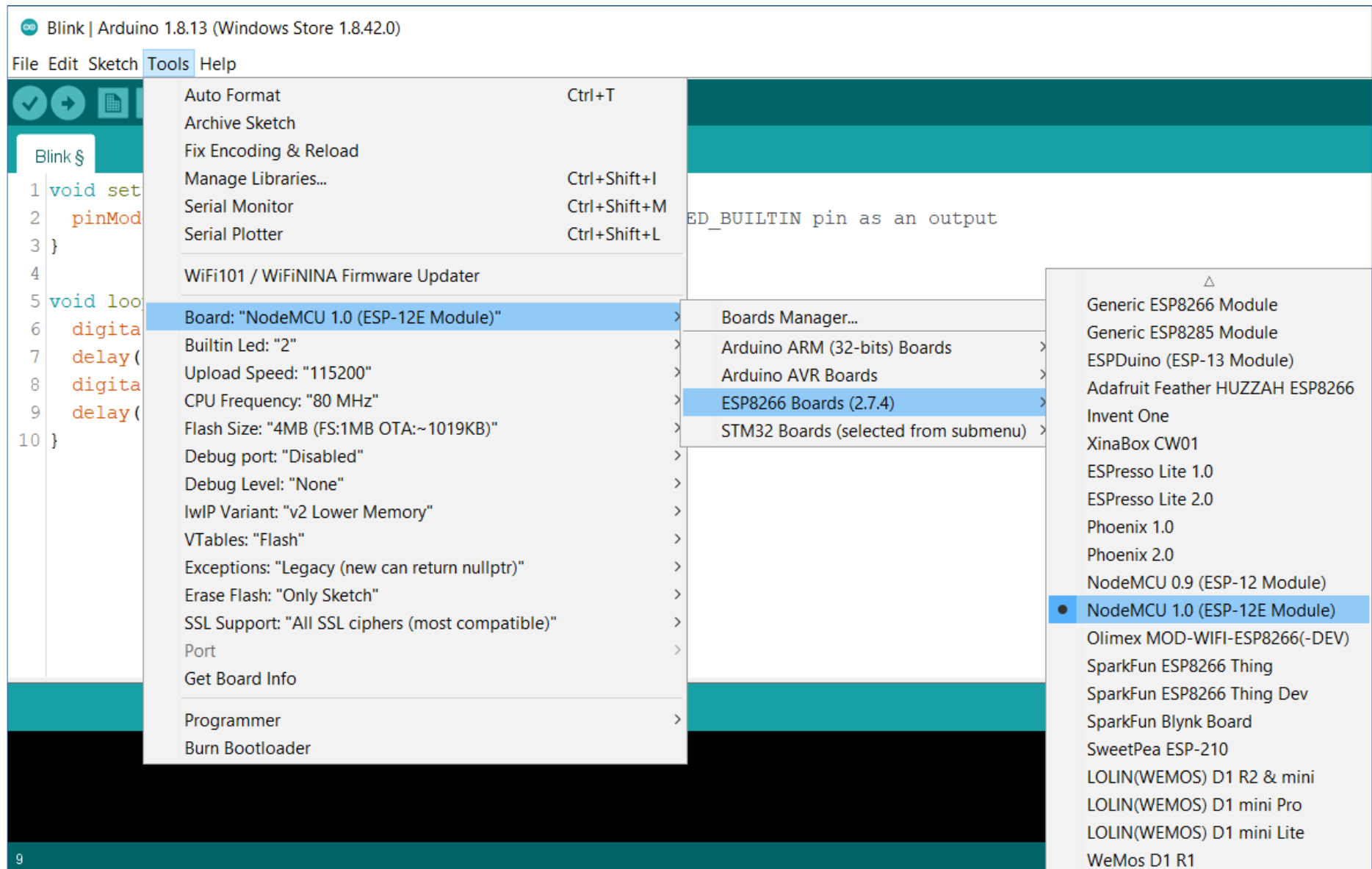
- A **Tools/Board/Boards Manager** menüpontra kattintva a felugró lapon keressük meg az **esp8266 by ESP8266 Community** csomagot, majd kattintsunk az **Install** gombra





