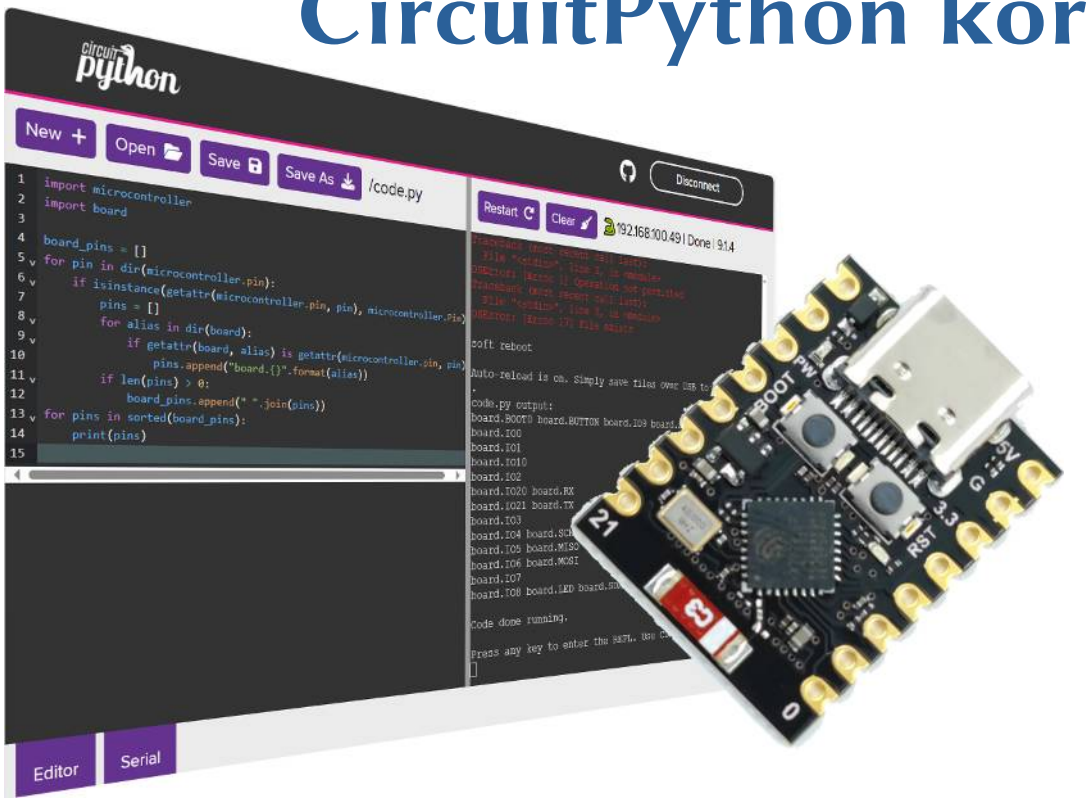
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



