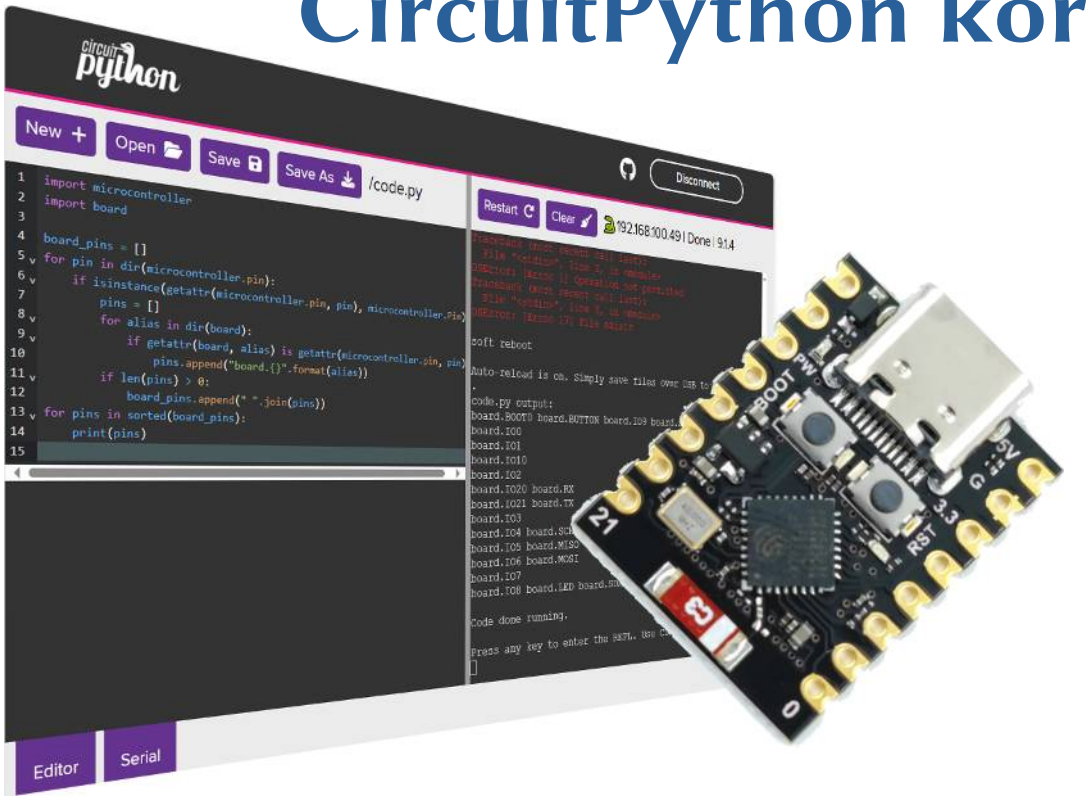
ESP32-C3 mikrovezérlők programozása CircuitPython környezetben