# ESP8266 kiegészítés telepítése Arduino környezetbe

- A **Tools/Board** menüben állítsuk be a kártyát, csatlakozás után a portot is



# ESP8266\_blink.ino – a beépített LED villogtatása

- Fordítsuk le és töltsük le az alábbi LED villogtató programot
- Letöltéskor az alábbihoz hasonló üzenetnek kell megjelennie

```
/* Blink
   Felvillantjuk a beépített LED-et, azután egy
   másodpercre lekapcsoljuk, s ezt ismételtetjük.
*/

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);    // Digitális kimenet
}

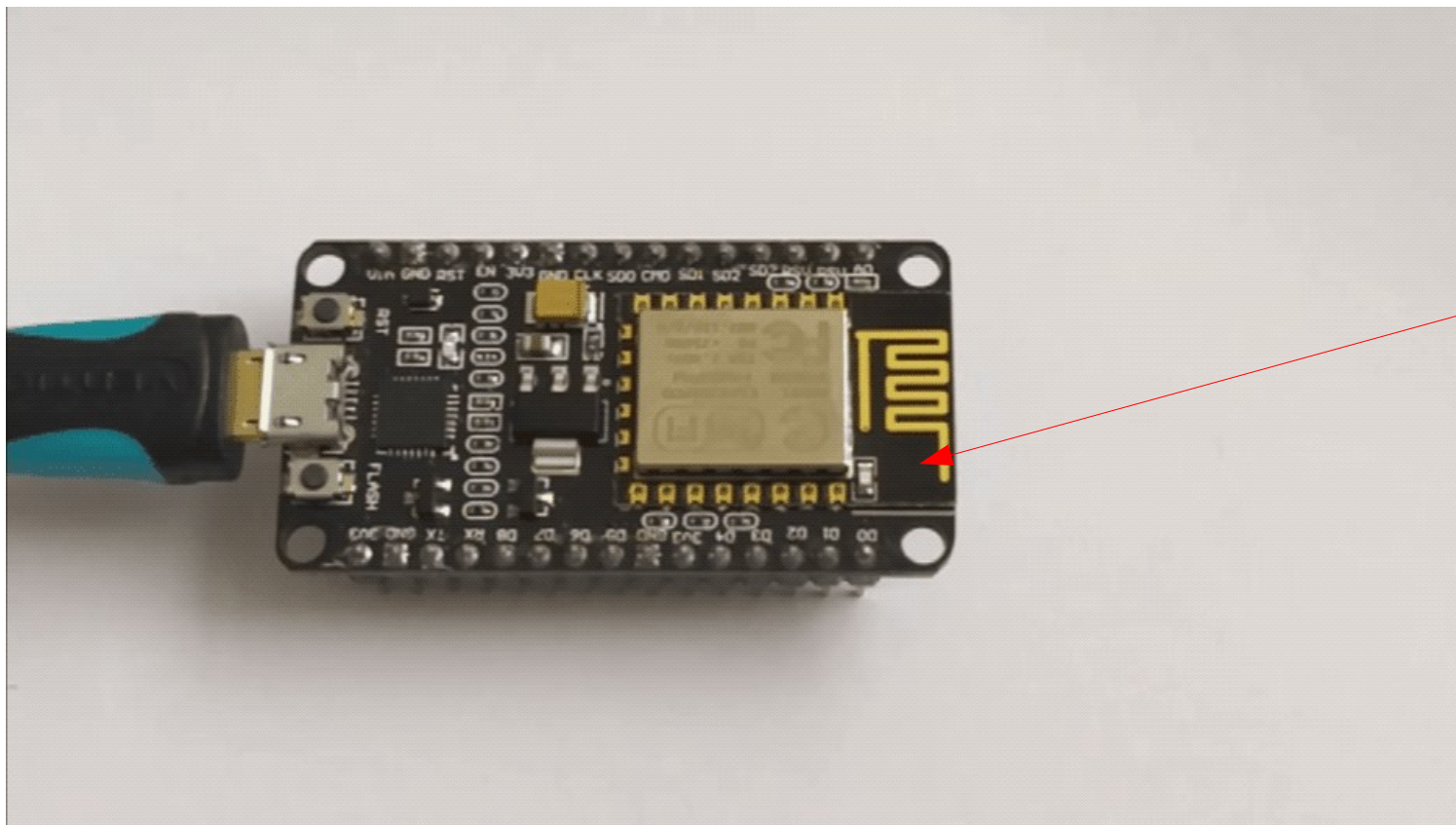
void loop() {
  digitalWrite(LED_BUILTIN, LOW);  // bekapcsoljuk a LED-et
  delay(100);                      // 100 ms várakozás
  digitalWrite(LED_BUILTIN, HIGH); // kikapcsoljuk a LED-et
  delay(1000);                     // 1 s várakozás
}
```

```
esptool.py v2.8
Serial port COM10
Connecting....
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: a0:20:a6:1c:51:07
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 261472 bytes to 193133...
Wrote 261472 bytes (193133 compressed) at 0x00000000 in 17.7 seconds (effective 118.4
kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

# ESP8266\_blink.ino:futási eredmény

- A NodeMCU kártyán a beépített LED a GPIO16 (D0) kivezetéshez tartozik, s a LED lehúzásra világít
- Ezen a kivezetésen PWM nem használható



LED

# 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

A Flash memóriát kezelő kivezetések foglaltak!

LED B  
GPIO16

Vin  
+5V - +10 V

LED A  
GPIO2,  
LED\_BUILTIN

Serial

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

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

# Digitális I/O függvények

- **pinMode(*pin*, *mode*)** – a kivezetés üzemmódjának beállítása  
GPIO1 – GPIO15 lehet INPUT, INPUT\_PULLUP vagy OUTPUT,  
GPIO16 pedig INPUT, INPUT\_PULLDOWN\_16 vagy OUTPUT
- **digitalWrite(*pin*, *level*)** – kimeneti szint beállítása (HIGH/LOW)
- **digitalRead(*pin*)** – bemeneti szint beolvasása (HIGH/LOW)
- **Programmegszakítások:** a digitális bemenetekhez a szokásos módon megszakítást rendelhetünk (`attachInterrupt()`, `detachInterrupt()`), kivéve GPIO16 és az analóg A0 bemenetet. A választható megszakítási típusok: CHANGE, RISING, FALLING
- A megszakítások **visszahívási függvényeinek** a RAM-ban kell lennie, amit a CACHE\_RAM\_ATTR direktíva garantál, például:  
`ICACHE_RAM_ATTR void gpio_change_handler(void *data) { ... }`

# ESP8266\_interrupt.ino

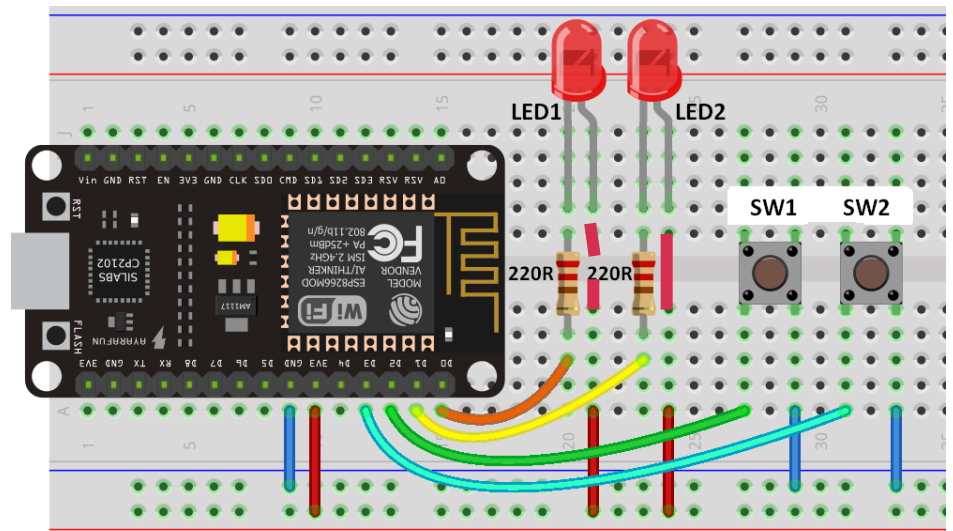
```
#define LED1 D0
#define LED2 D1
#define SW1 D2
#define SW2 D3

ICACHE_RAM_ATTR void sw1_handler(void) {
    digitalWrite(LED2, LOW); // LED2 on
}

ICACHE_RAM_ATTR void sw2_handler(void) {
    digitalWrite(LED2, HIGH); // LED2 off
}

void setup() {
    pinMode(LED1, OUTPUT); digitalWrite(LED1, HIGH);
    pinMode(LED2, OUTPUT); digitalWrite(LED2, HIGH);
    pinMode(SW1, INPUT_PULLUP); pinMode(SW2, INPUT_PULLUP);
    attachInterrupt(SW1, sw1_handler, FALLING);
    attachInterrupt(SW2, sw2_handler, FALLING);
}

