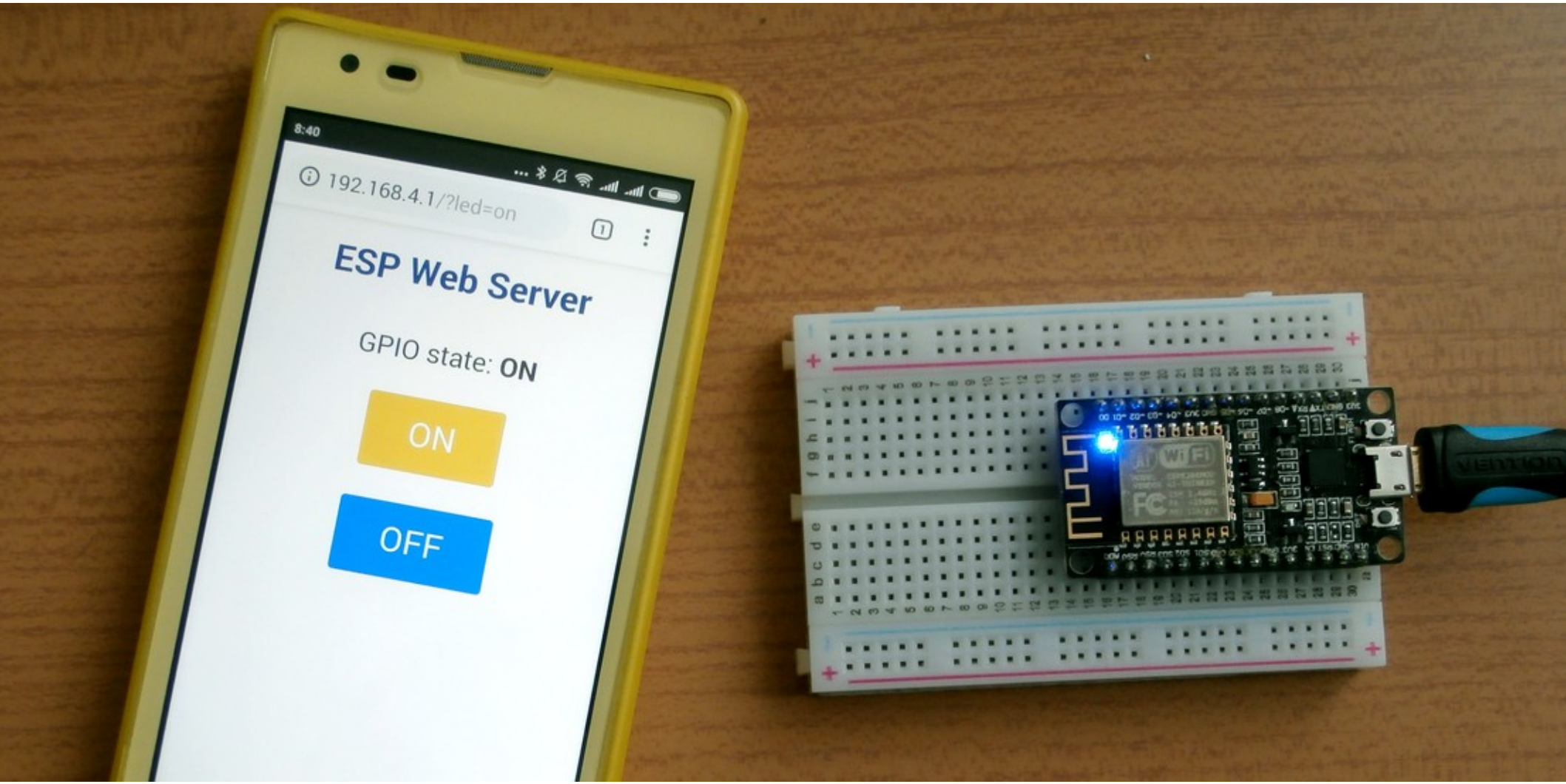
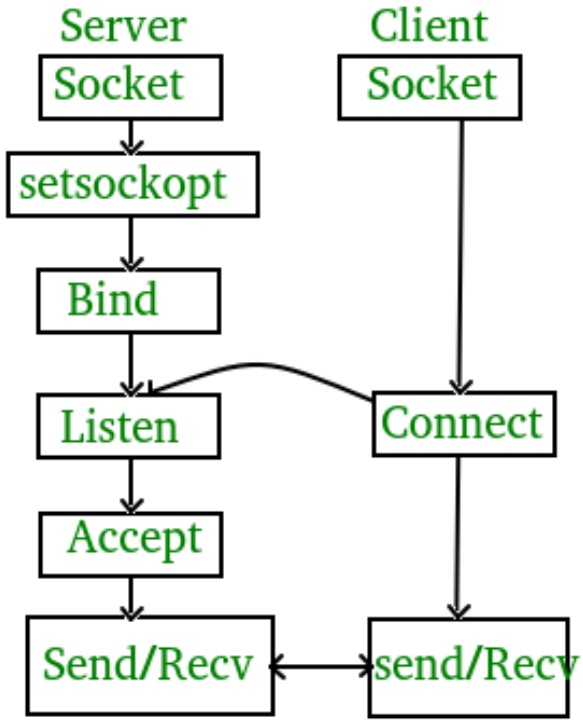
Vegyes témakörök