2. Dobókocka projekt

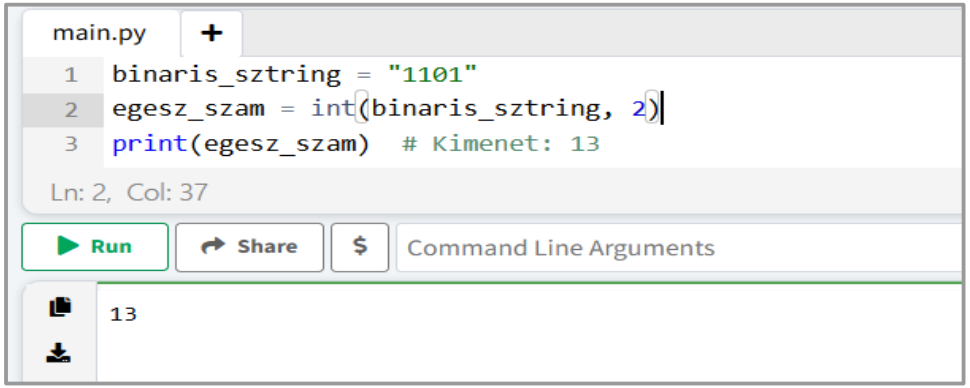
Felhasznált és ajánlott irodalom

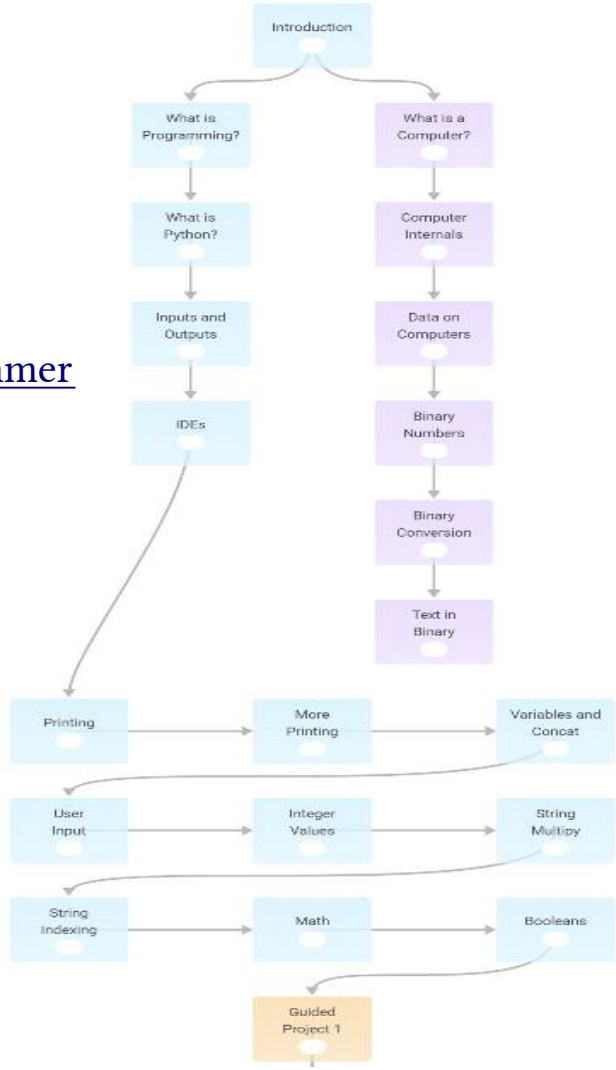
- **Python:**
  - Mark Pilgrim/Kelemen Gábor: Ugorj fejest a Python 3-ba!
  - P. Wentworth et al. (ford. Biró Piroska, Szeghalmy Szilvia és Varga Imre): Hogyan gondolkozz úgy, mint egy informatikus: Tanulás Python 3 segítségével
- **CircuitPython:**
  - Adafruit: https://circuitpython.org/downloads
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

Hogyan tanuljuk a Pythont?

- **W3Scools:** Python Tutorial
- **PyFlo:** Interactive beginners guide to becoming a Python programmer
- Online Python IDE



