MálnaSuli
Raspberry Pi Magyarország

http://www.malnasuli.hu/oktatas/a-python-programozasi-nyelv-1/

A PYTHON PROGRAMOZÁSI NYELV – 1.

Írta: K András | 2020.1.11. | Oktatási anyagok, Python, Raspberry Pi |

BEVEZETÉS – ALAPFOGALMAK

A Python egy általános célú, magas szintű, interaktív, objektum orientált programozási nyelv, melyet Guido van Rossum holland programozó kezdett el fejleszteni 1989 végén, majd hozott nyilvánosságra 1991-ben. A nyelv tervezési filozófiája az „olvashatóságot" és a programozói munka megkönnyítését helyezi előtérbe a futási sebességgel szemben.

A Python nyelv hordozható, azaz elérhető különféle Linux (Raspbian) és Windows operációs rendszerek, valamint macOS operációs rendszer alatt is. Ingyenes és korlátozás nélkül használható. Egyszerű szintaxis (-> az adott programozási nyelv formai szabályai) jellemzi, így tömör, könnyen olvasható programok írhatók Pythonban. A Python folyamatosan fejlődő nyelv, ami mögött lelkes felhasználók és fejlesztők közössége áll, akiknek többsége támogatja a szabad szoftvereket.

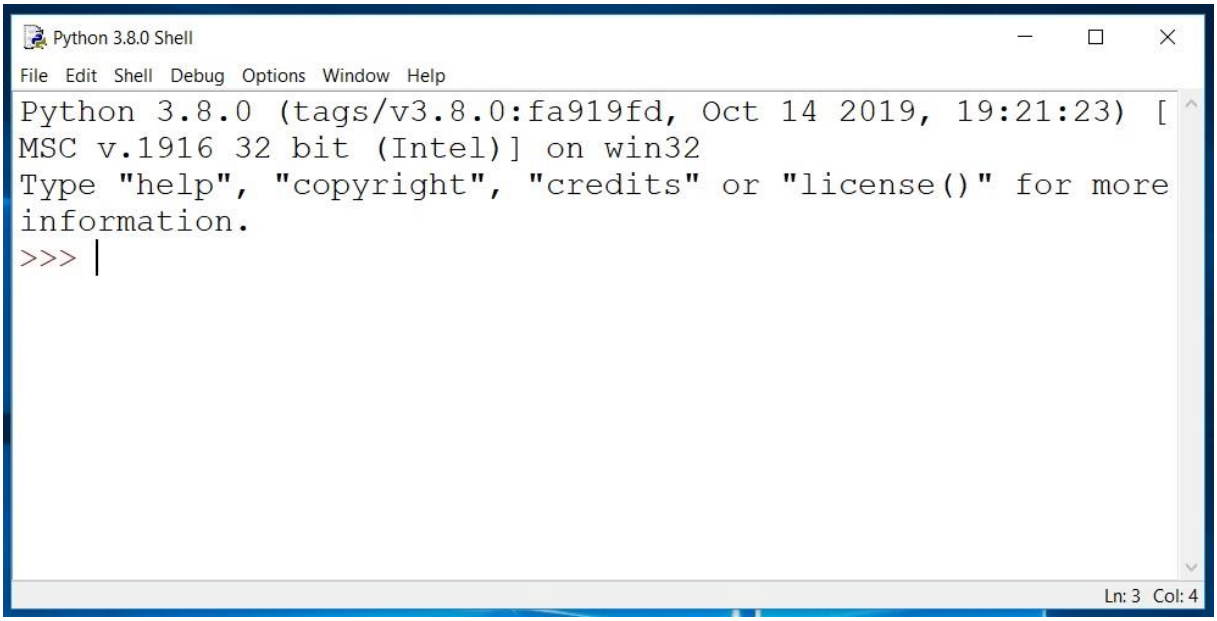
A Python értelmező és a sokrétű, alaposan kidolgozott alap-könyvtár (Standard Library) szabadon elérhető és felhasználható akár forráskódként, akár bináris formában minden jelentősebb platformra a Python weboldaláról. Ugyanitt találhatóak hivatkozások külső fejlesztésű modulokra, programokra és eszközökre, és kiegészítő dokumentációkra.