10. Robotvezérlés WiFi kapcsolaton keresztül

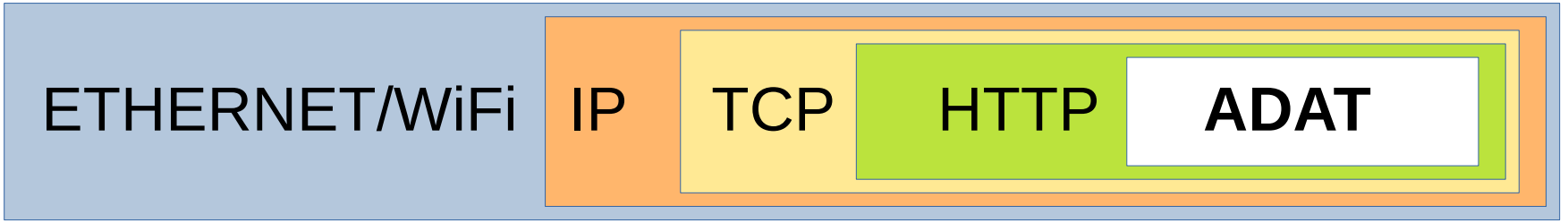
Hálózati kommunikáció socketek használatával

- A **socket** (csatlakozó) a számítógépes hálózaton két folyamat közötti kétirányú kommunikáció végpontja
- A kommunikáció kliens – szerver viszonylatban történik:
- A szerver létrehoz egy **socket**-et (amelyhez egy vagy több kliens csatlakozhat) és kapcsolódásra vár
- Ha egy kliens csatlakozik egy szerver egy **socket**-jéhez, akkor létrejön a kapcsolat és megkezdődhet a kommunikáció
- Az általunk használt IPv4 **socket**-ek egy 32 bites IP címből és egy port számmal címezhetők (**AF_INET** címformátum)
- Az általunk használt IPv4 **socket**-ek TCP (`SOCK_STREAM`) vagy UDP (`SOCK_DGRAM`) típusú üzenetcsomagokat továbbíthatnak



Milyen protokollt használjunk?

- A `SOCK_STREAM` socketek a **TCP** (Transmission Control Protocol) szintű adatforgalmat végzik. A **TCP** biztosítja, hogy az adatok sorrendtartón és hibamentesen érkezzenek meg.
- Az átvitt adatok értelmezése az alkalmazási szinten további protokoll használatát igényelheti (pl. HTTP, FTP, SMTP, NFS, Telnet stb.)
- **Adatbeágyazódás:** az átvitt adathoz minden réteg hozzáteszi, vagy lehámozza a maga fejlécét vagy keretét. Például HTML alkalmazási, TCP szállítási, IP hálózati, Ethernet/WiFi adatkapcsolati réteg
- Mi most a **HTTP** protokoll megvalósításával fogunk foglalkozni



HTTP protokoll – leegyszerűsítve

- A kliens egy `GET` paranccsal lekér egy oldalt (pl. `GET / HTTP/1.1`), illetve további információt is küldhet, a kérelmet üres sor zárja le
- A szerver egy HTTP fejlécet küld, melyet egy üres sor zár, majd elküldi a kért információt (a kért HTML oldal kódját)

```
GET / HTTP/1.1
Host: 192.168.4.1
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Linux;
Android 4.4.4; HM 1S)
Accept: text/html
Accept-Encoding: gzip, deflate
Accept-Language: hu,en
```

```
GET /?led=on HTTP/1.1
...
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Connection: close

<!DOCTYPE html>
<html>
  <head>
    <title>Az oldal címe</title>
    <!-- további fejléc-információk-->
  </head>
  <body>
    <p>első bekezdés</p>
    <p>második bekezdés</p>
  </body>
</html>
```

webserver_simple.py

- A legegyszerűbb webszerver a microPython mintapéldákból való, de több helyen megváltoztattuk!