void loop() {
    digitalWrite(LED1, LOW); // Turn the other LED on
    delay(100); // Wait for a 100 ms
    digitalWrite(LED1, HIGH); // Turn the other LED off
    delay(1000); // Wait for 1000 ms
}
```



SW1 és SW2 nyomógomb megszakításokkal be és ki kapcsolgatjuk LED2-t, miközben folyamatosan villogtatjuk LED1-et

# Soros kommunikáció

---

- A **Serial** objektum ugyanúgy használható, mint az Arduinonál
- A hw FIFO buffer 128 bájt, emellett az **Rx** buffer további 256 bájtos RAM bufferrel is rendelkezik, mérete a **begin** metódus hívása előtt módosítható a **setRxBufferSize(size\_t size)** metódus hívásával
- **write()** - nem blokkol, ha a kiírandó karakter belefér a FIFO tárba
- **read()** - nem blokkol, akkor sem, ha nincs beérkezett karakter
- **readBytes()** - blokkol, amíg be nem érkezik a kért számú karakter
- **flush()** - blokkol várakozás, amíg be a Tx FIFO ki nem ürül
- **Serial** az **UART0** modult használja: **GPIO1** (Tx) és **GPIO3** (Rx), de a **swap()** metódus átváltja a **GPIO15** (Tx) and **GPIO13** (Rx) lábakra. Újabb **swap()** hívás visszaváltja a **GPIO1** (Tx) és **GPIO3** (Rx) lábakra
- **Serial1** csak adatküldésre használható (**GPIO2** Tx), nem tartozik hozzá Rx bemenet

# Analóg bemenet

- Az **ESP8266EX** mikrovezérlő egy 10 bites ADC-vel rendelkezik, amely vagy a tápfeszültséget, vagy egy, a TOUT lábra kötött 0 – 1.0 V közötti külső feszültséget tudja mérni
- NodeMCU esetén az A0 bemenet egy feszültségosztón keresztül csatlakozik a TOUT lábhoz, így a mérési tartomány: 0 – 3.3 V
- **analogRead(A0)** – beolvassa és konvertálja az analóg feszültség értékét (0 – 1023 közötti érték)
- **A tápfeszültség méréséhez**
  - ❖ a **TOUT** lábat szabadon kell hagyni
  - ❖ a program legelején (közvetlenül az **#include ...** sorok után) az alábbi sort kell elhelyezni:

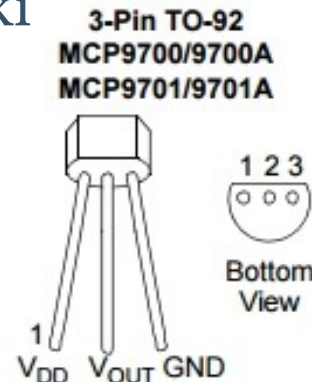
```
ADC_MODE(ADC_VCC);
```
  - ❖ a programon belül a feszültség értékét az **ESP.getVcc()** függvény hívásával kapjuk meg



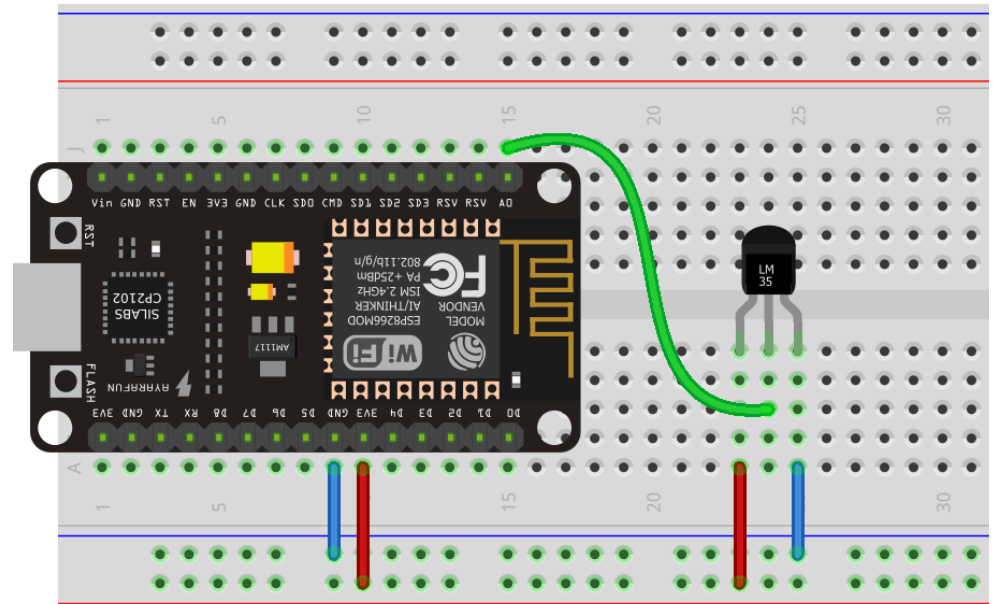
# ESP8266\_analog\_thermometer.ino

- Hőmérőt készítünk az MCP9700 szenzorral:  
0 °C-on 500 mV, érzékenység: 10 mV/°C
- Az eredményt a soros porton írjuk ki