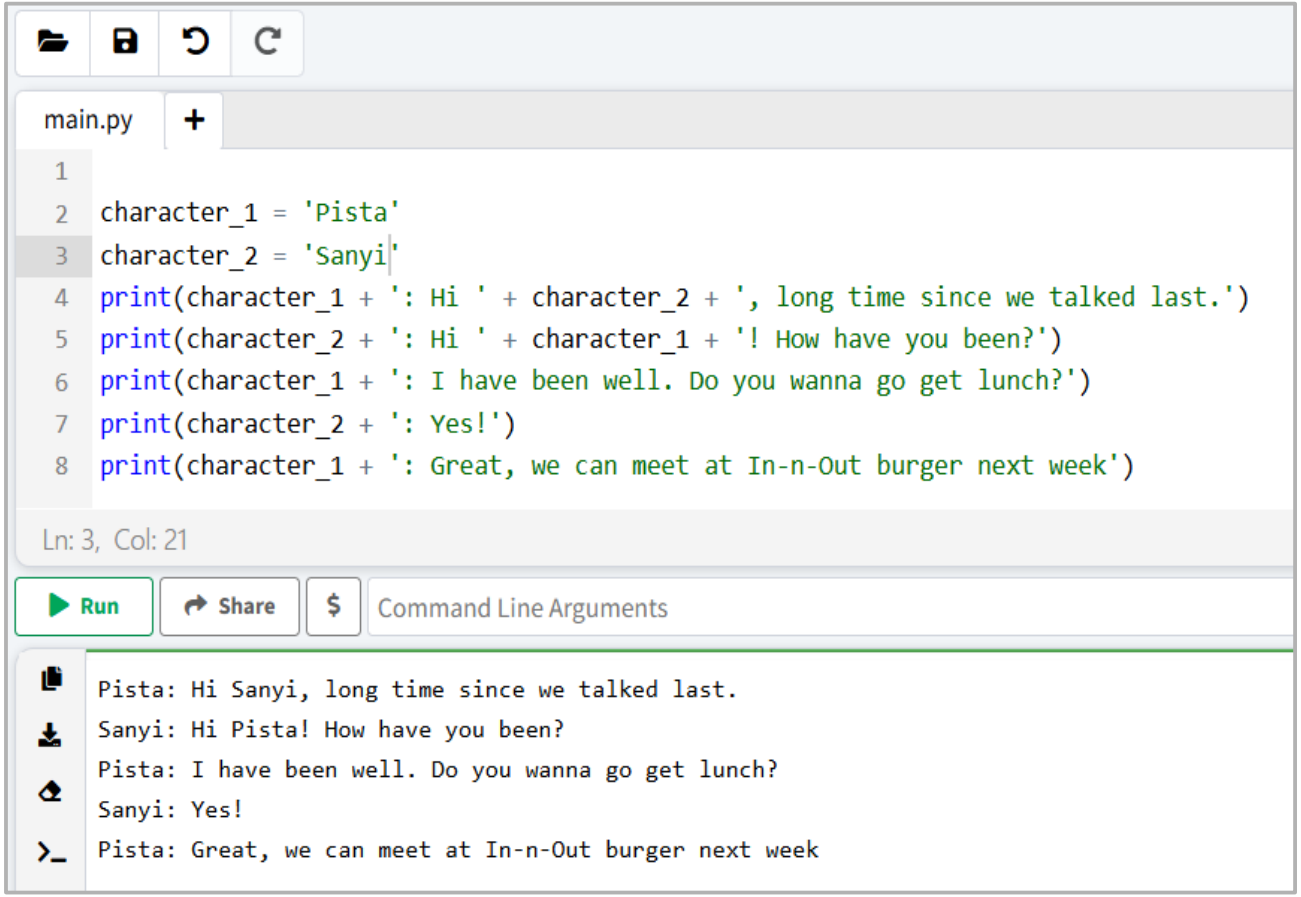


A Python nyelv elemei: interaktivitás

- Az **input()** függvény segítségével kiíratunk egy „promptot”, majd beolvasunk egy karaktersorozatot (**Enter** zárja le a beírást)
- A beolvasott szöveget egy-egy változóba mentjük el
- A **print()** segítségével szöveget vagy változók értékeit írathatjuk ki
- Több elem kiíratását vesszővel elválasztva adhatjuk meg

A Python nyelv elemei: szöveg összefűzése

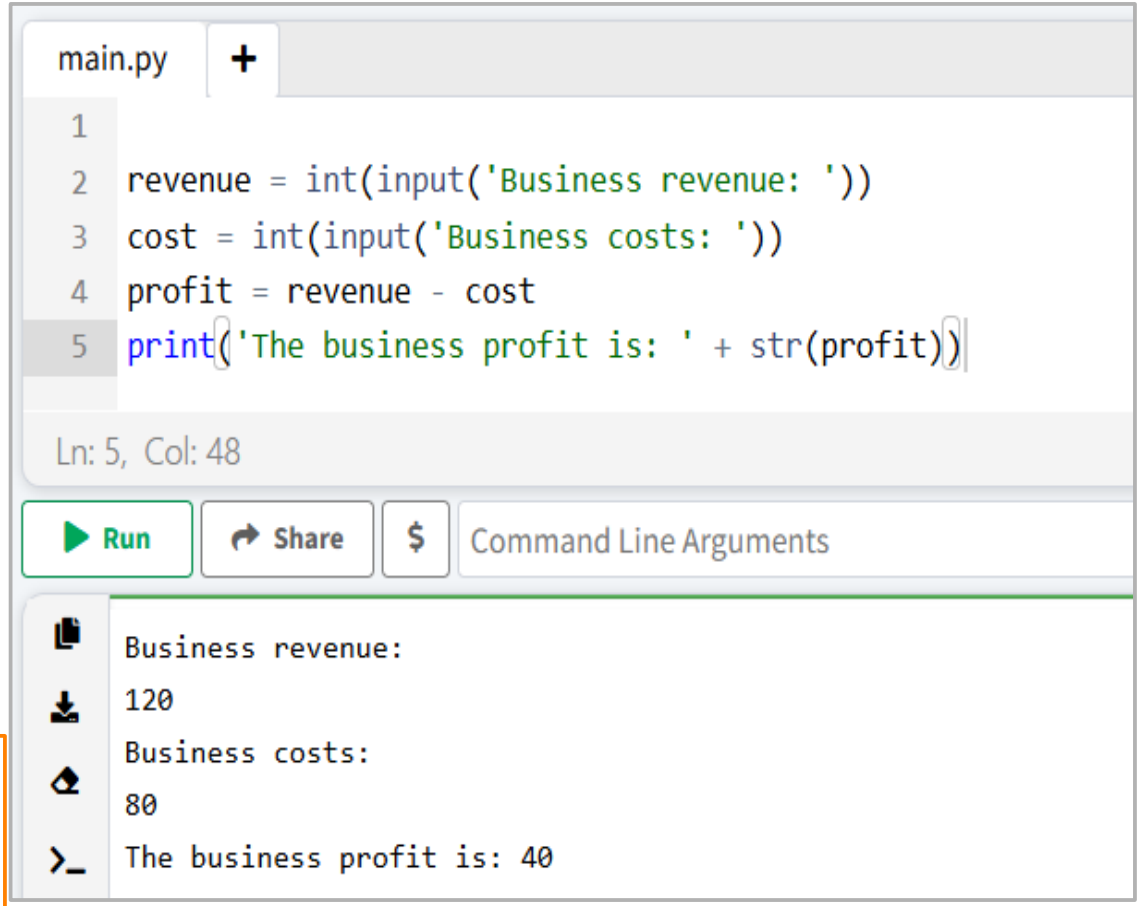
- Szövegek összefűzését, egyesítését (konkatenáció) a ’+’ jellel adhatjuk meg
- A **konkatenáció** művelet nemcsak a **print()** függvényben, hanem a szöveges típusú értékadásnál is használható
- A ’*’ jelet is használhatjuk szövegek ismétlődő összefűzésére. Például az `a = ’Apu’ * 3` eredménye `a = ’ApuApuApu’` lesz