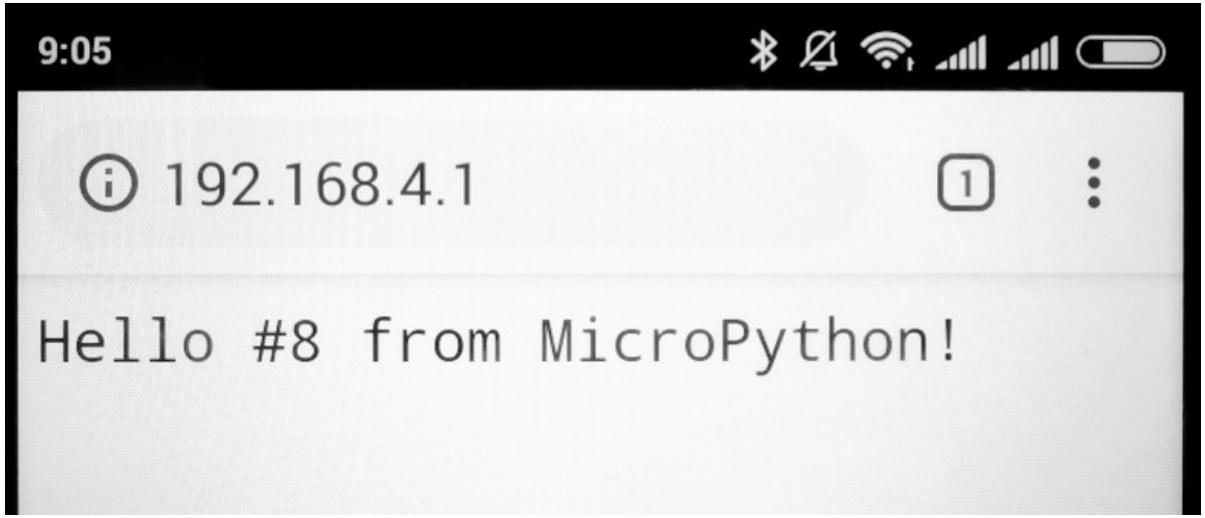
```
import network
import usocket as socket

ap = network.WLAN(network.AP_IF)  # Access point mód konfigurálása
ap.active(True)
ap.config(essid="ESP-AP")         # Alapértelmezett jelszó: micropythoN

CONTENT = b"""\
HTTP/1.0 200 OK

Hello #%d from MicroPython!
"""
# TCP socket konfigurálása és engedélyezése a HTTP portra
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))    # HTTP port
s.listen(5)         # várakozási sor hossza
print("Listening, connect your browser to http://<this_host>:80/")
```

Folytatás a következő oldalon!

webserver_simple.py

```
counter = 0
while True:
    res = s.accept()
    client_sock = res[0]
    client_addr = res[1]
    print("Client Address: ",client_addr)
    req = client_sock.recv(4096)
    print("Request:")
    print(req)
    client_sock.send(CONTENT % counter)
    client_sock.close()
    counter += 1
```

- Kliensként egy böngészővel csatlakoztunk
- Probléma: a számláló kettesével lépked



webserver_simple.py kimenete

- Mindig két GET parancs érkezik, ezért dupláz a számláló!

```
Listening, connect your browser to http://<this_host>:80/
Client Address:  ('192.168.4.2', 50038)
Request:
b'GET / HTTP/1.1\r\nHost: 192.168.4.1\r\nConnection: keep-alive\r\nCache-Control:
max-age=0\r\nSave-Data: on\r\nUpgrade-Insecure-Requests: 1\r\nUser-Agent: Mozilla/
5.0 (Linux; Android 4.4.4; HM 1S) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.110 Mobile Safari/537.36\r\nDNT: 1\r\nAccept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/
*;q=0.8\r\nAccept-Encoding: gzip, deflate\r\nAccept-Language: hu-HU,hu;q=0.9,en-
US;q=0.8,en;q=0.7\r\n\r\n'
44
Client Address:  ('192.168.4.2', 50039)
Request:
b'GET /favicon.ico HTTP/1.1\r\nHost: 192.168.4.1\r\nConnection: keep-alive\r\
nPragma: no-cache\r\nCache-Control: no-cache\r\nDNT: 1\r\nSave-Data: on\r\nUser-
Agent: Mozilla/5.0 (Linux; Android 4.4.4; HM 1S) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/70.0.3538.110 Mobile Safari/537.36\r\nAccept:
image/webp,image/apng,image/*,*/*;q=0.8\r\nReferer: http://192.168.4.1/\r\nAccept-
Encoding: gzip, deflate\r\nAccept-Language: hu-HU,hu;q=0.9,en-US;q=0.8,en;q=0.7\r\
n\r\n'
44
```