3. Analóg jelek mérése

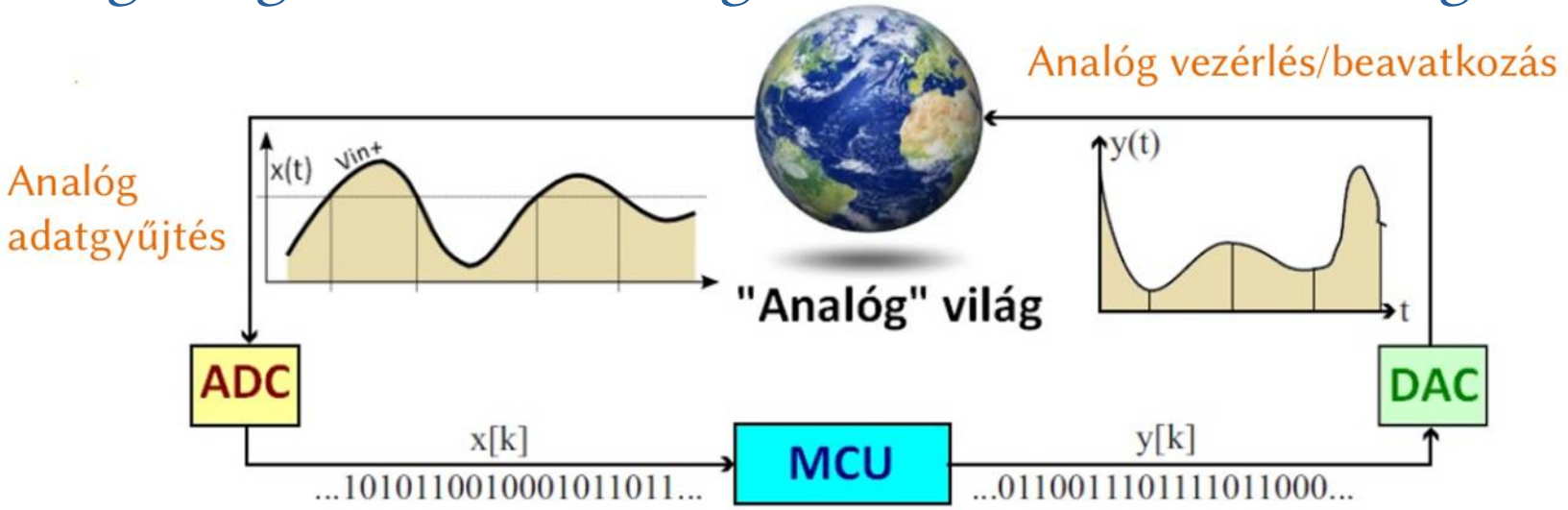
Felhasznált és ajánlott irodalom

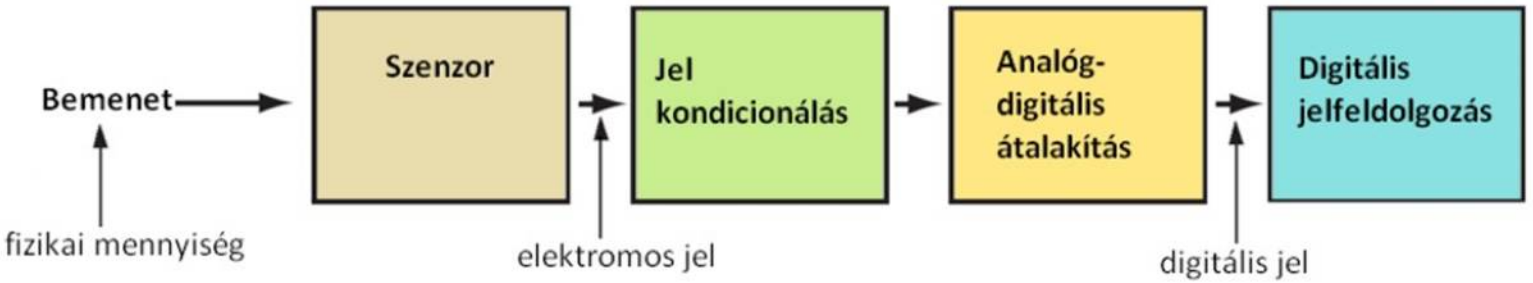
- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
  - P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- **CircuitPython:**
  - Adafruit: https://circuitpython.org/downloads
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

Analóg jelfeldolgozás

- Analóg világban élünk, de digitális mikrovezérlővel dolgozunk…



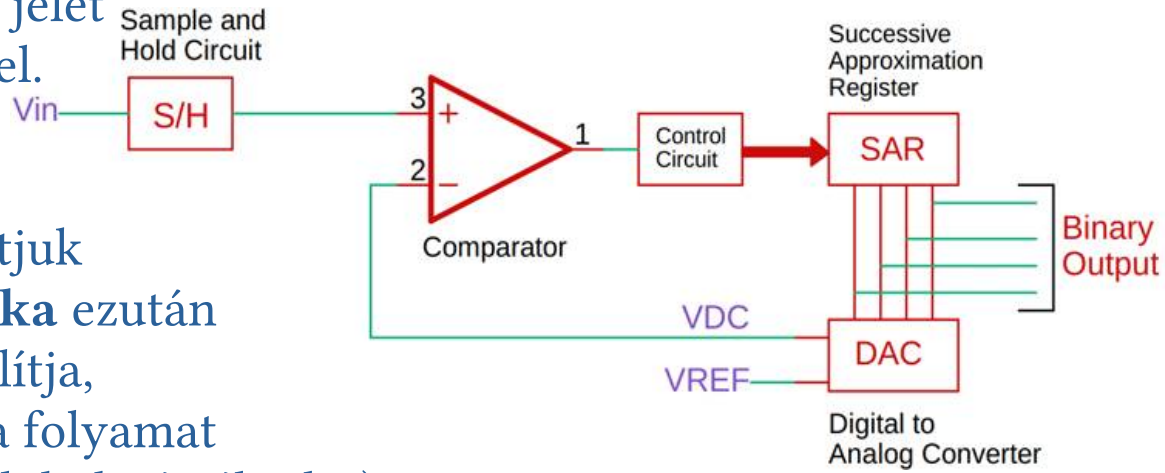


Az analóg adatgyűjtő ág elemei

- **Szenzor**
  - Olyan eszköz, ami fizikai mennyiségeket, pl. hőmérsékletet, fényt, nyomást vagy mozgást érzékel és alakít át elektromos jelekké (feszültséggé / árammá)
- **Jelkondicionálás**
  - A szenzorok által generált nyers jeleket olyan formába alakítják, amely alkalmas a további feldolgozásra és elemzésre. Ez magában foglalhatja a jelek erősítését, szűrését, zajcsökkentését és az ADC bemeneti tartományához és impedanciájához történő transzformálását
- **Analóg-Digitális Átalakító (ADC)**
  - Az analóg jelek folyamatosak és végtelen számú értéket vehetnek fel, míg a digitális jelek diszkrét értékekből állnak. Az ADC a bejövő analóg jelet digitális kóddá alakítja: meghatározza, hogy melyik sorszámú intervallumba esik a bejövő jel nagysága