A Python nyelv elemei: számok bevitele

- Ha a beírt számokkal műveletet akarunk végezni, akkor tudatni kell a Pythonnal, hogy **integer** típusú adataink vannak – erre szolgál az **int()** függvény
- Az **str()** függvény a fordítottját, az integer szám szöveggé alakítását végzi
- Az **int()** függvény második paramétere opcionális, és a számrendszert adja meg

```
binaris_sztring = "1101"
egesz_szam = int(binaris_sztring, 2)
print(egesz_szam)  # Kimenet: 13
```



For ciklus

- A program ismétlődő lépéseit a for ciklussal egyszerűbben írhatjuk:

```
for i in [1, 2, 3, 4, 5]:
    print(i, ". ciklus")
```



```
1 . ciklus
2 . ciklus
3 . ciklus
4 . ciklus
5 . ciklus
```

- A **range()** függvénnyel egyszerűsíthetjük a felsorolást, de a számsor nullától indul!

```
for i in range(5):
    print(i, ". ciklus")
```



```
0 . ciklus
1 . ciklus
2 . ciklus
3 . ciklus
4 . ciklus
```

Függvények létrehozása és használata

- A **Python függvények** olyan kódrészletek, amelyek egy adott feladatot hajtanak végre, és újra felhasználhatók a program különböző részein
- **Definiálás:** A függvényt a **def** kulcsszóval definiáljuk, majd megadjuk a függvény nevét és zárójelben a paramétereket (ha vannak), s a sort kettősponttal zárjuk. A függvény törzsét behúzással jelöljük
- **Hívás:** A függvényeket a nevükkel és zárójelben a paraméterekkel hívjuk meg
- A függvények a **return** kulcsszóval adhatnak vissza értékeket

```
def koszontes(nev):
    print(f"Szia, {nev}!")

# Függvény hívása
koszontes("Anna")
```

Szia, Anna!

```
def osszeadas(a, b):
    return a + b

eredmeny = osszeadas(3, 5)
print(eredmeny)
```

8

Numerikus tömbök létrehozása és kezelése

- **Tömb létrehozása:** Egy tömböt egyszerűen létrehozhatsz egy listával
- **Elemek elérése:** a tömb elemeit sorszámukkal (index) érhetjük el (a sorszámozás 0-tól kezdődik)

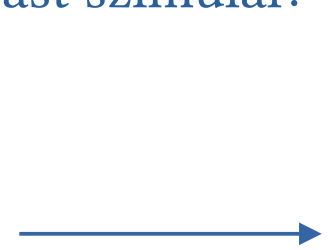
```
# Tömb létrehozása
a = [0] * 5
print("Kezdeti tömb:", a)

# Elem módosítása
a[3] = 12
print("Módosított tömb:", a)

# Elem elérése
print("A tömb 4. eleme:", a[3])

# Tömb hossza
print("A tömb hossza:", len(a))
```

```
Kezdeti tömb: [0, 0, 0, 0, 0]

Módosított tömb: [0, 0, 0, 12, 0]

A tömb 4. eleme: 12

A tömb hossza: 5
```

(Ál)véletlen számok

- Az **álvéletlen számok** (pseudo-random numbers) olyan számok, amelyek látszólag véletlenszerűek, de valójában egy matematikai algoritmus generálja ezeket a programban
- Az álvéletlen számok generálásához egy **kezdeti értéket** (seed) használnak, amely meghatározza a számok sorozatát. Ugyanazt a seed-et használva ugyanazt a számsorozatot kapjuk, ami tesztelésnél hasznos lehet
- Az alábbi **Python** program tíz kockadobást szimulál:

```
import random

def dobokocka():
    return random.randint(1, 6)

# Dobás szimulációja
for _ in range(10):
    print(dobokocka())
```



```
1
5
3
5
5
3
2
6
3
3
```

Ellenőrizzük az álvéletlen számok eloszlását

- Elvárás, hogy nagyon sok dobókocka gurítás esetén minden szám egyforma gyakorisággal szerepeljen. Ezt pl. így ellenőrizhetjük:

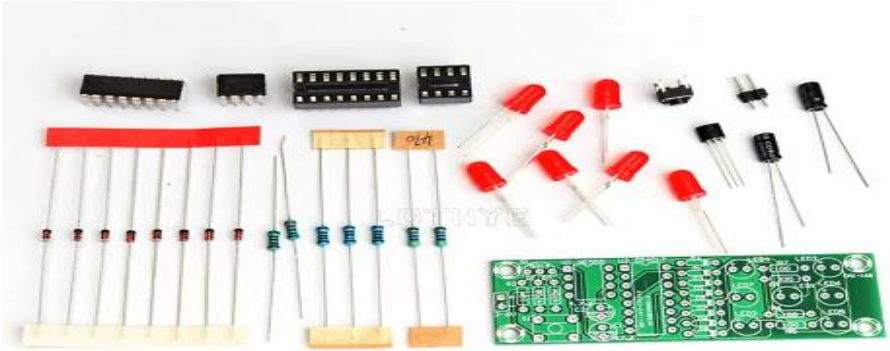
histogram.py

```
import random

# Generáljunk 120 véletlen számot 1 és 6 között
n = 120
# Számoljuk meg az egyes számok előfordulásait
Counts = [0] * 6    # Hat elemű tömb
for _ in range(n):
    number = random.randint(1,6)
    counts[number - 1] += 1

# Skálázási faktor
scaling_factor = 120 / n

# Hisztogram kiírása
print("number of samples: ",n)
for i in range(6):
    print(f"{i + 1}: {counts[i]} {'*' * int(counts[i] * scaling_factor)}")
```

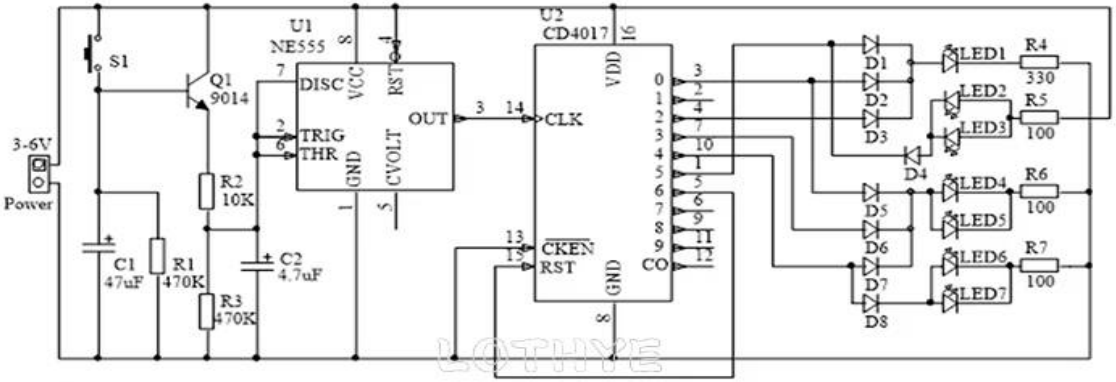
```
number of samples:  120
1: 19 *******************
2: 14 **************
3: 22 **********************
4: 26 **************************
5: 19 *******************
6: 20 ********************

number of samples:  1200
1: 219 **********************
2: 212 *********************
3: 218 **********************
4: 170 *****************
5: 201 ********************
6: 180 ******************

number of samples:  12000
1: 2096 **********************
2: 1977 ********************
3: 1960 ********************
4: 2002 **********************
5: 1979 ********************
6: 1986 ********************
```