Mielőtt a hiba kijavításához fognánk, nézzünk meg egy alkalmazást!

led_switch.py

- A Random Nerd Tutorials mintapéldája egy webszerver alkalmazást mutat be a `http_server_simplistic.py` mintapélda átdolgozásával
- A program segítségével a NodeMCU kártya beépített LED-jét fogjuk kapcsolgatni (GPIO2 láb, alacsony szint esetén világít).
- Itt most csak az AP interfészt használjuk (IP = 192.168.4.1)

```
import usocket as socket                    # socket library
from machine import Pin                     # We need a pin for the LED
import network                              # network library
ap = network.WLAN(network.AP_IF)            # Access Point interface
ap.active(True)                             # Make it active
ap.config(essid="ESP-AP")                   # define new SSID
led = Pin(2, Pin.OUT)                       # LED is at GPIO2

def web_page():                             # combine text and data
  if led.value() == 1: gpio_state="OFF"
  else: gpio_state="ON"
  html = """ ... blabla, lásd külön lapon! ... """
  return html
```

Folytatás a következő oldalon!

led_switch.py

```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)  # s is a TCP socket
s.bind(('', 80))                                      # accept connection at port 80
s.listen(5)                                           # max. length of waiting que

while True:
  conn, addr = s.accept()                             # wait for a client, return a stream
  print('Got a connection from %s' % str(addr))
  request = conn.recv(4096)                           # read incoming request
  request = str(request)                              # convert it to string
  print('Content = %s' % request)
  led_on = request.find('/?led=on')                   # look for ON command
  led_off = request.find('/?led=off')                 # look for OFF command
  if led_on == 6:                                     # switch ON LED if needed
    print('LED ON');  led.value(0)
  if led_off == 6:                                    # switch OFF LED if needed
    print('LED OFF'); led.value(1)
  response = web_page()                               # get actualized HTML content
  conn.send('HTTP/1.1 200 OK\n')                      # send HTTP header first
  conn.send('Content-Type: text/html\n')
  conn.send('Connection: close\n\n')
  conn.sendall(response)                              # send HTML data
  conn.close()                                        # close connection
```

Folytatás a következő oldalon!