Fokozatos megközelítésű ADC működése

- A fokozatos megközelítésű **Analóg-Digitális Átalakító** (**SAR ADC**) bináris keresési algoritmust alkalmaz az analóg jelek digitális formába történő átalakítására
- Az analóg bemeneti jelet egy **mintavétel és tartás** áramkör (Sample and Hold Circuit) rögzíti, hogy a bemeneti jel stabil maradjon a konverzió ideje alatt
- **Kezdeti Beállítás:** A **SAR ADC** egy N-bites regisztert használ, amelyet először középskálára állítanak (például 100...00, ahol az MSB 1). Ez a beállítás a **DAC** (Digitális-Analóg Átalakító) kimenetét a referenciafeszültség felére állítja
- **Összehasonlítás:** Az analóg bemeneti jelet összehasonlítják a **DAC** kimeneti jelével. Ha a bemenő jel nagyobb, a **SAR regiszter** legmagasabb helyiértékű bitje (MSB) 1 marad, különben 0-ra állítjuk
- **Bináris Keresés:** A **SAR vezérlő logika** ezután a következő bitre lép, és azt magasra állítja, majd újabb összehasonlítást végez. Ez a folyamat folytatódik egészen az LSB-ig (legkisebb helyiértékű bit)



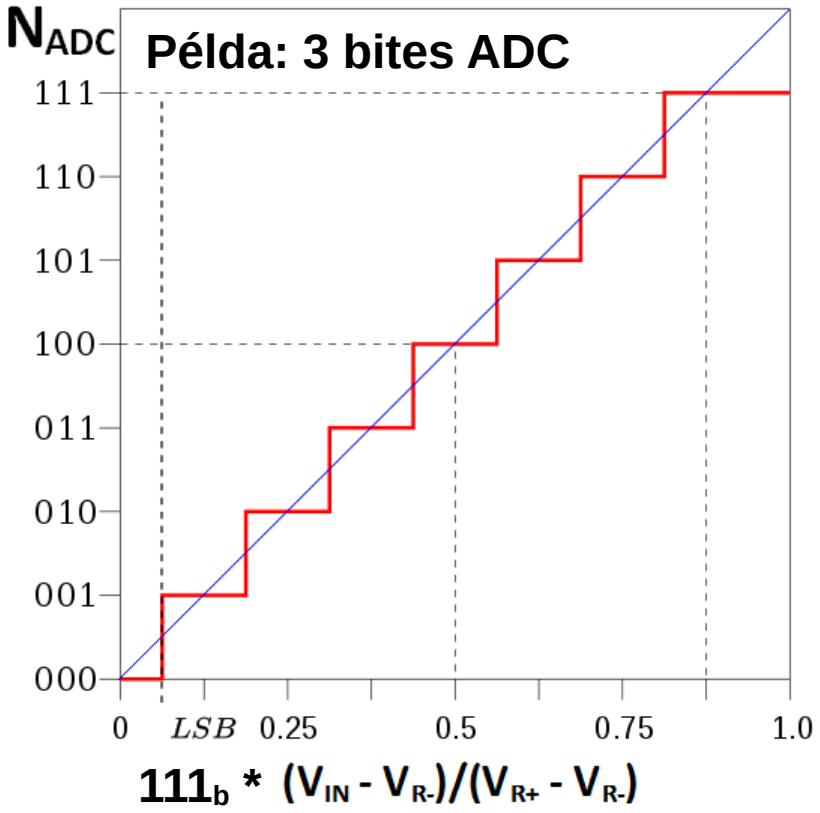
Ábra forrása: hu.amen-technologies-com

ADC – Analóg-digitális átalakító

- Az **ADC** feladata az, hogy diszkrét kódokká alakítsa a bejövő analóg jelet
- A konverzió digitális értéke (N_{ADC}):
  - Végkitérés: N_{ADC}= 4095, ha a felbontás 12 bit bemenő jel ≥ V_{R+} −1.5*LSB
  - Nulla: N_{ADC} = 0, ha a bemenő jel ≤ V_{R-} + 0.5 LSB