```
void setup () {  
  Serial.begin(115200);  
}  
  
void loop () {  
  long reading = 0;  
  for (int i = 0; i < 3110; i++) {  
    reading += analogRead(A0);  
  }  
  long voltage = reading / 1024;  
  Serial.print (voltage); // mV  
  Serial.print (" mV, ");  
  float tempC = (voltage - 500)/10.0;  
  Serial.print (tempC, 1); // Celsius  
  Serial.print (" °C, ");  
  float tempF = (tempC * 9 / 5) + 32;  
  Serial.print (tempF, 1); // Fahrenheit  
  Serial.println (" °F");  
  delay (5000);  
}
```



COM10		
762 mV,	26.2 °C,	79.2 °F
763 mV,	26.3 °C,	79.3 °F
762 mV,	26.2 °C,	79.2 °F
762 mV,	26.2 °C,	79.2 °F
763 mV,	26.3 °C,	79.3 °F
762 mV,	26.2 °C,	79.2 °F
762 mV,	26.2 °C,	79.2 °F
761 mV,	26.1 °C,	79.0 °F
762 mV,	26.2 °C,	79.2 °F

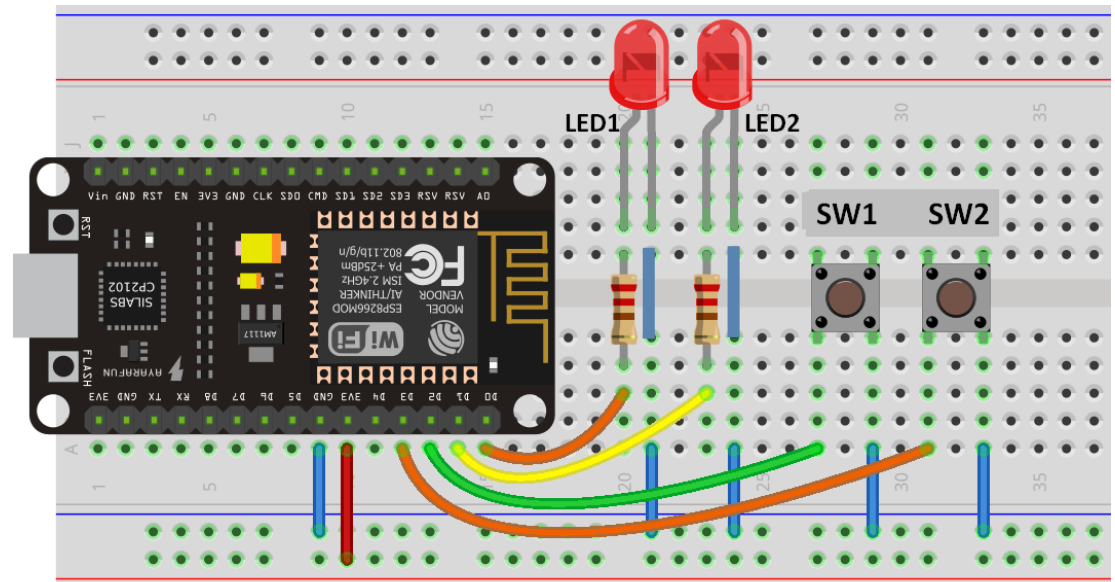


# PWM kimenetek vezérlése (**csak 3.0-tól!**)

- `analogWrite(pin, value)` – engedélyezi és beállítja a szoftveres PWM-et, ahol *pin* a használt kivezetés száma, *value* a kitöltés értéke (alapértelmezetten 0-255, de változtatható a határ...)
- `analogWrite(pin, value, openDrain)` formában is használható
- `analogWriteRange(new_range)` – a PWM periódus értékének megváltoztatása (15-65 535 közötti érték adható meg)
- `analogWriteResolution(bits)` – a PWM periódus bitszámának megváltoztatása (4-16 közötti érték adható meg)
- A PWM frekvencia alapértelmezetten 1kHz, de változtatható
- `analogWriteFreq(new_frequency)` – a PWM frekvencia módosítása (100Hz – 40 000Hz közötti érték adható meg)
- **Megjegyzés:** Az ESP8266 szoftveresen kezeli a PWM csatornákat. Több csatorna, nagyobb frekvencián leterheli a CPU-t
- **Megjegyzés:** a 2.7.4 kiadásnál PWM\_RANGE alapértelmezetten 1023, az újabb, 3.0 verziótól pedig 255 (hogy kompatibilis legyen az Arduinoval)

# ESP8266\_dimmer.ino

```
#include <Ticker.h>
Ticker dimmer;
#define LED1 D0
#define LED2 D1
#define SW1 D2
#define SW2 D3
uint16_t dim_value = 0;
void setup() {
  pinMode(LED1, OUTPUT);
  analogWriteRange(65535);
  analogWrite(LED2, dim_value);
  pinMode(SW1, INPUT_PULLUP);
  pinMode(SW2, INPUT_PULLUP);
  dimmer.attach(0.05, buttoncheck);
}
void loop() {
  analogWrite(LED2, dim_value * dim_value);
  digitalWrite(LED1, LOW); // Turn the other LED on
  delay(250); // Wait for 250 ms
  analogWrite(LED2, dim_value * dim_value);
  digitalWrite(LED1, HIGH); // Turn the other LED off
  delay(250); // Wait for 250 ms
}
void buttoncheck() {
  if (!digitalRead(SW1) && (dim_value < 255)) dim_value++;
  if (!digitalRead(SW2) && (dim_value > 0)) dim_value--;
}
```

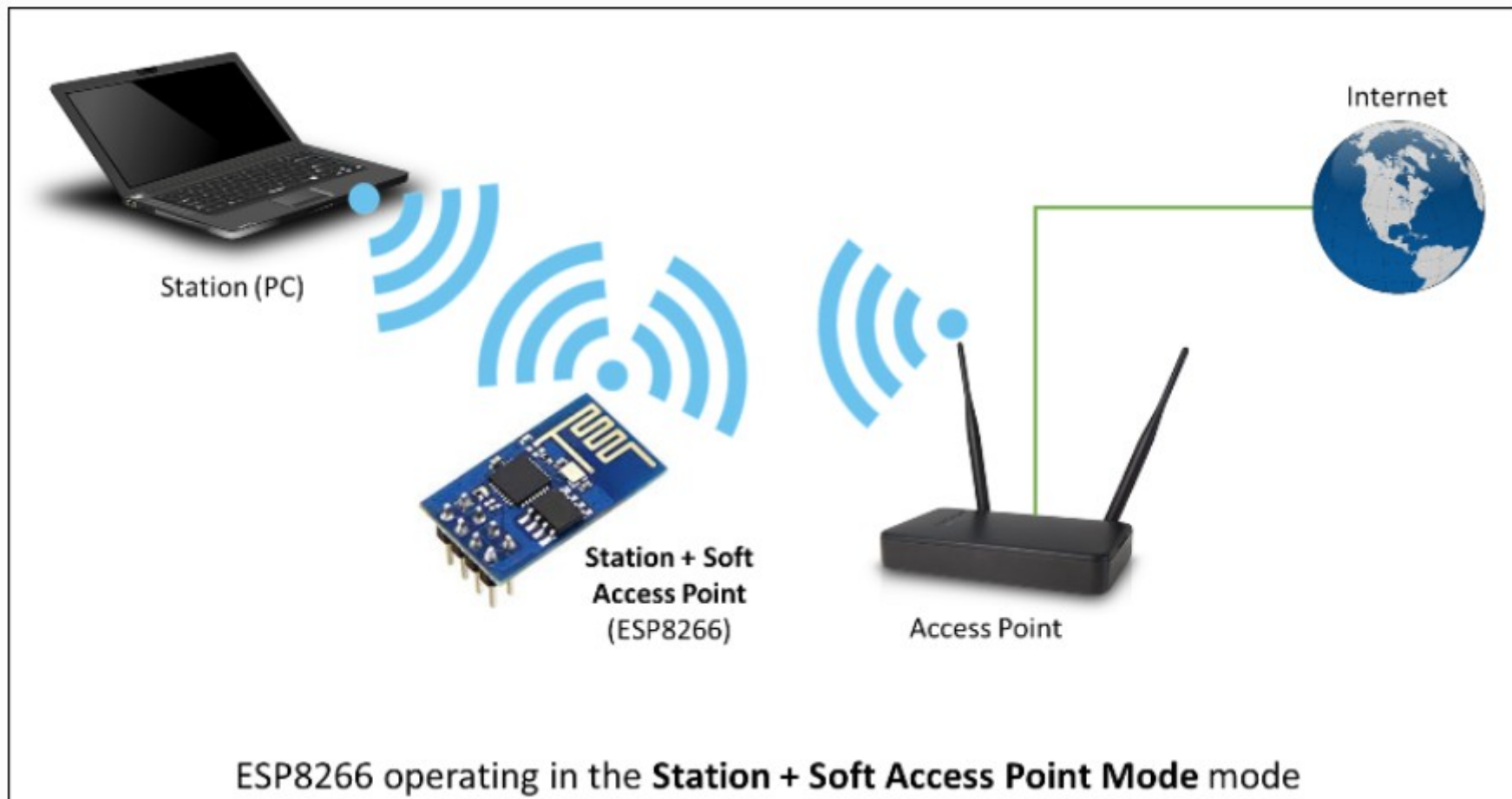


SW1 és SW2 nyomógomb segítségével LED2 fényerejét változtatjuk, miközben folyamatosan villogtatjuk LED1-et

A kapcsolás lényegében azonos a 14. oldalon láthatóval, csak a PWM jobb szabályozhatósága érdekében a LED anódok vezérlésére tértünk át. (Ügyeljünk a polaritásra!)

# Az ESP8266 WiFi programkönyvtár

- Az **ESP8266** WiFi könyvtára túlnőtte az Arduino WiFi lehetőségeit, ezért az eltérések miatt külön néven és dokumentációval érhető el
- Az **ESP8266** modul lehet hálózati végpont (*station*), hozzáférési pont (*access point*) vagy egyidejűleg mindkettő



# Az ESP8266WiFi programkönyvtár

- Az **ESP8266WiFi** programkönyvtár számos objektumosztályra tagozódik. A dokumentációban is ezen tagolódás szerint, kezelhető részekre bontva találjuk meg a részleteket
  - ❖ **Scan** – a WiFi hálózatok pásztázására és felderítésére szolgál
  - ❖ **Station** – ezt a módot használjuk egy hálózatra csatlakozáshoz
  - ❖ **Soft AP** – ezt a módot hozzáférést biztosít a többi eszköz számára
  - ❖ **Client** – a kliens objektum szerverek szolgáltatásait veszi igénybe
  - ❖ **Server** – a szerver kliensek kéréseit fogadja és szolgálja ki
  - ❖ **UDP** – user datagram protocol üzenetcsomagok küldése/fogadása
  - ❖ **Generic** – az ESP8266 SDK egyedi funkcióit kezelő csomag

**B**

BufferDataSource  
BufferedStreamDataSource

**C**

ClientContext

**D**

DataSource

**E**

ESP8266WiFiAPClass  
ESP8266WiFiClass  
ESP8266WiFiGenericClass  
ESP8266WiFiMulti  
ESP8266WiFiScanClass  
ESP8266WiFiSTAClass

**P**

ProgmemStream

**S**

SList  
SSLContext

**U**

UdpContext

**W**

WifiAPlist\_t

WiFiClient

WiFiClientSecure

WiFiEventHandlerOpaque

WiFiEventModeChange

WiFiEventSoftAPModeProbeRequestReceived

WiFiEventSoftAPModeStationConnected

WiFiEventSoftAPModeStationDisconnected

WiFiEventStationModeAuthModeChanged

WiFiEventStationModeConnected

WiFiEventStationModeDisconnected

WiFiEventStationModeGotIP

WiFiServer

WiFiUDP

# ESP8266\_wifi\_scan.ino

- Pásztázzuk a WiFi hálózatokat és soros porton listázzuk ki!