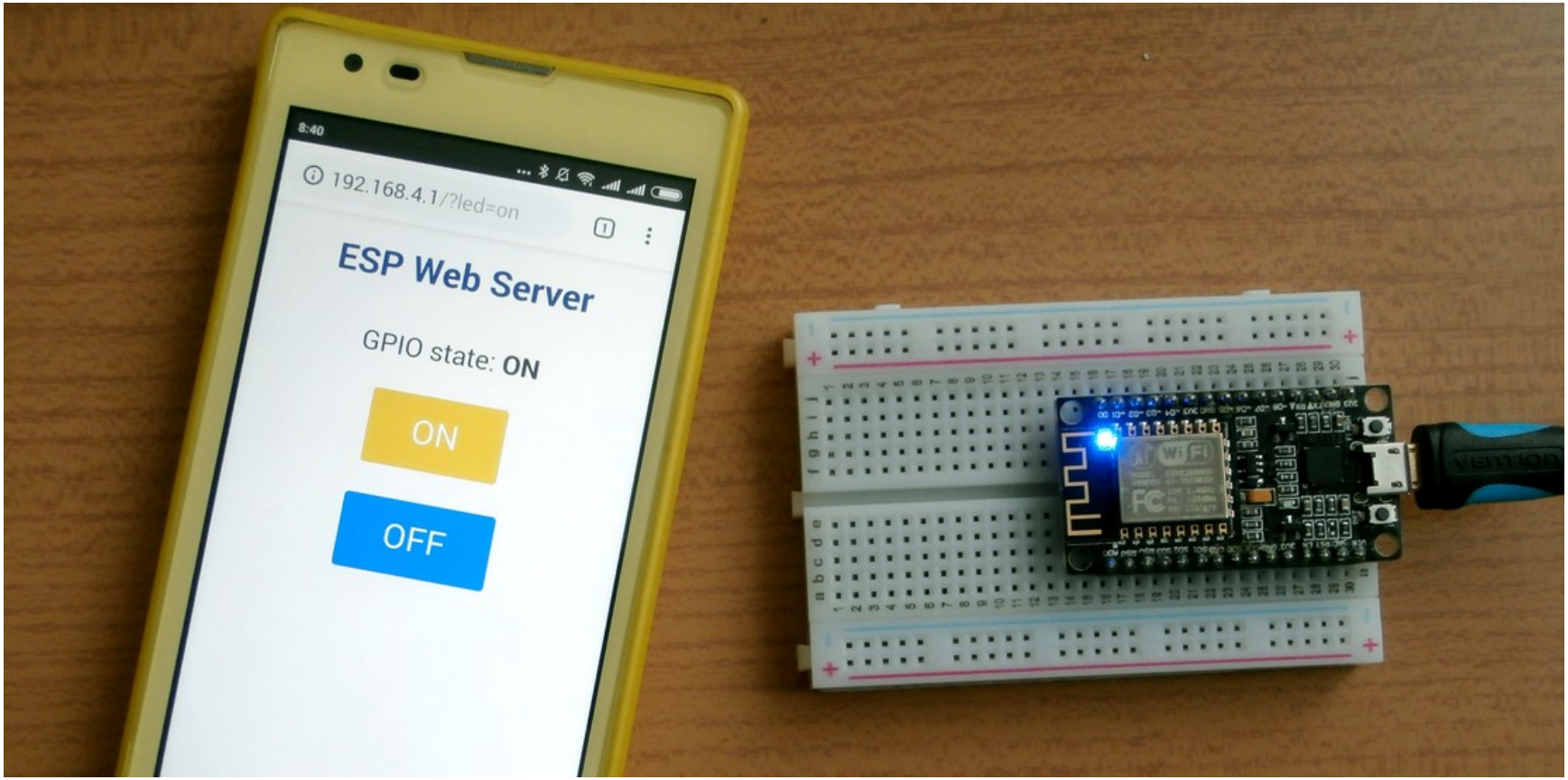
led_switch.py

- A programmal egy web böngészőn keresztül kommunikálhatunk
- A HTML kód „kibontva” az alábbi listában olvasható
- A gombokra való kattintás után a **`GET`** parancs **„`?led=on`”** vagy **„`?led=off`”** paraméterrel egészül ki, ez teszi lehetővé a vezérlést

```
"""<html>
  <head><title>ESP Web Server</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" href="data:,">
    <style>
      html {font-family: Helvetica; display:inline-block; margin: 0px auto; text-align: center;}
      h1 {color: #0F3376; padding: 2vh;}p{font-size: 1.5rem;}
      .button{display: inline-block; background-color: #e7bd3b; border: none; border-radius: 4px;
        color:  white; padding: 16px 40px; text-decoration: none; font-size: 30px; margin: 2px;
        cursor: pointer;}
      .button2{background-color: #4286f4;}
    </style>
  </head>
  <body>
    <h1>ESP Web Server</h1>
    <p>GPIO state: <strong>""" + gpio_state + """</strong></p>
    <p><a href="/?led=on"><button class="button">ON</button></a></p>
    <p><a href="/?led=off"><button class="button button2">OFF</button></a></p>
  </body>
</html>"""
```

led_switch.py futtatása

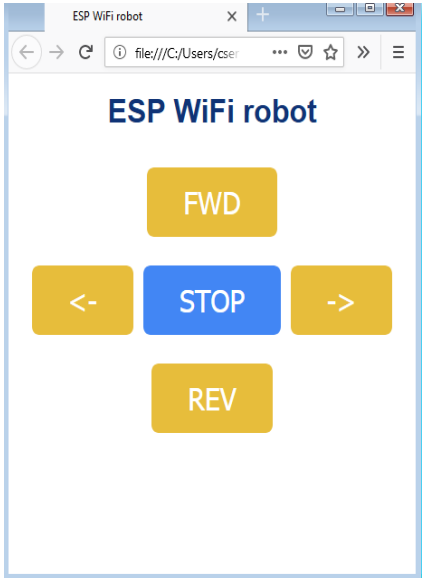
- Indítsuk el a `led_switch.py` programot a mikrovezérlőn!
- Csatlakozzunk egy WiFi eszközzel (pl. mobiltelefon) az alábbi paraméterekkel: `SSID = ESP-AP, Password = micropythoN`
- Csatlakozzunk egy böngészővel a `192.168.4.1` címre!



Továbbfejlesztési javaslat

- Robot irányításához például így írhatjuk át a HTML lapot (nyilván a firmware-t is át kell írni hozzá)

```
<html>
<head>
  <title>ESP WiFi robot</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <style>
    html {font-family:Helvetica;display:inline-block;margin:0px auto;text-align:center;}
    h1 {color: #0F3376; padding: 2vh;}p{font-size: 1.5rem;}
    .button {display: inline-block; background-color: #e7bd3b; border: none;
      border-radius: 8px; color: white; padding: 16px 40px; text-decoration: none;
      font-size: 30px; margin: 2px; cursor: pointer;}
    .button2 {background-color: #4286f4;}
  </style>
</head>
<body>
  <h1>ESP WiFi robot</h1>
  <p><a href="/?btn=F"><button class="button ">FWD</button></a></p>
  <p><a href="/?btn=L"><button class="button"><-</button></a>
     <a href="/?btn=S"><button class="button button2">STOP</button></a>
     <a href="/?btn=R"><button class="button">-></button></a></p>
  <p><a href="/?btn=B"><button class="button">REV</button></a></p>
</body>
</html>
```



