3. alkalom (pinek kezelése, LED használata, véletlenszámok)

Tematikai egység	**Alkalmazott módszerek, munkaformák**	**Időtartam**
Ismétlés	Egyéni munka, majd a megoldás megbeszélése közösen	10 perc
Pinek, ki- és bemenet	Frontális tanári magyarázat, közös programírás, önálló munka	15 perc
LED rákötése az eszközre	Bemutató, önálló munka	10 perc
LED villogtatása	Önálló munka	15 perc
A random modul bemutatása	Frontális tanári magyarázat	5 perc
Sorsolóprogram	Közös programírás	5 perc
Dobókocka készítése	Közös programírás	5 perc
Véletlenszám előállítási módok	Frontális tanári magyarázat, csoportszintű megbeszélés, pármunka	15 perc
A véletlenszámok előállításának elve	Frontális tanári magyarázat, megbeszélés	10 perc

1. Pinek, ki- és bemenet

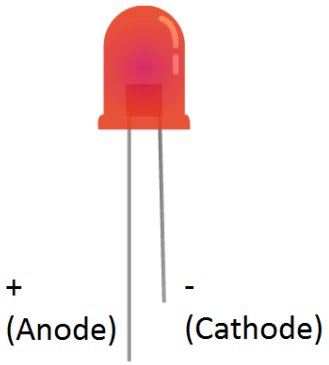
A micro:bit alján fémcsíkok találhatóak, amitől az eszköz úgy néz ki, mintha fogai lennének. Ezek az ún. ki- és bemeneti (I/O) pinek. Összesen 19 pint használhatunk, amikből 5 darab nagyobb, mint a többi. Ezen öt pin mindegyikén található egy jelölés, ezek sorban a 0, az 1, a 2, a 3V, és a GND (ground, földelés). Ezekre krokodilcsipeszek segítségével különféle eszközöket csatlakoztathatunk. Kódszinten minden pin egy objektummal van reprezentálva, aminek a neve `pinN`, ahol `N` a pin száma. Például a 0-s pinnel való munkához a `pin0` nevű objektumot használhatjuk. Minden ilyen objektumnak vannak metódusai, ám ezek nem egyeznek meg minden pin esetén, hiszen különböző célokra használhatjuk őket. Nézzünk egy rövid példát a bemenet kezelésére!

A legegyszerűbb azt vizsgálnunk, hogy meg van-e érintve egy adott pin. Szimuláljuk le azt, hogy megcsikizzük a micro:bitet!

```
while True:
    if pin0.is_touched():
        display.show(Image.HAPPY)
    else:
        display.show(Image.SAD)
```

A végtelen cikluson belül ezúttal azt vizsgáljuk, hogy a 0-s számú pint érintjük-e éppen. Erre a célra az `is_touched()` metódust használhatjuk. Amikor érintjük a pint, a micro:bit mosolyog, különben szomorú. A program kipróbálása a következőképpen zajlik: egyik kezünkkel fogjuk meg a GND jelű pint, majd a másik kezünkkel érintsük meg a 0-s számút! Ha minden jól megy, látni fogjuk kirajzolódni a mosolygó arcot. Ezzel megvalósítottunk egy nagyon alapvető bemenet-kezelést, ám a pinek segítségével rengeteg egyéb lehetőségünk is van, amire a későbbiekben még nézünk példákat.

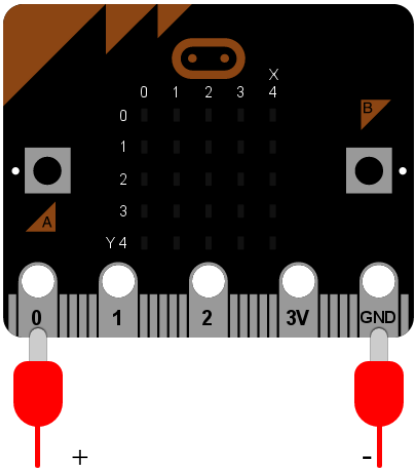
Most pedig nézzük meg a – talán – legegyszerűbb példát a kimenet kezelésére. Ehhez valamilyen eszközt kell csatlakoztatnunk a pinekre, az én választásom ezúttal egy LED-re esett. Ahogy az 1. ábrán láthatjuk, egy LED-nek két kivezetése van, a hosszabb a pozitív (anód), a rövidebb a negatív (katód). A LED kizárólag akkor fog helyesen működni, ha az egyes kivezetéseket jó helyre kötjük be.