TELEPÍTÉS – HASZNÁLAT

A Python legfrissebb változata jelen segédlet írásának idején a 3.8.0 verzió. (Release Date: Oct. 14, 2019). Linux és macOS operációs rendszerek alatt általában alapból telepítve van a Python megfelelő verziója. A Raspberry Pi számítógépen futó Raspbian operációs rendszernek része a Python 3. A Windows operációs rendszer alá külön lehet letölteni és telepíteni a https://www.python.org oldalról.
A Python nyelvnek két nagy verziója van, a Python 3 és a Python 2. A Python 3 olyan újításokat tartalmaz, amik miatt a változat visszafelé nem kompatibilis a korábbi verzióval. Ebben a leírásban a Python 3-at használjuk, és ennek a telepítését javasoljuk az Olvasónak is.

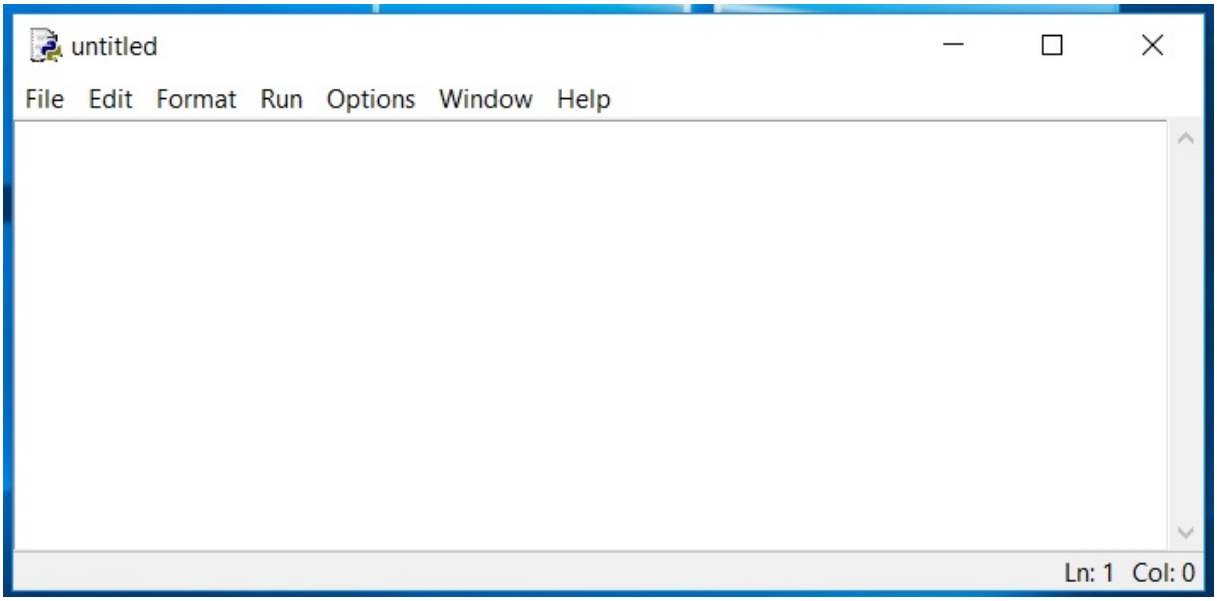
Python környezet telepítése

A telepítés után indítható a Python Shell (héj) program, aminek parancsértelmező ablaka a következőképpen néz ki:



Python shell

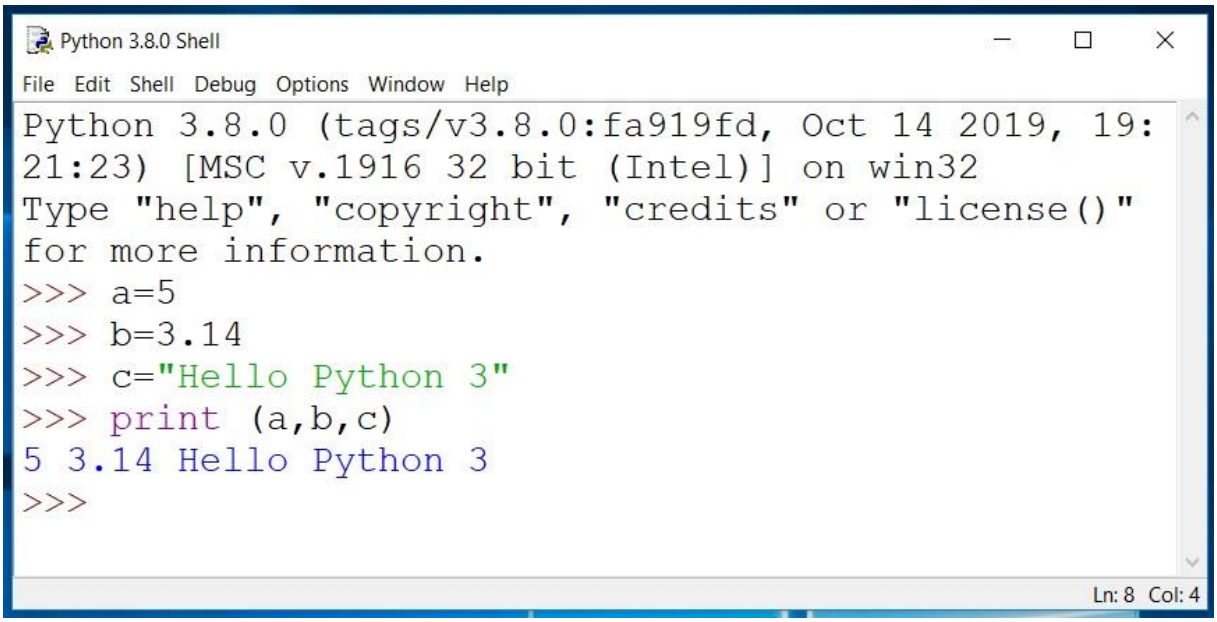
Az ablakban tesztelhetők a Python parancsok, ha program scriptet kívánunk írni, akkor a file menü new file menüpontjával tudunk ehhez ablakot nyitni.