V_{R+} és V_{R-} a referencia-forrás két sarka

- Közbeeső értékekre:
  $N_{ADC} = 4096 * (V_{IN} - V_{R-})/(V_{R+} - V_{R-})$
- A fenti képletből V_{IN}-t kifejezve ezt kapjuk:
  $V_{IN} = (V_{R+} - V_{R-}) * N_{ADC}/4096 + V_{R-}$
- V_{R-} általában = 0



Analóg bemenetek

- Két **ADC** van, de **ADC2** csak a **WiFi** letiltott állapotában használható

Az ADC kezelése CircuitPythonban

- Az **Espressif** mikrovezérlőinek felbontása és mérési tartománya különböző, de a **CircuitPython** ADC-t kezelő függvényei mindet úgy transzformálják, mintha 16 bites felbontásúak és 0 – 3.3 V mérési tartományúak lennének
- Az **ESP32-C3** ADC-je például 12 bites, tehát fizikailag 0 – 4095 közötti számot ad ki, de a **CircuitPython analogio** könyvtári modul függvénye már 0 – 65 535 közötti számot ad a konverzió végeredményéül

Az **analogio** könyvtári modul két osztályt definiál:

- *class* **analogio.AnalogIn**(*pin: board.Pin*) – az ADC kezelése (*boad.Pin* lesz a bemenet)

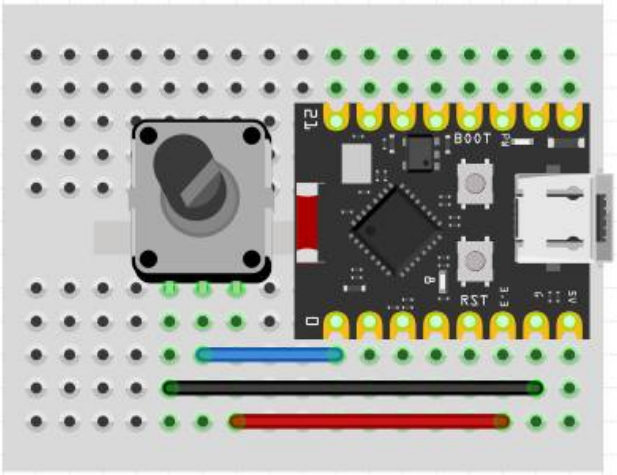
  függvények:

  deinit() – megszünteti a Pin hozzárendelését az ADC-hez
  value – a mérést indít és egy 0 – 65 535 közötti számot ad vissza
  reference_voltage – a névleges referencia feszültséget adja vissza

- class **analogio.AnalogOut**(*pin: board.Pin*) – analóg kimenet (DAC) kezelése, ami esetünkben nem használható, mert az **ESP32-C3** mikrovezérlő nem rendelkezik ilyen perifériával

adc_proba.py

- Az alábbi program másodpercenként megméri a **GPIO0 (A0)** bemenetre kötött feszültséget és kiírja a terminálon
- Az ábrán látható kapcsolásban egy potméter csúszkáját kötöttük az **A0** bemenetre




```
import time
import board
import analogio

A0 = analogio.AnalogIn(board.IO0)

while True:
    adc_raw = A0.value
    adc_volt = round(adc_raw * A0.reference_voltage/65536, 3)
    print("Raw data:", adc_raw, "Voltage:", adc_volt, "V")
    time.sleep(1)
```

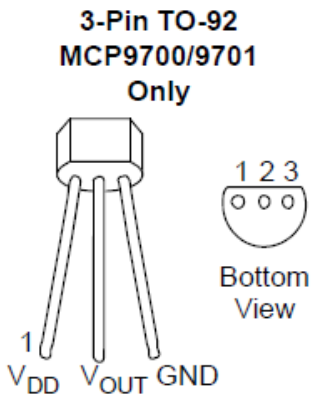
```
Raw data: 23115 Voltage: 1.164 V
Raw data: 22996 Voltage: 1.158 V
Raw data: 22540 Voltage: 1.135 V
Raw data: 26591 Voltage: 1.339 V
Raw data: 26591 Voltage: 1.339 V
Raw data: 28239 Voltage: 1.422 V
Raw data: 28180 Voltage: 1.419 V
Raw data: 29828 Voltage: 1.502 V
Raw data: 32072 Voltage: 1.615 V
Raw data: 32946 Voltage: 1.659 V
Raw data: 35706 Voltage: 1.798 V
Raw data: 38566 Voltage: 1.942 V
Raw data: 38824 Voltage: 1.955 V
Raw data: 39181 Voltage: 1.973 V
Raw data: 39042 Voltage: 1.966 V
```