1. ábra A LED és kivezetései[9]

A LED-et krokodilcsipeszek segítségével tudjuk rákötni az eszközre. A pozitív kivezetést kössük a 0 pinre, a negatívat pedig a GND jelzésűre, ahogy az az ábrán is látszik!

[9] http://www.makerspace-uk.co.uk/wp-content/uploads/2016/07/led-annotated.png Elérés dátuma: 2018.08.04.



2. ábra A LED összekötése a micro:bittel[10]

Ha esetleg nincs krokodilcsipeszünk, alufóliával és ragasztószalaggal is dolgozhatunk. Az alábbi ábrán látható egy megoldás erre a problémára, amin egy fülhallgató van rákötve a micro:bitre. Ez természetesen kényelmetlenebb, mint az előző módszer, ám eszköz hiányában remek helyettesítő megoldás.

3. ábra Alufólia használata krokodilcsipeszek helyett[11]

[10] http://www.makerspace-uk.co.uk/wp-content/uploads/2016/07/LED-no-wires_bb.png Elérés dátuma: 2018.08.04

[11] https://pxt.azureedge.net/blob/a2f8a1af1a5c4122df2bd13d1e9fd314fd295ef5/static/mb/device/croc-clips/foilcircuit.jpg Elérés dátuma: 2018.08.04.

Amint készen vagyunk az összekötéssel, ki is próbálhatjuk azt egy egyszerű programmal. Gépeljük be az alábbi sort a LED felkapcsolásához.

```
pin0.write_digital(1)
```

Ez ugyan csak pár másodpercig izgalmas, ám hamar rá fogunk jönni, hogy a pinek segítségével szinte végtelen számú ötletet meg lehet valósítani!

Gyakorlásképpen bonyolítsuk egy kicsit a fenti programot!

Feladat a diákok számára

Oldd meg, hogy másodpercenként kétszer villanjon fel a LED! Egy felvillanás ideje legyen 20 ms.

A diákok számára ez egy nem feltétlenül gyorsan, de megoldható feladat, a számokkal való kísérletezés során hamar rá lehet jönni a megfelelő értékekre. Segítségképpen álljon itt a megoldás:

```
while True:
    pin0.write_digital(1)
    sleep(20)
    pin0.write_digital(0)
    sleep(480)
```

Ha esetleg nem lenne elég látványos az eredmény, bátran kísérletezzünk az időértékekkel!

Ezáltal készítettünk egy jelzőfényt, ami sok további programnak lehet a kiindulópontja.

2. A random modul, véletlenszámok előállítása, az előállítás elvei

Néha szükségünk van arra, hogy valami ne legyen előre eldöntve, hanem véletlenszerűen történjen meg. Ehhez segítségül hívhatjuk a Pythonban található `random` modult. Például, ha a csoport tagjai közül akarunk kiválasztani valakit véletlenszerűen, egy ilyen programot kell írnunk:

```
import random

names=["Attila", "Bence", "Csilla", "Dani", "Emese", "Fruzsi"
]

display.scroll(random.choice(names))
```

Ezúttal nem elég a minden program elején használt általános import utasítás, hanem importálnunk kell a `random` modult is. Az ebben található `choice()` metódussal tudunk egy lista elemeiből véletlenszerűen kiválasztani egyet. Miután kiválasztottunk egy nevet, a korábban használt `scroll()` metódussal kiírjuk azt a képernyőre. Ilyen egyszerűen készíthetünk egy sorsolóprogramot.