```
#include "ESP8266WiFi.h"

void setup() {
  Serial.begin(115200);
  Serial.println();
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
}

void loop() {
  Serial.print("Scan start ... ");
  int n = WiFi.scanNetworks();
  Serial.print(n);
  Serial.println(" network(s) found");
  for (int i = 0; i < n; i++) {
    Serial.println(WiFi.SSID(i));
  }
  Serial.println();
  delay(5000);
}
```

```
COM10

Scan start ... 12 network(s) found
DIGI-7Pe5
TP-LINK_72DC
DEBIWIFI
szabina
TP-LINK_D0D8C8
boroka 2.4
Krassoi
DIGI-TJ9r
Csilla
CSP-LINK
UPC1041130
DIRECT-UE-BRAVIA

Scan start ... 8 network(s) found
TP-LINK_D0D8C8
DIGI-7Pe5
DEBIWIFI
szabina
DIGI-TJ9r
```

Forrás: [arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html](https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html)

# ESP8266\_wifi\_scan2.ino 2/1.

- Aszinkron pásztázás és részletesebb kiírás

```
#include "ESP8266WiFi.h"

#define BLINK_PERIOD 250
long lastBlinkMillis;
boolean ledState;

#define SCAN_PERIOD 5000
long lastScanMillis;

void setup() {
  Serial.begin(115200);
  Serial.println();

  pinMode(LED_BUILTIN, OUTPUT);

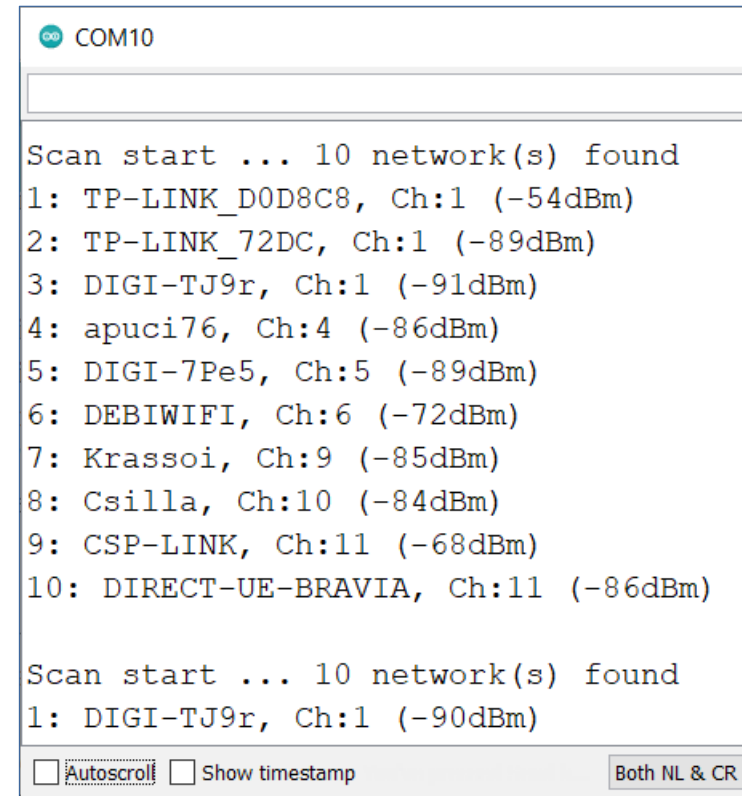
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);
}
```

A háttérben egy LED villogtatást is végzünk és az időzítéset `delay()` helyett a `millis()` függvénnyel oldjuk meg

Forrás: [arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html](https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html)

# ESP8266\_wifi\_scan2.ino 2/2.