http_server.py

- A helyi hálózathoz a helyi routeren a STA_IF interfész segítségével tudunk kapcsolódni
- A fölösleges `favicon.ico` lekérésekhez egy hibaüzenetet definiálunk

```
import usocket as socket
import network

ssid = 'XXXXXX'
password = 'xxxxxx'
station = network.WLAN(network.STA_IF)
station.active(True)
station.connect(ssid, password)
while station.isconnected() == False:
  pass
print('Connection successful')
print(station.ifconfig())

ERR406 = "HTTP/1.0 406 Not Acceptable\r\n"
```

A programnak itt csak a releváns sorait mutatjuk be!

Folytatás a következő oldalon!

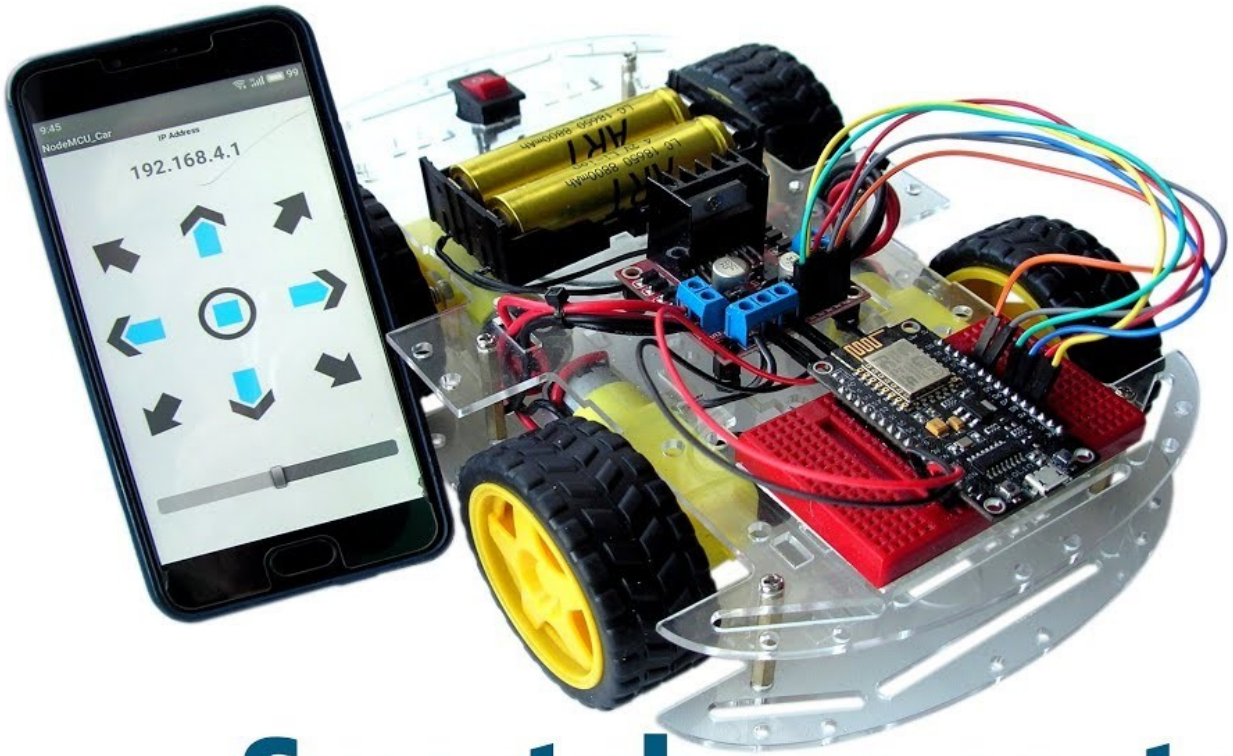
http_server.py

- Bemutatjuk a soronkénti kiolvasást és a `favicon.ico` lekérés kiszűrését a dupla számlálóléptetés elkerülésére

```
counter = 0
while True:
    client_sock, client_addr = s.accept()
    req = client_sock.readline()              # Soronként olvasunk
    print(req)
    mystr = req.decode('ASCII')               # text → string átalakítás
    while True:                               # beolvassuk többi sort is
        h = client_sock.readline()
        if h == b"" or h == b"\r\n":
            break
        print(h)
    if mystr.find("/favicon") < 0:            # ha nem favicon.ico lekérés
      client_sock.write(CONTENT % counter) # formázott kiíratás
      counter += 1
    else:
      client_sock.write(ERR406)               # favicon.ico helyett hibaüzenet
    client_sock.close()                       # a kapcsolat lezárása
```

Vezérlés Android alkalmazással

- Andriy Baranov **NodeMCU_Car** projektje WiFi vezérlésű robot. A firmware Arduino IDE alá készült, de a vezérlő Android alkalmazás HTTP protokolt használ, így mi is használhatjuk!