Legtöbbször a `random` modult inkább számok előállítására használjuk. A véletlenszámok használata nagyon gyakori például a játékokban, elég csak szimplán a dobókockára gondolni, ami számtalan játék alapeleme. Készítsünk most el egyet a micro:biten!

```
import random

display.show(random.randint(1, 6))
```

Az eszköz minden újraindításnál kiír egy 1 és 6 közötti számot, a `randint()` metódus segítségével. Láthatjuk, hogy ez két számot vár paraméteréül, annak az intervallumnak az alsó- és felső határát, amiből szeretnénk véletlenszámokat visszakapni (a két határ is beletartozik az intervallumba). A függvény ezután, ahogy a neve is jelzi, egy véletlenszerűen kiválasztott egész számot ad vissza. Hívjuk fel a diákok figyelmét a függvény nevében található `int` szóra, és ha korábban nem találkoztak ezzel, mondjuk el nekik, hogy az `int`, vagy `integer` a különböző programnyelvekben az egész szám típusnak felel meg.

Ha esetleg `0` és `N` közötti számokra lenne szükségünk, használhatjuk a `random.randrange()` metódust is. Ez `0` és a paraméteréül megadott szám közötti véletlen egész számot ad vissza, azonban ennél a paraméterben megadott szám nem lesz része az intervallumnak, amiből a szám kikerül. Erre a különbségre érdemes nyomatékosan felhívni a figyelmet!

Miután az egész számok előállításával már boldogulunk, mindenképpen szót kell ejteni arról az esetről is, amikor valós számokra lenne szükségünk. Hívjuk ezeket a számokat az életkori sajátosságoknak megfelelően így, esetleg tizedestörtnek, vagy csak szimplán írjunk fel a táblára egy példát, hogy demonstráljuk ezeknek a számoknak az alakját. Ilyen számok előállítására a `random.random()` metódust tudjuk használni, ami `0.0` és `1.0` közötti valós számokat ad vissza (a két határ is beleértendő az intervallumba). Amennyiben ennél nagyobb számra volna szükségünk, adjuk hozzá az eredményt a `random.randrange()` metódus által visszaadott számhoz, ahogy ezt az alábbi program

szemlélteti. A program megmutatása előtt a csoport szintjétől függően hagyjuk a diákokat gondolkodni a probléma megoldásán! Akár páros munkát is alkalmazhatunk, hogy egymást segítve jöhessenek rá a megoldásra.

```
import random

answer = random.randrange(100) + random.random()
display.scroll(answer)
```

A program egy 0 és 100 közötti valós számot állít elő, majd kiírja ezt a kijelzőre.

Érdemes lehet kicsit többet beszélni a véletlenszámok előállításának elvéről, hiszen ez Pythonban egy egyszerű példán keresztül bemutatható. Amit hasznos tudni a véletlenszámokról az az, hogy valójában nem véletlenek, csak úgy tűnhetnek. Ezeknek az előállítása nem más, mint véletlenszerű eredmények számolása egy bizonyos kezdeti (ún. `seed`) értékből. Ez az érték valami olyan, ami nem lehet kétszer ugyanaz, legtöbbször a rendszeridő. Ez kiegészülhet esetleg még pár dologgal, a nagyon nagy valószínűséggel jó eredmény érdekében. Hogyha megismételhetően véletlen eredményt szeretnénk előállítani, mi is beállíthatjuk a `seed` értékét egy tetszőleges számra. Ugyanazzal a `seed` értékkel minden egyes programlefutáskor ugyanazt a számsorozatot kapjuk vissza a `random` metódusait meghívva. Nézzünk egy példát rá a korábbi, dobókockát szimuláló kódot kiegészítve:

```
1. import random
2.
3. random.seed(1337)
4. while True:
5.     if button_a.was_pressed():
6.         display.show(random.randint(1, 6))
```

A 3. sorban adtuk meg a kiindulási értéket, ami jelen esetben 1337. Ezután akárhányszor megnyomjuk az „A” gombot, kapunk egy új egész véletlenszámot. A csavar a dologban, hogy akármikor elindítjuk a programot, a kapott számsorozat ugyanaz lesz (2, 3, 5, 1, 1…). Ebből is látszik, hogy a véletlenszám generálás valójában nem teljesen véletlenszerű, hanem csupán valamilyen műveletsorozat segítségével kiszámolt értékeket kapunk vissza.