Script ablak

Az ide begépelt parancsok nem kerülnek azonnal végrehajtásra, hanem mentés után a Run menü Runmodule parancsával futtathatók.

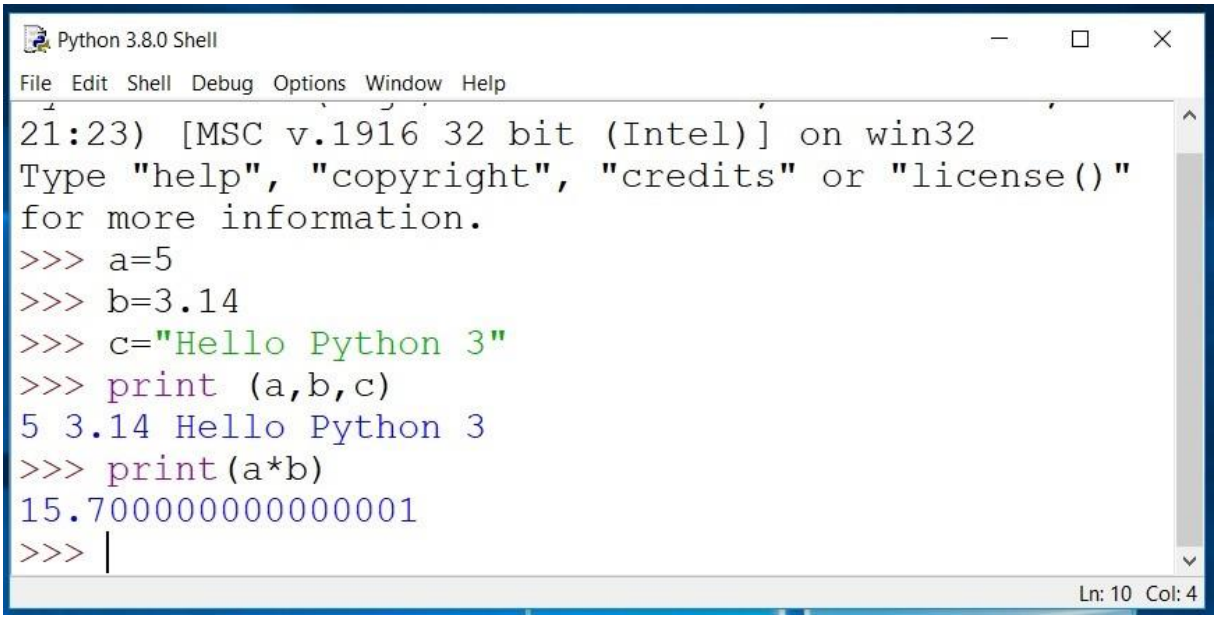
A Shell-ben tesztelhetjük a parancsokat. Példaként nézzük meg, hogy a Python hogyan kezeli a változókat! (-> a memóriának egy bizonyos rekeszét, amelyben adatokat kívánunk tárolni, és később visszaolvasni, változónak nevezzük és a nevével „hivatkozzuk”.)