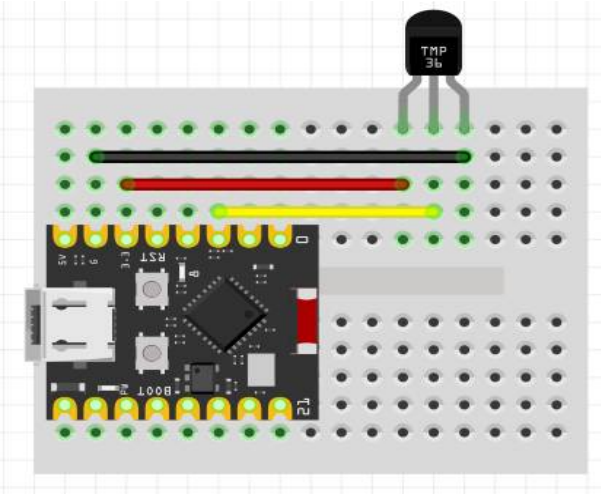
analog_thermometer.py – analóg hőmérő használata

Microchip **MCP9700**

- VDD = 2,5 – 5,5 V
- Mérési tart.: –40 – 150 °C
- Érzékenység: 10 mV / °C
- Nullapont: 500 mV @ 0 °C

Kössük a hőmérő kimenetét az ESP32 GPIO2 bemenetére!





analog_thermometer.py

```
import time
import board
from analogio import AnalogIn
A0 = AnalogIn(board.IO2)            # Példányosítjuk az AnalogIn osztályt éa IO2-höz rendeljük

while True:
    adc_raw = A0.value              # Egy mérést végzünk
    adc_volt = round(adc_raw * A0.reference_voltage/65536, 3)
    tempC = round((adc_volt - 0.5) * 100, 1)
    print("Raw data:", adc_raw, "Voltage:", adc_volt, "V  Temperature:", tempC, "C")
    time.sleep(5)
```

analog_thermometer.py – futási eredmény

- Az ábrán az első oszlopban a nyers adatok láthatók
- A második oszlop a voltokban kifejezett feszültség értékét jelenti

$$adc\,volt = raw\,adc \cdot v_{ref} / 65536$$

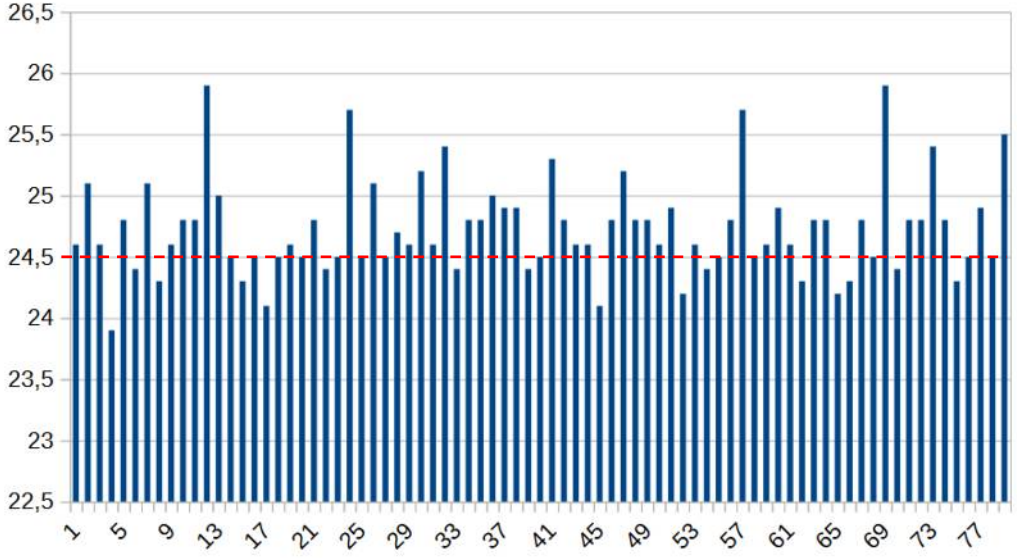
- Az utolsó oszlopban a hőmérséklet számított értéke látható