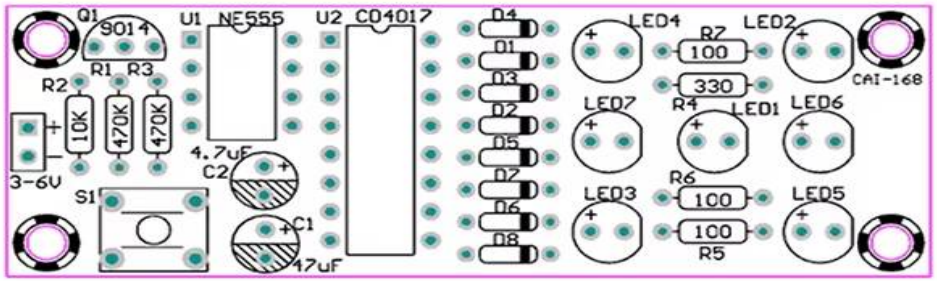
Elektronikus dobókocka

- Aliexpress: DIY Dice Production Kit



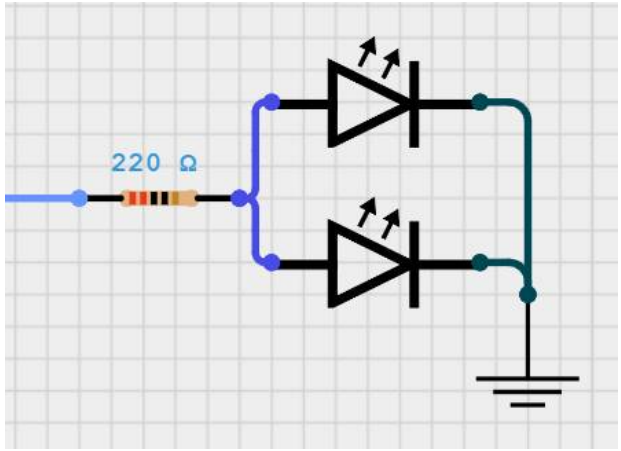
Ez egy NE555 és CD4017 IC-k segítségével megvalósított elektronikus dobókocka
Mi a gond az alábbi kapcsolással?

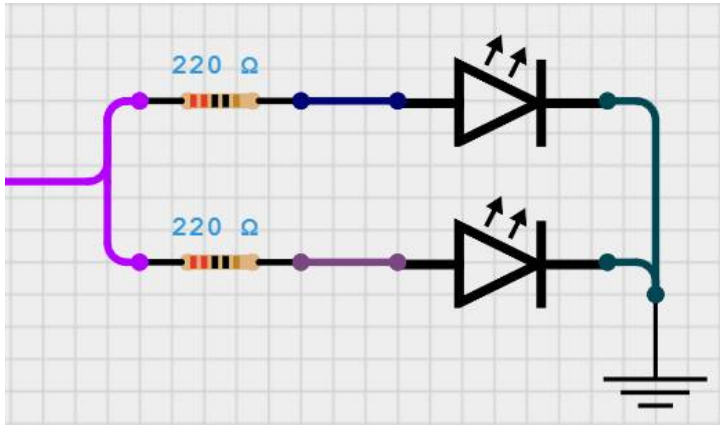




Miért kell minden LED-hez külön áramkorlátozás?

- **1. eset:** ha párhuzamosan kötjük a LED-eket, nem tudjuk szabályozni az átfolyó áram megoszlását. Ha az egyik LED-nek picivel kisebb a nyitófeszültsége, azon nagyobb áram fog folyni a másik LED rovására, emiatt a LED-ek eltérő fényerővel világítanak
- **2. eset:** ha a LED-ek saját áramkorlátozó ellenállással rendelkeznek, akkor egyik sem tudja „elvenni” a másik LED áramát, ezért kiegyenlített lesz a fényerejük