Első programunk futtatása

Az egyszerű példában egyenként gépeljük be a sorokat, és mindegyik végén nyomjuk le az Enter billentyűt. Az első három sor egyszerű értékadás, amivel egyben definiáljuk is az a, b, c változók típusát is. Az „a” és a „b” változók számérték, a „c” pedig szöveg (string – karakterfüzér) tárolására alkalmas változó. A következő sorban a print parancs hatására a Shell kiírja a változók értékét.

Folytatva a gépelést a változók értékével műveleteket is végezhetünk:



Műveletek eredménye

Ha a változókkal nem csak egyszer szeretnénk elvégezni a műveleteket, akkor célszerű a parancsokat egy programban elhelyezni, így tetszőleges alkalommal tudjuk futtatni a scriptet.

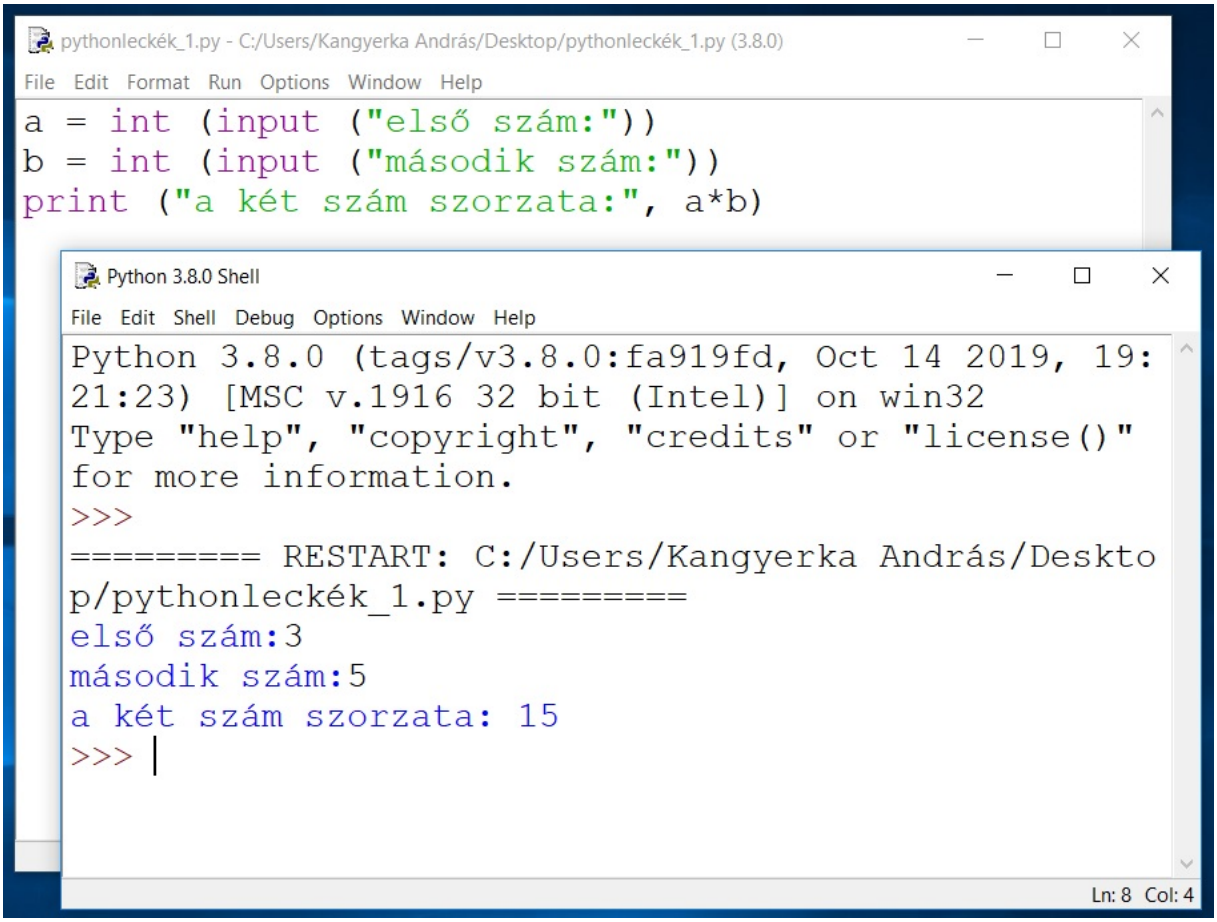
```
a = int (input ("első szám:"))
b = int (input ("második szám:"))
print ("a két szám szorzata:", a*b)
```

A programban szereplő python 3 függvények:

input(prompt) adatbekérés felhasználótól.
print() to display adatkiírás a konzolra.