```
void loop() {
  long currentMillis = millis();
  // blink LED
  if (currentMillis - lastBlinkMillis > BLINK_PERIOD) {
    digitalWrite(LED_BUILTIN, ledState);
    ledState = !ledState;
    lastBlinkMillis = currentMillis;
  }
  // trigger Wi-Fi network scan
  if (currentMillis - lastScanMillis > SCAN_PERIOD) {
    WiFi.scanNetworks(true);
    Serial.print("\nScan start ... ");
    lastScanMillis = currentMillis;
  }
  // print out Wi-Fi network scan result upon completion
  int n = WiFi.scanComplete();
  if (n >= 0) {
    Serial.printf("%d network(s) found\n", n);
    for (int i = 0; i < n; i++) {
      Serial.printf("%d: %s, Ch:%d (%ddBm) %s\n", i + 1, WiFi.SSID(i).c_str(),
        WiFi.channel(i), WiFi.RSSI(i), WiFi.encryptionType(i)==ENC_TYPE_NONE?"open":""");
    }
    WiFi.scanDelete();
  }
}
```



```
COM10

Scan start ... 10 network(s) found
1: TP-LINK_D0D8C8, Ch:1 (-54dBm)
2: TP-LINK_72DC, Ch:1 (-89dBm)
3: DIGI-TJ9r, Ch:1 (-91dBm)
4: apuci76, Ch:4 (-86dBm)
5: DIGI-7Pe5, Ch:5 (-89dBm)
6: DEBIWIFI, Ch:6 (-72dBm)
7: Krassoi, Ch:9 (-85dBm)
8: Csilla, Ch:10 (-84dBm)
9: CSP-LINK, Ch:11 (-68dBm)
10: DIRECT-UE-BRAVIA, Ch:11 (-86dBm)

Scan start ... 10 network(s) found
1: DIGI-TJ9r, Ch:1 (-90dBm)

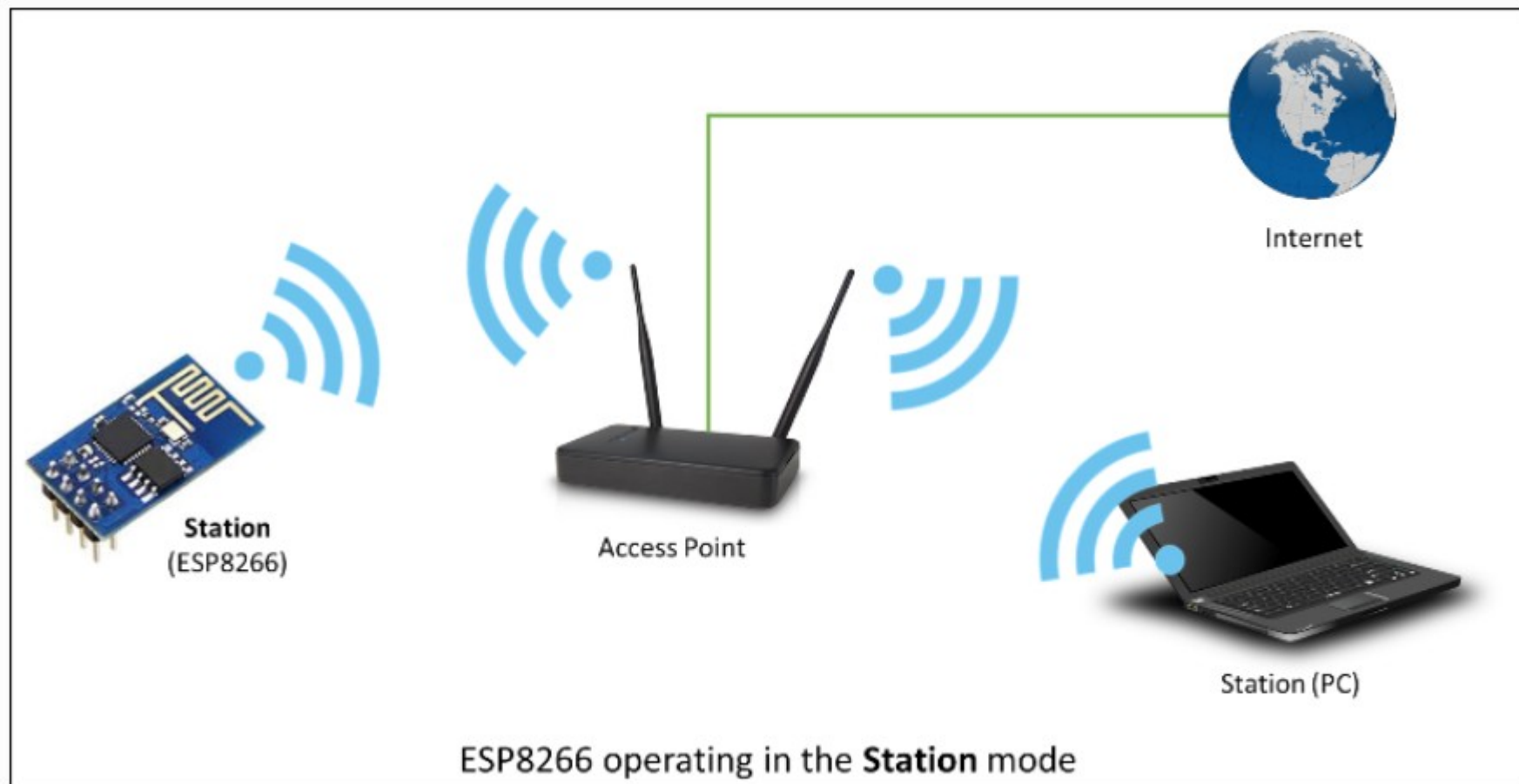
 Autoscroll  Show timestamp Both NL & CR
```

Forrás: [arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html](https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-examples.html)



# ESP8266 hálózati végpontként (station)

- Okosotthon megoldásoknál, otthoni IOT alkalmazásoknál általában egy meglévő hálózatra célszerű csatlakoztatni az ESP8266 modult
- A hálózati elérhetőséghez fel kell csatlakoznunk a hálózatra – általában egy routerre – ehhez kell az SSID/passwd páros



# Csatlakozás a hálózathoz (STA mód)

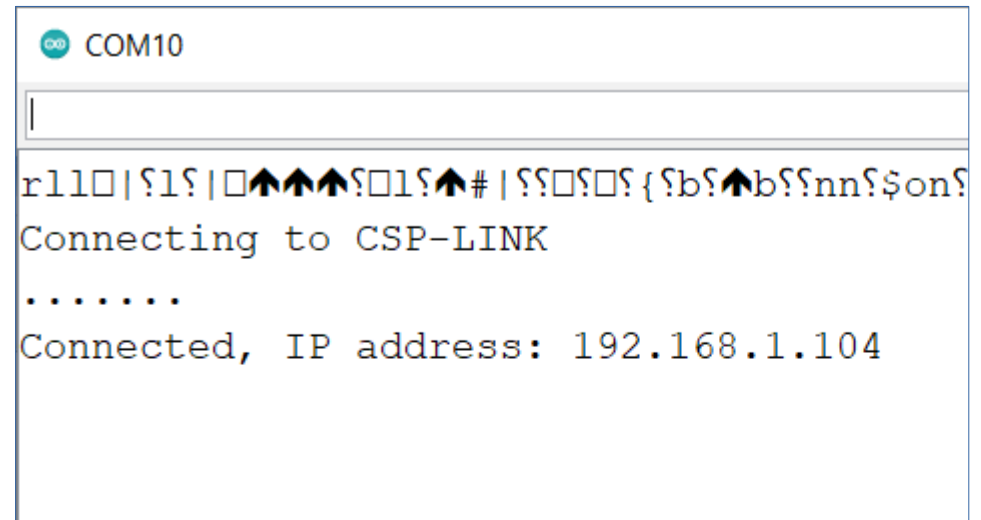
- Programjainkat többnyire azzal kezdjük, hogy az ESP8266 modult felcsatlakoztatjuk a hálózatra
- Ennek egyszerű módját mutatja az alábbi program