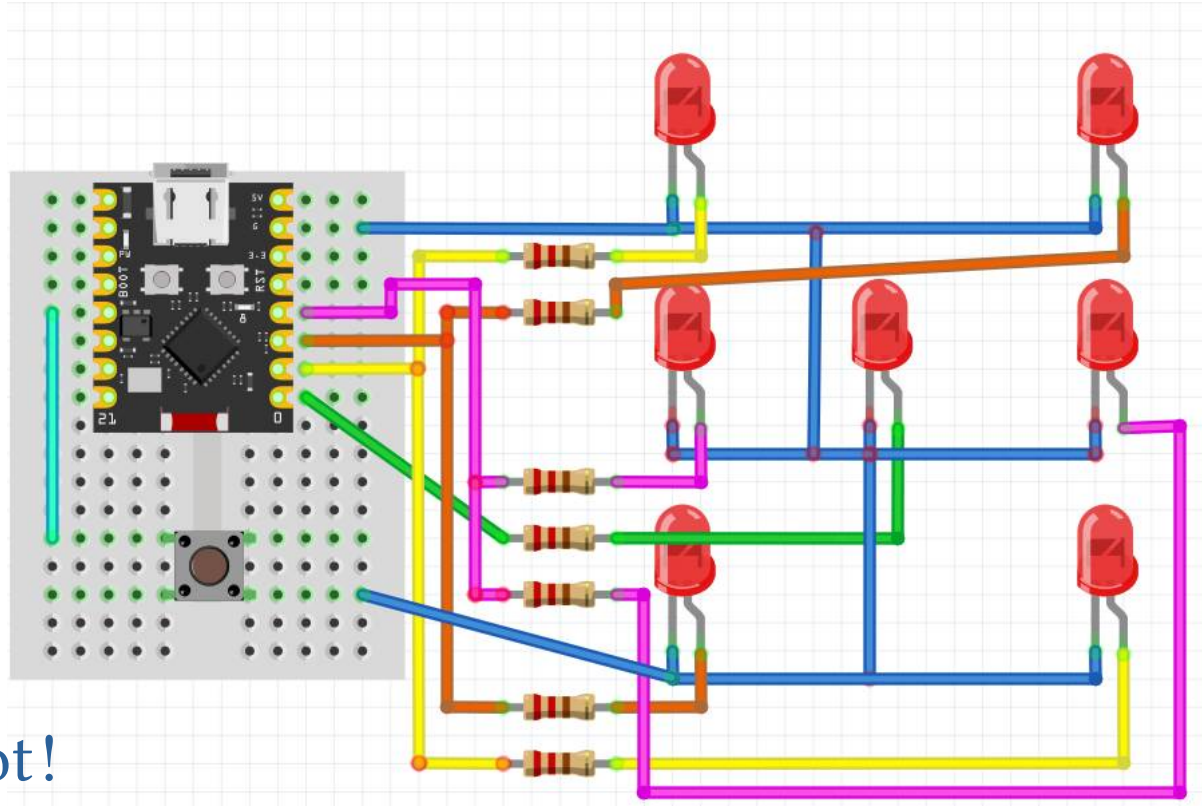


Az ESP32 C3 Super Mini kártya kivezetései

Megjegyzés: Az A5 analóg bemenet (ADC2) nem használható, ha a WiFi használatban van!

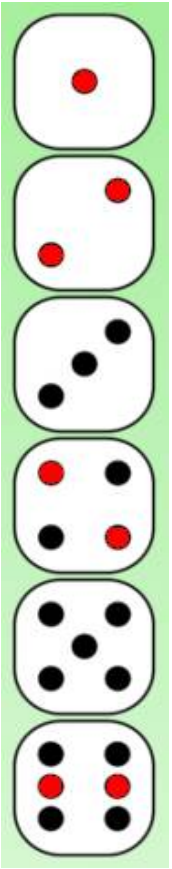
Elektronikus dobókocka készítése

- Készítsünk elektronikus dobókockát, ami gombnyomásra egy véletlen számot jelenít meg!
- A nyomógombot a **GPIO9** és a **GND** közé kötjük (így a beépített **BOOT** gombot is használhatjuk)
- A LED-eket csoportosítva a **GPIO0, GPIO1, GPIO2** és a **GPIO3** kimenetekre kötöttük
- Minden LED-hez külön-külön korlátozzuk az áramot!



A LED-ek csoportosítása

- Bizonyos LED-ek mindig párban világítanak, ezeket párhuzamosítjuk



GPIO0	Páratlan számokra = 1 páros számokra = 0	A **középső LED** minden páratlan szám esetén világít (1, 3, 5)
GPIO1	Minden 1-nél nagyobb szám esetén = 1	A **bal alsó és jobb felső LED** minden 1-nél nagyobb szám esetén világít (2, 3, 4, 5, 6)
GPIO0 + GPIO1	Az előző kettő kombinációja	A hármas számhoz nem kell új LED csoportot bevezetni
GPIO1 + **GPIO2**	Minden 3-nál nagyobb számra **GPIO2** = 1	A **jobb alsó és bal felső LED** minden 3-nál nagyobb szám esetén világít
GPIO0 + GPIO1 + GPIO2	Az előző három csoport kombinációja	Az ötös számhoz nem kell új LED csoportot bevezetni
GPIO1 + GPIO2 + **GPIO3**	**GPIO3** = 1, ha a szám 6, Egyébként pedig 0	A két oldalsó-középső LED akkor és csak akkor világít, ha a szám = 6

ESP32_dobokocka.py – 2/1.

ESP32_dobokocka.py

```
import time
import board
import digitalio
import random

# LED-ek inicializálása
led_pins = [board.IO0, board.IO1, board.IO2, board.IO3]
leds = [digitalio.DigitalInOut(pin) for pin in led_pins]
for led in leds:
    led.direction = digitalio.Direction.OUTPUT

# Gomb inicializálása
button = digitalio.DigitalInOut(board.BUTTON)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

def roll_dice():                          # Kocka ”gurítás” véletlen számmal
    return random.randint(1,6)
```