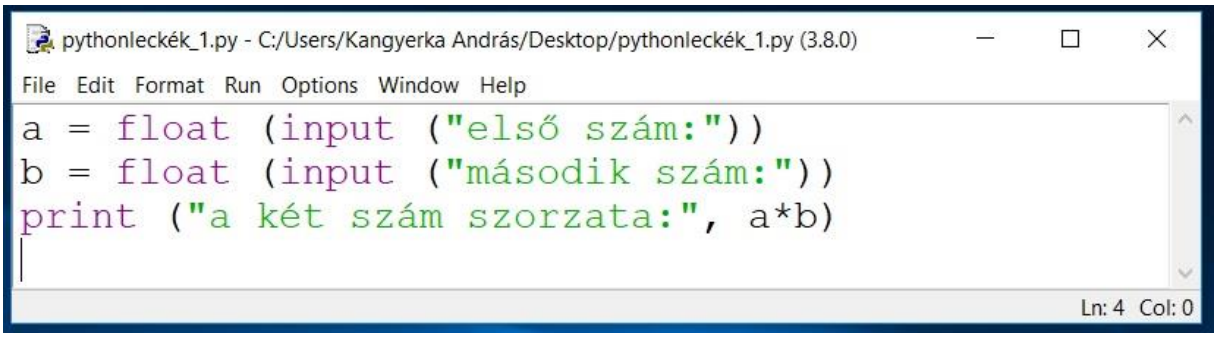
Az input () függvény beolvas egy bemeneti eszközről, például a billentyűzetről egy bevitt sort, és azt karakterlánccá konvertálja. Ezt a bevitt karakterláncot felhasználhatjuk a Python kódban. Így olvashatjuk be pl. a műveletünkhöz a számokat. Az így bevitt adatok azonban még nem numerikus (szám) értékek, a művelet elvégzéshez számmá kell őket konvertálni. Erre szolgál az input függvény elé írt int (integer – egész szám) parancs, aminek hatására az „a” illetve a „b” változókba a string formátumban beolvasott értékek számmá konvertálva kerülnek beírásra.

A program illetve a futás eredménye a konzolon:



Változók konverziója

Ha nem csak egész számokkal szeretnénk műveleteket végezni, akkor az adatbevitelre használt változókat egész típusúról tizedes tört tárolására is alkalmas un. „float” típusúra kell változtatni. A program a változtatás után:



Float típusú változók

Az első lecke zárásaként nézzünk egy példát két karakteres típusú (string) adattal való műveletvégzésre!

String típusú változók

A program futási eredménye:

Adatbekérés string típusú változóba, futtatás eredménye

A PYTHON PROGRAMOZÁSI NYELV – 2. DÖNTÉSHOZATAL

Írta: K András | 2020.3.17. | Oktatási anyagok, Python, Raspberry Pi |

A malnasuli.hu oldalon megjelenő cikkeket, illetve a cikkekben közreadott mintaprogramokat a publikálás előtt természetesen átnézzük és teszteljük. Ennek ellenére előfordulhat, hogy az oldalon minden igyekezetünk ellenére hibás példaprogram jelenik meg. Köszönjük az oldal olvasóinak, ha jelzik felénk ezeket az esetleges hibákat, amiket igyekszünk gyorsan orvosolni és javítani.
Köszönjük tehát kedves olvasónk, Soocy észrevételét, mely alapján javítottuk a 4. példaprogramot és a hozzá kapcsolódó leírást.
2021.01.03.

1. FELTÉTELEK ÉS A „HA” (IF) KULCSSZÓ

A Python támogatja a matematikában megszokott logikai feltételeket:

- Egyenlő: a == b
- Nem egyenlő: a! = B
- Kevesebb, mint: a <b
- Kevesebb vagy egyenlő: a <= b
- Nagyobb, mint: a> b
- Nagyobb vagy egyenlő: a> = b

Ezeket a feltételeket többféle módon, többféle kombinációban lehet használni, leggyakrabban a döntéshozatal és a ciklusok esetében.

Az első egyszerű példa a döntéshozatalra bekér két numerikus értéket, és összehasonlítja azokat:

```
a = input("az első szám: ")
b = input("a második szám: ")
if b >a:
print("a második szám nagyobb mint az első")
```

A példában két változót használunk (a,b) amiket az input paranccsal „töltünk fel” értékkel. Az ifb > a feltétel igaz értéke esetén végrehajtódik a print parancs, ami a terminál ablakba írja az összehasonlítás eredményét.