```
#include <ESP8266WiFi.h>
const char* ssid = "*****";
const char* password = "*****";

void setup(void) {
  Serial.begin(115200);
  Serial.println();
  Serial.printf("Connecting to %s\n", ssid);
  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();
  Serial.print("Connected, IP address: ");
  Serial.println(WiFi.localIP());
}

void loop() {}
```

Írjuk át a saját hálózati névre és jelszóra!

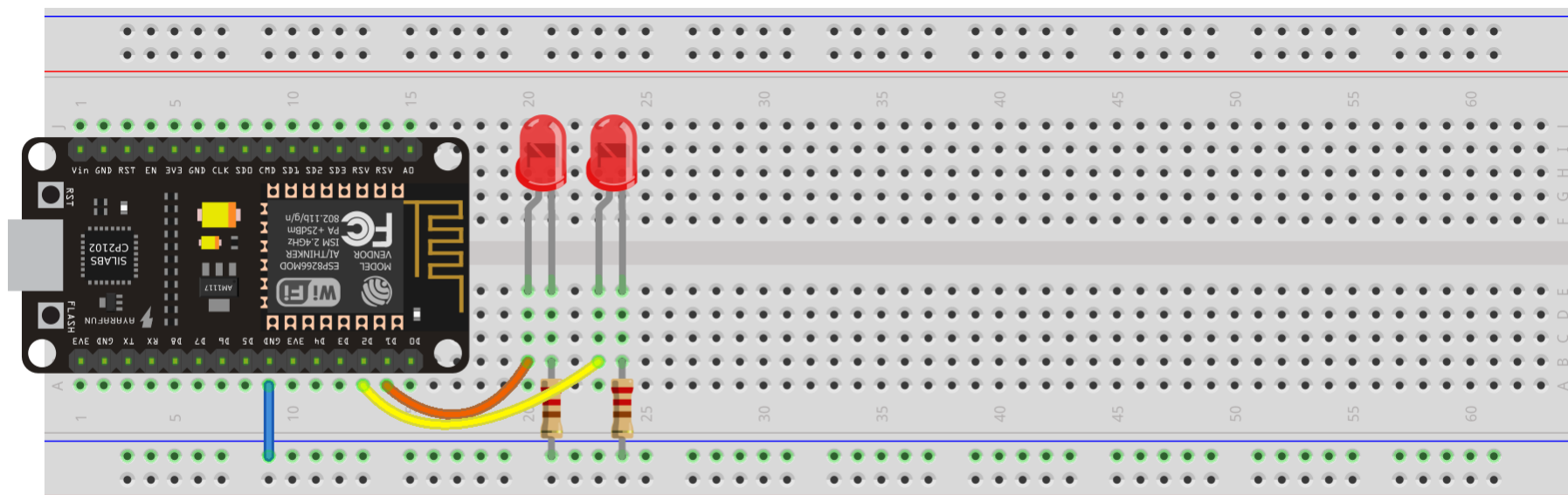