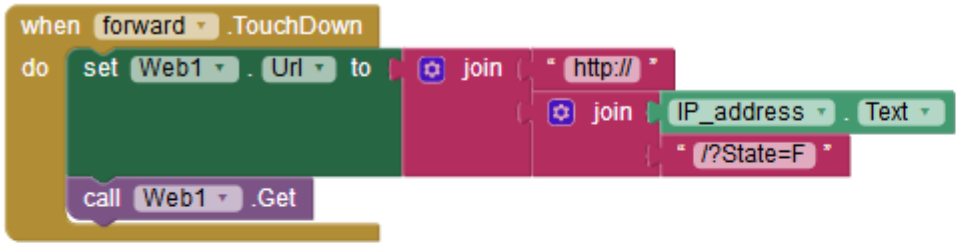


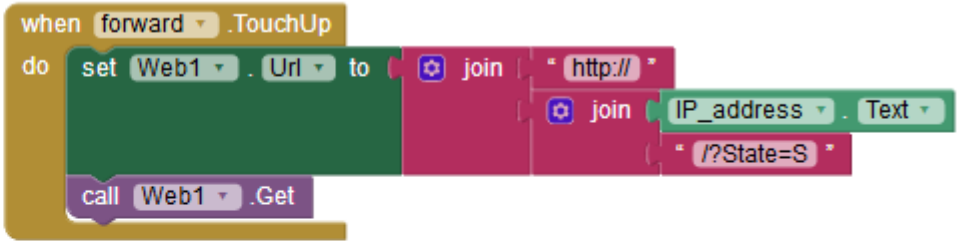
Smartphone controlled

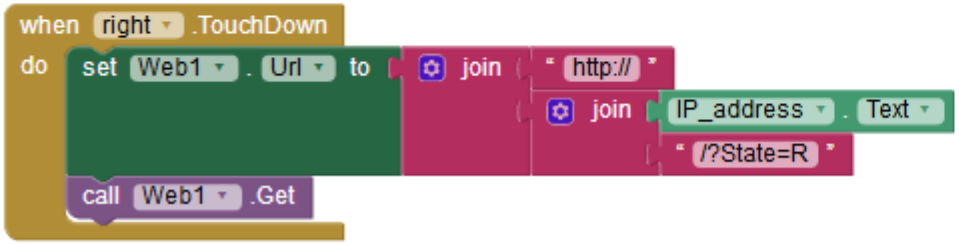
NodeMCU car

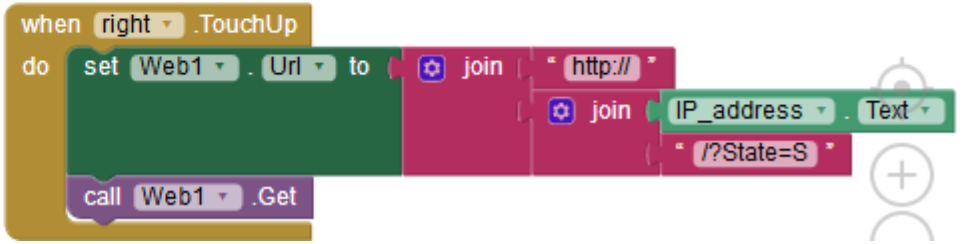
Vezérlés Android alkalmazással

- Az Android alkalmazás a **MIT AppInventor** online fejlesztői környezetében készült, így telepítés nélkül mi is használhatjuk, ha például módosítani akarjuk (link: appinventor.mit.edu/explore/)