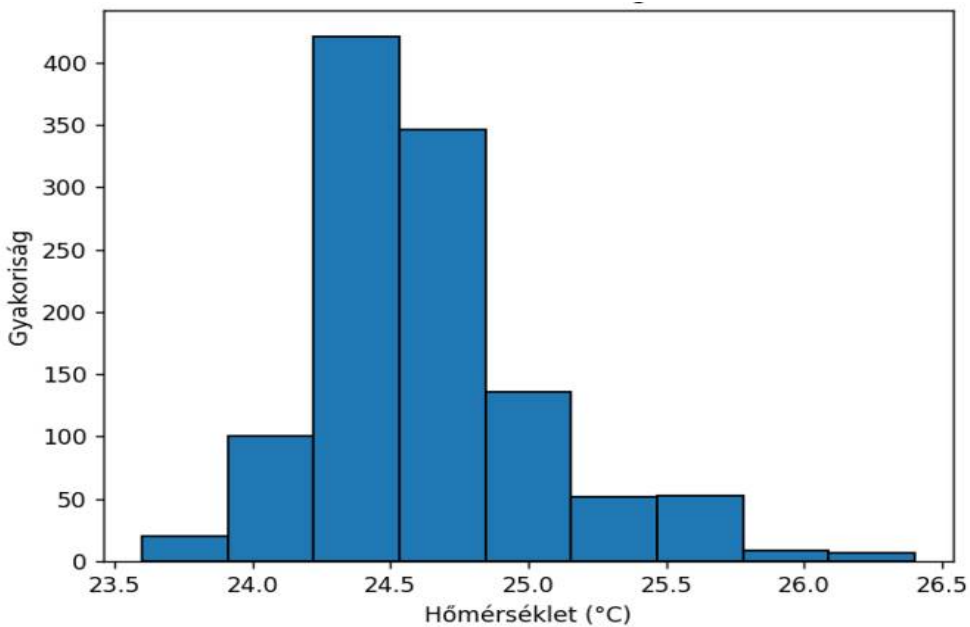
$$tempC = (adc\,volt - 0.5\,V) \cdot 100\,^{o}C/V$$

```
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14795 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14795 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14795 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14596 Voltage: 0.735 V  Temperature: 23.5 C
Raw data: 15192 Voltage: 0.765 V  Temperature: 26.5 C
Raw data: 14934 Voltage: 0.752 V  Temperature: 25.2 C
Raw data: 14914 Voltage: 0.751 V  Temperature: 25.1 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14914 Voltage: 0.751 V  Temperature: 25.1 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14814 Voltage: 0.746 V  Temperature: 24.6 C
Raw data: 14934 Voltage: 0.752 V  Temperature: 25.2 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 15073 Voltage: 0.759 V  Temperature: 25.9 C
Raw data: 14914 Voltage: 0.751 V  Temperature: 25.1 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14775 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14834 Voltage: 0.747 V  Temperature: 24.7 C
```

Miért szórnak az adatok?

- Amikor méréseket végzünk, az adatok ingadozhatnak, mert különböző tényezők (például elektromos zavarjelek) befolyásolják őket. Ezek az ingadozások gyakran véletlenszerűek, és normál eloszlásúak, ami azt jelenti, hogy a legtöbb mérés az átlag körül helyezkedik el, és csak néhány mérés tér el nagyon az átlagtól
- Ha sok mérést végzünk, eltérések pozitív és negatív irányban is előfordulnak, így az átlag közelebb lesz a valós értékhez. **Az átlagolás** tehát segít megszabadulni a véletlenszerű hibáktól, és pontosabb eredményt kapunk.





averaging_thermometer.py – Hőmérés átlagolással

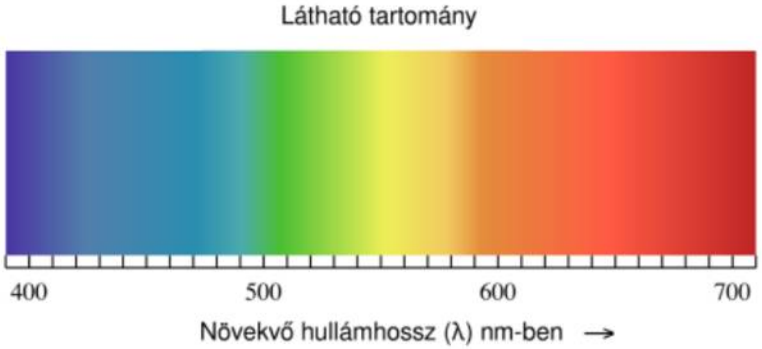
- A `get_avg(ch, n)` függvény *n* darab mérés eredményét átlagolja, n = 1000 esetén az ingadozások alig látható mértékűre csökkentek