A programnak ez a verziója nem reagál az összehasonlítás hamis értékére, illetve csak egyszer hajtódik végre. Fontos szintaktikai szabály, hogy az if…. összehasonlítás sorát „:” zárja, valamint az, hogy az utána következő, az összehasonlítás igaz értéke esetén végrehajtódó sorok behúzással, tabulátorral íródjanak! A program alábbi beírása esetén:

```
a = input("az első szám: ")
b = input("a második szám: ")
if b >a:
print("a második szám nagyobb mint az első")
```

hibaüzenetet kapunk. Ez a behúzás elhagyására utal!

2. AZ „ELIF” KULCSSZÓ

Az elif kulcsszó a pythonban kb. azt jelenti: “Ha a korábbi feltételek nem voltak igazak, akkor próbáld ki ezt a feltételt”.

Nézzük meg az előbbi programot ezzel a bővítéssel:

```
a = input("az első szám: ")
b = input("a második szám: ")
if b >a:
print("a második szám nagyobb mint az első")
elif a==b:
print("a két szám egyenlő")
```

Ez a programváltozat már két esetet tud levizsgálni:ha a második szám a nagyobb, illetve, ha a két szám egyenlő.

Az összes lehetőséget az alábbi megoldással értékelhetjük ki:

```
a = input("az első szám: ")
b = input("a második szám: ")
if b >a:
print("a második szám nagyobb mint az első")
elif a==b:
print("a két szám egyenlő")
```

```
elifa > b:
print("az első szám nagyobb mint a második")
```

A fenti megoldást adja az „else” parancs használata is:

3. AZ „ELSE” KULCSSZÓ

Az alapvető különbség, hogy míg az „elif” kulcsszóval konkrét leválogatást tettünk meg, az „else” bármi egyéb értékre ugyan azt a választ adja. Ez ebben az esetben azonos mint az előbbi példában, hiszen csak három lehetőség van.

```
a = input("az első szám: ")
b = input("a második szám: ")
if b >a:
print("a második szám nagyobb mint az első")
elif a==b:
print("a két szám egyenlő")
else:
print("az első szám nagyobb mint a második")
```

Ha csak egy végrehajtandó utasítás van, akkor ugyanabba a sorba is írhatjuk, mint az if utasítást.

```
if a > b: print("a nagyobb mint b")
```

4. EGYMÁSBA ÁGYAZOTT DÖNTÉSHOZATAL (NESTED IF)

A programozás során előfordulhat olyan helyzet, hogy egy feltétel teljesülését egy korábbi feltétel teljesülése esetén szeretnénk levizsgálni. Ilyen esetekben használhatjuk a beágyazott if szerkezetet. Erre olyankor lehet szükség, ha pl. két feltétel egyidejű teljesülését szeretnénk kimutatni.

Az ún. beágyazott if megoldás esetében egy második if-elif-else szerkezetet használunk az első if-elif-else belsejében.

A fent említett megoldás szintaxisa:

```
if kifejezés1:
parancs(ok)
if kifejezés2:
parancs(ok)
elif kifejezés3:
parancs(ok)
else
parancs(ok)
elif kifejezés4:
parancs(ok)
else:
parancs(ok)
```

Nézzünk egy példát a fenti esetre. Legyen a feladat annak eldöntése, hogy egy adott szám osztható-e egyszerre 3-mal illetve 2-vel, vagy csak az egyikkel, vagy a másikkal, vagy egyikkel se. A megoldásban alkalmazzuk a Python un. modulo (%) (maradék nélküli osztás) függvényét, ami az oszthatóság teljesülése esetén 0-t ad eredményül.

A mintaprogram:

```
x = int(input("írj be egy számot: "))
if x%2 == 0:
if x%3 == 0:
print ("a szám osztható 3-mal és 2-vel")
else:
print ("a szám osztható 2-vel de nem osztható 3-mal")
else:
if x%3 == 0:
print ("a szám osztható 3-mal de nem osztható 2-vel")
else:
print ("a szám sem 2-vel, sem 3-mal nem osztható")
```

5. KOMBINÁLT DÖNTÉSHOZATAL

Bizonyos esetekben szükségünk lehet a leválogatásokat bizonyos értékhatárokhoz kötni, pl. ponthatárok és osztályzatok esetén. Könnyítsük meg a dolgozatokat javító és pontozó tanár dolgát egy olyan egyszerű kis algoritmussal, ami kiszámolja az adott pontszámhoz tartozó érdemjegyet!