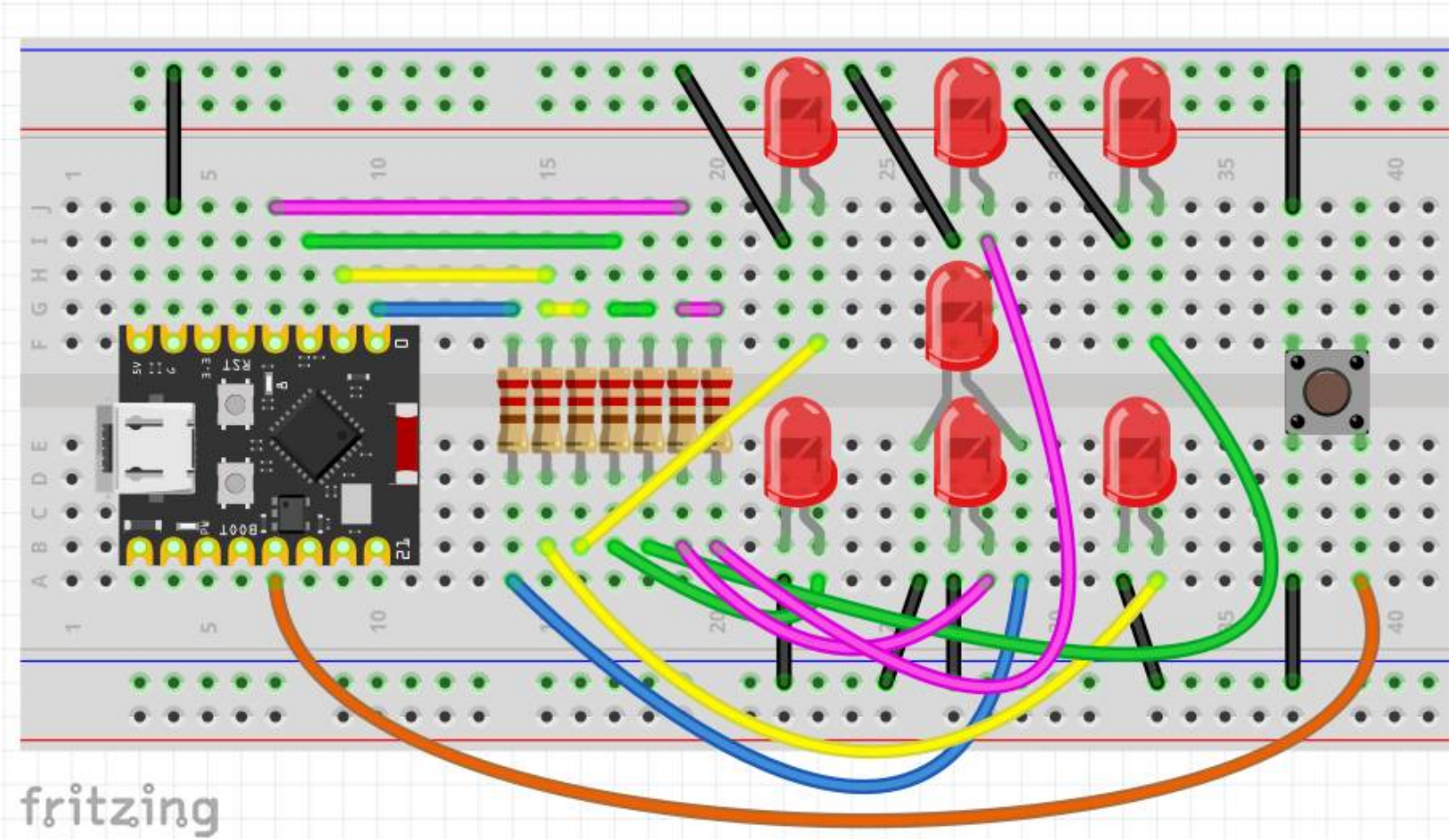
A LED csoportok:

leds[0]
leds[1]
leds[2]
leds[3]

ESP32_dobokocka.py – 2/2.

```
def display_number(number):
    # LED-ek állapotának beállítása
    leds[0].value = (number%2 == 1)  # Középső LED (páratlan számok)
    leds[1].value = (number > 1)     # Bal alsó és jobb felső LED
    leds[2].value = (number > 3)     # Jobb alsó és bal felső LED
    leds[3].value = (number == 6)    # Oldalsó középső LED-ek

while True:
    while button.value:              # Várunk a gomb lenyomására
        pass
    time.sleep(0.02)                 # Pergésmentesítő késleltetés
    number = roll_dice()             # Kockagurítás és kijelzés
    display_number(number)
    while not button.value:          # Várunk a gomb felengedésére
        pass
    time.sleep(5)                    # Hosszabb késleltetés
```

Egy lehetséges elrendezés a próbapanelon



Ellenőrizzük a programot

- Egészítsük ki a **display_number()** függvényt néhány kiíró utasítással: írassuk ki a „véletlen” számot és a LED vezérlő kimenetek állapotát
- A fentiekkel ellenőrizhetjük, hogy a kimenetek értékeit jól számoltuk-e ki

```
def display_number(number):
    # LED-ek állapotának beállítása
    leds[0].value = (number%2 == 1)  # Középső LED (páratlan számok)
    leds[1].value = number > 1       # Bal alsó és jobb felső LED
    leds[2].value = number > 3       # Jobb alsó és bal felső LED
    leds[3].value = number == 6      # Oldalsó középső LED-ek
    print(number,end=" ")
    for i in range(4):
        print(leds[i].value,end=" ")
    print(" ")
```

```
code.py output:
4 False True True False     (GPIO0, GPIO1, GPIO2, GPIO3)
3 True True False False
2 False True False False
3 True True False False
1 True False False False
2 False True False False
```