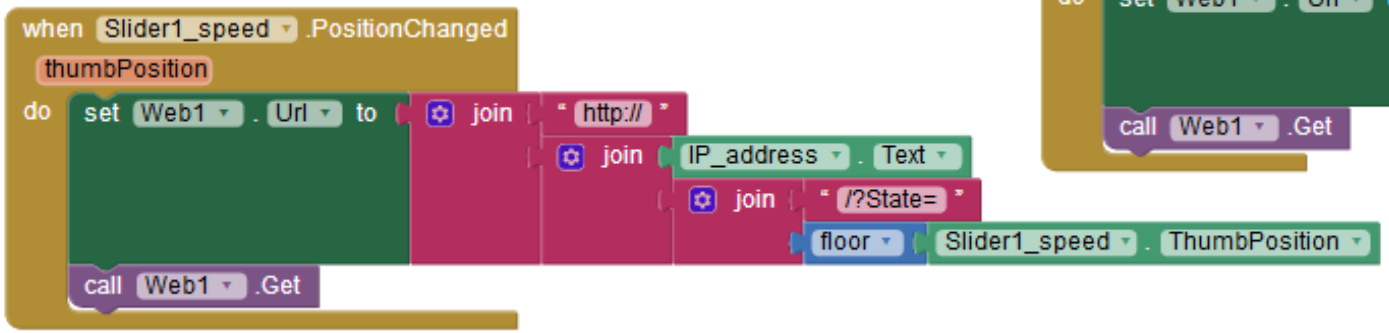


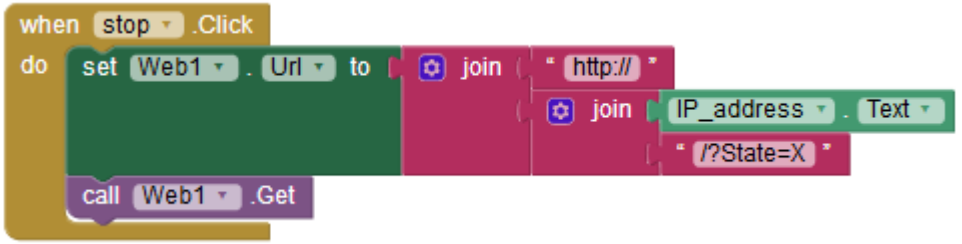






… és így tovább





nodemcu_car.py

- A kommunikációt kipróbáló programnak itt csak a beérkező lekéréseket kiszolgáló részét mutatjuk be, a program eleje megegyezik a korábbiakkal

```
while True:
    client_sock, client_addr = s.accept()  # kapcsolódásra várunk
    print("Client address:", client_addr)
    req = str(client_sock.readline())      # az első sor a lekérés (GET)
    while True:
       h = client_sock.readline()          # beolvassuk a többi sort is
       if h == b"" or h == b"\r\n":
           break
       print(h)
    print("Request: ",req)
    if req.find("/?State=") > 0:           # >0 ha volt State paraméter
      print("Command: ",req[14])           # a dekódolt parancs
    w = client_sock.write("HTTP/1.0 200 OK\n")
    w = client_sock.write('Content-Type: text/html\n')
    w = client_sock.write('Connection: close\n  \n\n')
    client_sock.close()
```

Tapasztalatok

- A `NodeMCU_Car.apk` alkalmazás is jól használható, bár a sebességállító csúszka mozgatása esetén túlterheli a webszervert
  - **Javaslat:** semmi sem indokolja a csúszkaállás mozgatás közbeni kiküldését ezért a sebességet is a gombok lenyomásakor kellene kiküldeni
- Az **ESP8266 MicroPython** adaptáció nem alkalmas többszálú futtatásra, ezért akadályozó tényező hogy a `socket.accept()` blokkoló várakozást jelent, közben más feladat nem végezhető.
- **Megoldási lehetőségek:**
  - Periodikus, rövid idejű várakozás `socket.settimeout()` beállítással
  - A `select` modul `select.poll()` osztályának használata a socket állapotának lekérdezésére, szintén timeout-tal
- A következő oldalakon bemutatunk egy-egy egyszerű példát a fentiek megvalósítására

timeout.py

- Ez a program a **`ledswitch.py`** program módosított változata a **`socket.settimeout()`** használatának bemutatására

```
import socket               # Itt most nem használhatjuk a usocket modult!
import select

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.settimeout(0.5)           # timeout értéke itt 0.5 sec
s.listen(1)
...
while True:
  try:
      conn, addr = s.accept()
      # --- lekérés beolvasása, feldolgozása és válasz küldése ---
      conn.close()
  except OSError as er:
      # print(er)
      # --- Egyéb feladat végrehajtása ---
```

polling.py

- Ez a program a **`ledswitch.py`** program módosított változata a **`select.poll()`** használatának bemutatására

```
import usocket as socket
import uselect as select
...
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(('', 80))
s.listen(1)
p = select.poll()
p.register(s,select.POLLIN)

while True:
  fvdevent = p.poll(1000)      # timeout értéke milliszekundumokban
  if fvdevent :
      conn, addr = s.accept()
      # --- lekérés beolvasása, feldolgozása és válasz küldése ---
  else:
      # --- Egyéb feladat végrehajtása ---
```

NodeMCU kártya MicroPythonnal