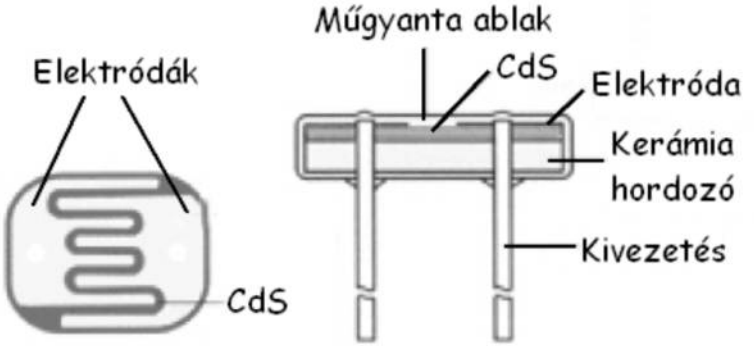
```
import time
import board
from analogio import AnalogIn

A0 = AnalogIn(board.IO2)

def get_avg(ad_ch, n):
    sum = 0
    for _ in range(n):
        sum += ad_ch.value
    return int(sum/n + 0.5)

while True:
    adc_raw = get_avg(A0, 1000)
    adc_volt = round(adc_raw *
      A0.reference_voltage/65536, 3)
    tempC = round((adc_volt - 0.5) * 100, 1)
    print("Raw data:", adc_raw, "Voltage:",
      adc_volt, "V  Temperature:", tempC, "C")
    time.sleep(5)
```

```
code.py output:
Raw data: 14786 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14783 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14786 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14786 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14785 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14789 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14789 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14791 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14786 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14790 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14790 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14788 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14792 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14785 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14785 Voltage: 0.744 V  Temperature: 24.4 C
Raw data: 14787 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14786 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14789 Voltage: 0.745 V  Temperature: 24.5 C
Raw data: 14789 Voltage: 0.745 V  Temperature: 24.5 C
```

CdS fényérzékeny ellenállás

A GL55 típusú kadmiumszulfid (CdS) anyagú ellenállások vezetőképessége a fény hatására növekszik.
A spektrális érzékenység maximuma 540 nm (sárgászöld).





Típus	Sötét-ellenállás	Ellenállás @ 10 lx	γ	Válaszidő
GL5528	1 MΩ	10 – 20 kΩ	0.6	20 – 30 ms

$$\gamma = lg\left(\frac{R\,10}{R\,100}\right)$$

R10 és R100 a 10 és 100 lux-nál mért ellenállás értékek.

LDR_test.py - fénymérés

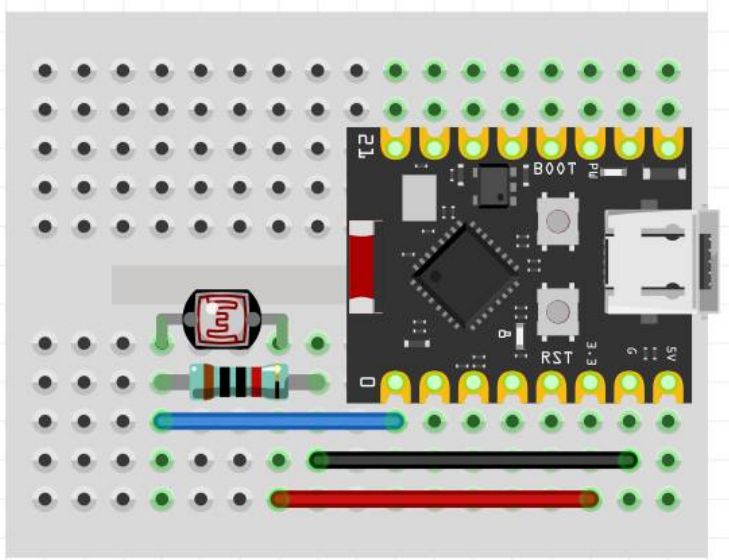
- Az LDR (fényérzékeny ellenállás) és a 10 kΩ-os ellenállás feszültség-osztót alkot. Az osztó feszültségét az ADC segítségével megmérve, az LDR ellenállása meghatározható

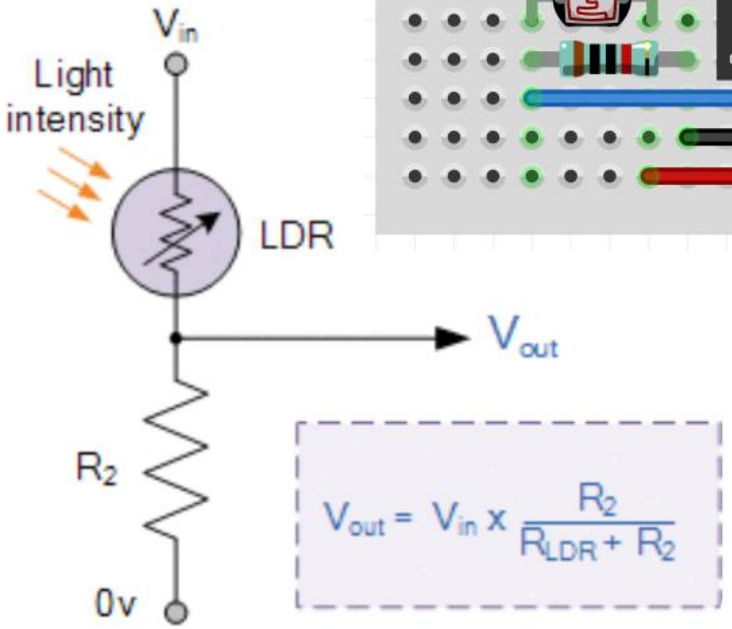
$$V_{OUT} = V_{REF} \cdot \frac{R_2}{R_{LDR} + R_2}$$