A ponthatárok legyenek: – 20: elégtelen, 21 – 30: elégséges, 31 – 50: közepes, 51 – 80: jó, 81 – 100: jeles.

```
x = int(input("írd be a pontszámot: "))
ifx > 80:
print("jeles")
if x > 50 and x <80:
print("jó")
if x > 30 and x <50:
print("közepes")
if x > 20 and x <30:
print("elégséges")
elif x < 20:
print("elégtelen")
```

A fenti példaprogramot érdemes kombinálni egy ciklussal, hogy ne kelljen minden egyes érték megadása utána újra futtatni az alkalmazást. Erre szolgál a python „While” illetve „For” parancsa!

A forráskódok letölthetőek:

- 1. program
- 2. program
- 3. program

- 4. program
- 5. program
- 6. program

A PYTHON PROGRAMOZÁSI NYELV – 3. CIKLUSOK (ITERÁCIÓ) – A WHILE CIKLUS

Írta: K András | 2020.3.24. | Oktatási anyagok, Python, Raspberry Pi |

CIKLUSOK (ITERÁCIÓ) – A WHILE CIKLUS

Az iteráció röviden azt jelenti, hogy a program ugyanazt a kódblokkot újra és újra végrehajtja, a feladattól függő számban, vagy egy logikai feltétel bekövetkeztéig. Az iterációt végrehajtó programozási struktúrát ciklusnak, huroknak nevezzük. A programozásban az iterációnak kétféle típusa van: határozatlan és határozott.

Határozatlan iteráció: a ciklus végrehajtásának száma nincs pontosan előre megadva. A kijelölt blokkot a program többször végrehajtja, mindaddig, amíg bizonyos feltétel(ek) teljesülnek.
Határozott iteráció: a kijelölt programblokk végrehajtásának száma egyértelműen meghatározott már akkor, amikor a ciklus elindul.

A WHILE CIKLUS

A Python „while” ciklus felépítése nagyon egyszerű:

```
while <expr>:
<statement(s)>
```

A „statements” rész tulajdonképpen a létrehozott ciklus magja, az a programblokk, ami meghatározott számban lefut, az „expr” részben szabjuk meg, hogy ez hányszor történjen meg.

Nézzünk egy egyszerű példaprogramot:

```
n = 5
while n > 0:
n -= 1
print (n)
```

Mi történik ebben a példában:

Az n értéke kezdetben 5. A 2. sorban a while állítás vizsgálatána eredménye(n> 0), ami igaz, tehát a ciklusmag lefut. A 3. sorban a ciklusmagban n-t 1-el csökkentjük, majd az eredményt kinyomtatjuk.

Amikor a ciklusmaglefutott, a program végrehajtása visszatér a hurok tetejére a 2. sorba, és a kifejezést újra kiértékeljük. Még mindig igaz, tehát a ciklusmag újra végrehajtja a feladatot, és az eredmény nyomtatásra kerül.

Ez addig folytatódik, amíg n nullává nem válik. Ezen a ponton, amikor a kifejezést teszteljük, az eredmény hamis, így a hurok véget ér.

Fontos, hogy a while hurok a vezérlő logikai kifejezést előbb teszteli, mielőtt bármi más megtörténik. Ha hamis a kiértékelés eredménye, akkor a ciklusmag egyszer sem fut le.

A ciklusmagban szereplő n -= 1 kifejezés jelentése: n értékét csökkentsed eggyel.

```
n = 0
while n > 0:
n -= 1
print (n)
```

A fenti program az előzőek értelmében egyszer sem fut le, mert a kiindulási feltétel (n > 0) hamis.

Nézzünk egy példát „felfelé” számlálásra, amikor a ciklusváltozó értékét egyesével növeljük!

```
count = 0
while (count < 9):
print ('The count is:', count)
count = count + 1
print ("Good bye!")
```

Az első sorban a count változó értékét nullára állítjuk, a while állítás vizsgálatát pedig úgy állítjuk be, hogy a vizsgálat mindaddig igaz legyen, míg a változó kisebb mint 9.