```
COM10
r11|?1?|?↑↑↑?01?↑#|??□?□?{?b?↑b??nn?on?
Connecting to CSP-LINK
.....
Connected, IP address: 192.168.1.104
```

Forrás: [arduino-esp8266.readthedocs.io/en/stable/esp8266wifi/station-class.html](https://arduino-esp8266.readthedocs.io/en/stable/esp8266wifi/station-class.html)

# ESP8266\_webserver.ino

- Kedvcsinálónak ajánlom a [Random Nerd Tutorials](#) honlapon található alábbi egyszerű és érdekes projekt, amelyben:
  - ❖ Az ESP8266 modul webszerverként funkcionál
  - ❖ Web böngészővel kapcsolódhatunk hozzá
  - ❖ Két virtuális nyomógomb segítségével két LED-et kapcsolgathatunk (gyakorlatiasabb alkalmazáshoz a LED-ek helyett relétet vagy szilárdtest relétet is kapcsolgathatunk, pl. az otthoni fényforrások távvezérlésére)

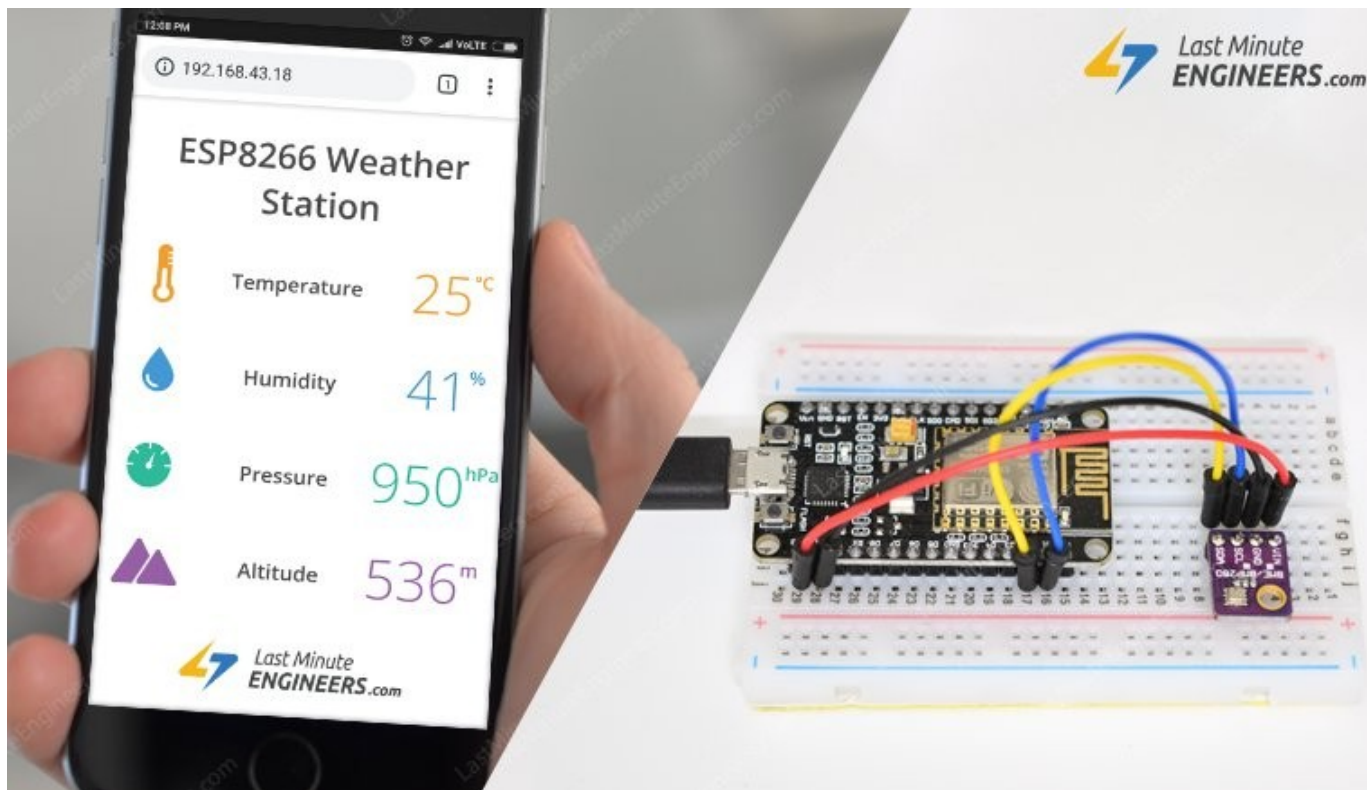


fritzing

# Időjárás állomás BME280-nal

- Egy mások kedvcsináló alkalmazás a [Last Minute Engineers](https://www.lastminuteengineers.com) honlapján található, s szintén egy webszervert valósít meg, ami egy, az I2C buszon kiolvasható **BME280** szenzorral méri és egy webböngésző csatlakozása esetén kijelzi a mért értékeket

## [Create A Simple ESP8266 Weather Station With BME280](https://www.lastminuteengineers.com)



# A BME280/BMP280 I2C szenzor használata

- A Bosch gyártmányú **BME280** szenzor nyomást, hőmérsékletet és relatív páratartalmat mér, a **BMP085**, **BMP180**, **BMP183** utódja
- A **BMP280** is hasonló, de az páratartalmat nem mér
- A szenzor **SPI** és **I2C** interfésszel rendelkezik, mi egy 5 V-tal és 3,3 V-tal egyaránt használható, **I2C** bekötésű modult használtunk
- Az **I2C** cím átforrasztással változtatható: **0x76** vagy **0x77**



Temperature: -40°C to 85°C (±1.0°C accuracy)



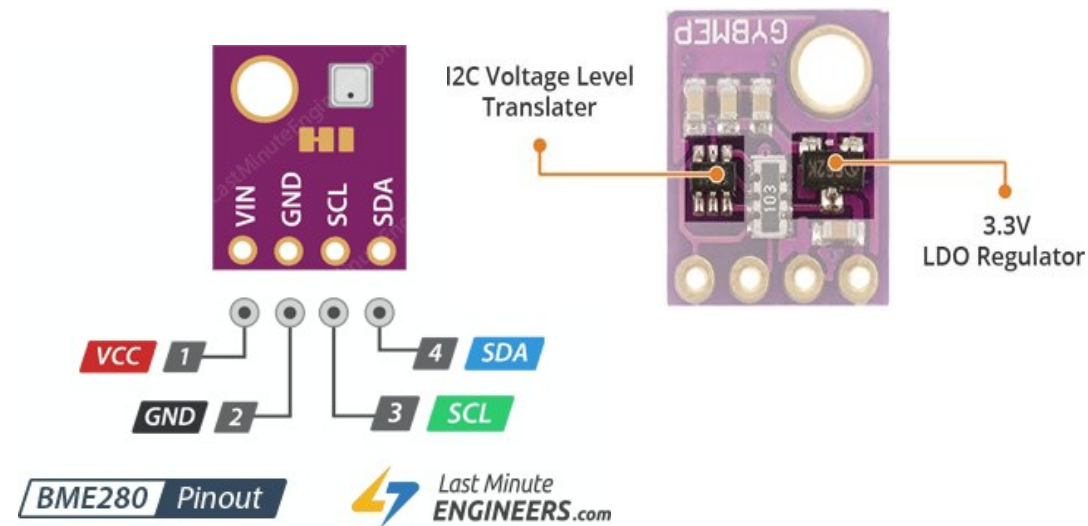
Humidity: 0 to 100% RH (±3% accuracy)



Pressure: 300hPa to 1100hPa (±1hPa accuracy)



Altitude: 0 to 30,000ft (±1 meter accuracy)



Az ábrák forrása: [lastminuteengineers.com/bme280-arduino\\_tutorial](https://lastminuteengineers.com/bme280-arduino_tutorial)

# Programkönyvtár telepítés, beállítások

- A **BME280** szenzor használatához telepíteni kell az **Adafruit\_BME280** és az **Adafruit\_sensor** programkönyvtárakat (az Arduino IDE *Tools/Manage Libraries* menüpontjában írjuk be a fenti neveket keresőszóként)

```
#include <ESP8266WebServer.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;

/*Put your SSID & Password*/
const char* ssid = "YourNetworkName"; // Enter SSID here
const char* password = "YourPassword"; //Enter Password here

ESP8266WebServer server(80);
```

# ESP8288\_WS\_bmp280.ino (részlet)

---

```
void setup() {
  Serial.begin(115200);
  delay(100);
  bme.begin(0x76);
  Serial.println("Connecting to ");
  Serial.println(ssid);
  //connect to your local wi-fi network
  WiFi.begin(ssid, password);
  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: "); Serial.println(WiFi.localIP());
  server.on("/", handle_OnConnect);
  server.onNotFound(handle_NotFound);
  server.begin();
  Serial.println("HTTP server started");
}

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

# ESP8288\_WS\_bmp280.ino (részlet)

---

```
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");
}
```

```
String SendHTML(float temperature, float humidity, float pressure, float altitude) {
  String ptr = "<!DOCTYPE html>";
  ptr += "<html>";
  ptr += "<head>";
  ptr += "<title>ESP8266 Weather Station</title>";
  ptr += "<meta name='viewport' content='width=device-width, initial-scale=1.0'>";
  ptr += "<link href='https://fonts.googleapis.com/css?family=Open+Sans:300,400,600' rel='stylesheet'>";
  ptr += "<style>";
  . . .
  . . .
  ptr += "</body>";
  ptr += "</html>";
  return ptr;
}
```



# ESP8288\_WS\_bmp280.ino: a végeredmény