$$V_{OUT} = N_{ADC} \cdot V_{REF} / 65536$$

$$\frac{N_{ADC}}{65536} = \frac{R_2}{R_{LDR} + R_2}$$

$$R_{LDR} = \frac{R_2 \cdot 65536}{N_{ADC}} - R_2$$





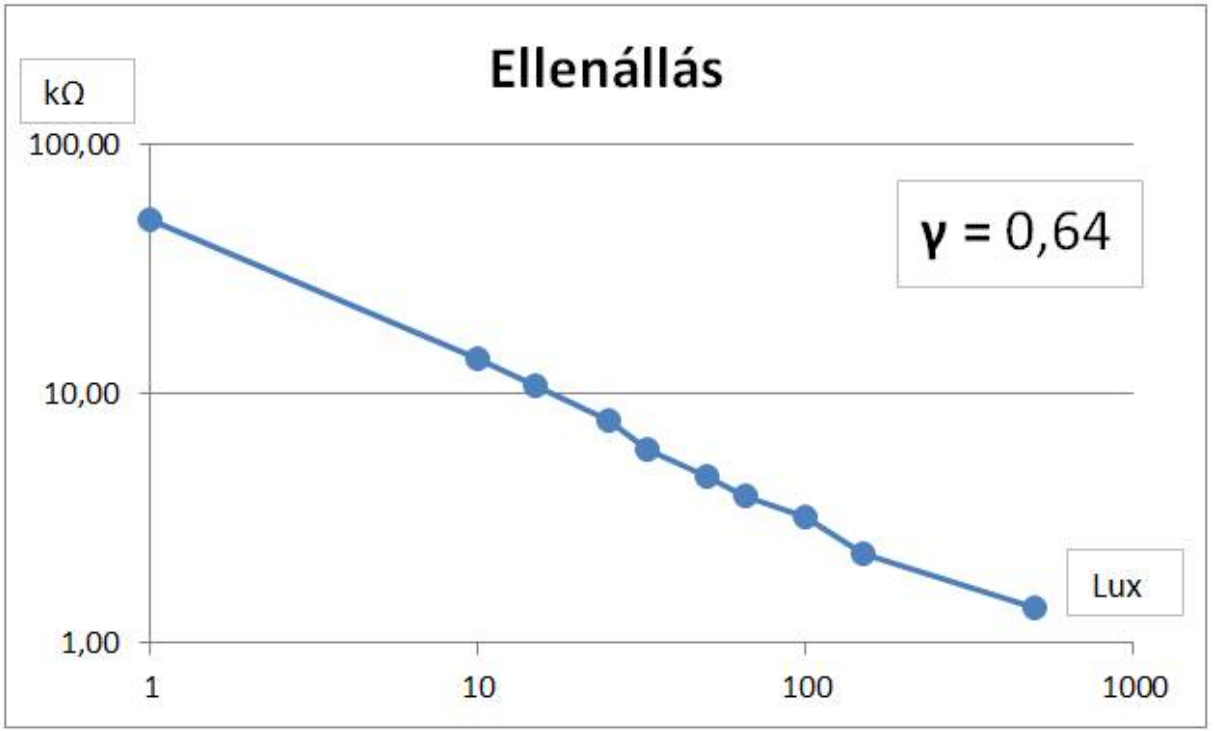
LDR_test.py - fénymérés

```
import time
import board
import analogio

adc = analogio.AnalogIn(board.IO0)  # GPIO0 (A0) bemenet
known_resistor = 10.0               # Ismert értékű ellenállás

def calculate_resistance(adc_value):
    # LDR ellenállásának kiszámítása
    return known_resistor * (65535 - adc_value) / adc_value

while True:
    adc_value = adc.value       # ADC nyers adat
    ldr_resistance = calculate_resistance(adc_value)
    # Eredmények kiíratása
    print("ADC: ", adc_value, "LDR: ", ldr_resistance, "kOhm")
    time.sleep(5)
```

A fotoellenállás kalibrálása

- Feladat: változtassuk a megvilágítást, és mérjük meg az LDR ellenállását a megvilágítás függvényében! (Luxmérő alkalmazást találunk az okostelefonokhoz is)

Megvilágítás [lx]	Ellenállás [kΩ]
1	50,00
10	14,00
15	10,90
25	7,80
33	6,00
50	4,70
65	3,90
100	3,20
150	2,30
500	1,40



Mindkét tengely logaritmikus