Az eredmény a következő lesz:

```
The count is: 0
The count is: 1
The count is: 2
The count is: 3
The count is: 4
The count is: 5
The count is: 6
The count is: 7
The count is: 8
Good bye!
```

A VÉGTELEN CIKLUS

A ciklus végtelen ciklussá válik, ha avizsgálati részben beállított feltétel soha nem válik hamissá(FALSE). Ezzel a megoldással azonban óvatosan kell eljárni a ciklusok programozása során, mert olyan ciklust állíthatunk elő, amiből a programunk nem tud kilépni.Az ilyen ciklust végtelen ciklusnak hívják.

```
var = 1
while var == 1 : # This constructs an infinite loop
num = int(input("Enter a number :"))
print ("You entered: ", num)
print ("Good bye!")
```

A második sorban beállított logikai feltétel állandóan igaz értéket ad, így a ciklus végtelenné válik. A ciklusból való kilépés egyetlen módja a Ctrl + c billentyűkombináció alkalmazása.

Elegánsabb megoldás, ha figyeljük a bevitt értékeket, és egy előre megadott szám bevitelekor (pl. 0) a „break” paranccsal kilépünk a ciklusból:

```
while True: # This constructs an infinite loop
num = int(input("Enter a number :"))
print ("You entered: ", num)
if num == 0:
break # break here
print ("Good bye!")
```

Hasonló megoldást eredményez a cikluson belül elhelyezett felhasználói döntést kérő vezérlésátadó utasítás, aminek segítségével elegánsan tudunk kilépni a programból:

```
import random
while True:
input("Press enter to roll the dice")
# get a number between 1 to 6
num = random.randint(1,6)
print("You got",num)
option = input("Roll again?(y/n) ")
# condition
if option == 'n':
break
```

A példa kockadobást szimulál. A program első sora importálja a véletlenszám generáláshoz szükséges library-t, ennek a segítségével állítjuk elő az 1-6 közé eső egész számot, ami egy „whileTrue” típusú végtelen cikluson belül mindaddig fut, amíg az „új dobás?” (Roll again?) kérdésre y(es) a válasz. Nem válasz esetén kilépünk a ciklusból.

Az utolsó példaprogram Raspberry Pi-n futtatható, aminek a GPIO portjára három darab LED diódát kapcsolunk, sorban a 11-es, a 13-as, valamint a 15-ös lábra.

A program első két sorában a szükséges library-k importálása történik, az első az időzítés, a második a GPIO port kezelése miatt. A következő négy sorban beállítjuk a tüskesor számozásának módját (BOARD – panel szerint), valamint a három kimeneti tüske (11,13,15) adatirányát (OUT).

A következő három sor inicializálj, alaphelyzetbe állítja a kimeneteket, azaz kikapcsolja az ide kötött LED-eket.

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(11, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.output(11, False)
GPIO.output(13, False)
GPIO.output(15, False)
while True:
GPIO.output(11, GPIO.HIGH)
time.sleep(3)
GPIO.output(13, GPIO.HIGH)
time.sleep(3)
GPIO.output(11, GPIO.LOW)
GPIO.output(13, GPIO.LOW)
GPIO.output(15, GPIO.HIGH)
time.sleep(5)
GPIO.output(15, GPIO.LOW)
GPIO.output(13, GPIO.HIGH)
time.sleep(3)
GPIO.output(13, GPIO.LOW)
```

A forráskódok letölthetőek:

- 1. program
- 2. program
- 3. program
- 4. program
- 5. program
- 6. program

A PYTHON PROGRAMOZÁSI NYELV – 4. CIKLUSOK (ITERÁCIÓ) – A FOR CIKLUS

Írta: K András | 2020.3.26. | Oktatási anyagok, Python, Raspberry Pi |

Az előző leckében bemutatott „while” ciklus mellett a Python a többi nyelvbőljól ismert „for” folyamatvezérlő utasítást is használja, egy kis csavarral.

A Pythonban a for utasítás használata, működése kissé különbözik attól, amit a C vagy a Pascal nyelvben megszokhattunk. Ahelyett, hogy egy változó mindig a számok aritmetikai növekedésével iterálódna (mint például a Pascal-ban), vagy hogy a felhasználó képes legyen meghatározni mind az iterációs lépést, mind a megállási feltételt (C-nyelv), a Python „for” parancsa bármilyen sorozat (lista vagy akár karakterlánc) elemeit annak sorrendjében tudja „bejárni”. Például:

```
words = ['macska', 'kutya', 'pirosalma']
for w in words:
print(w, len(w))
```

A program végigmegy a lista elemein, (macska, kutya, pirosalma), egymás után kinyomtatja az elemeket, illetve minden egyes szó mellé kiírja, hogy hány karakterből állnak. Ezt a „len()” függvénnyel oldja meg, ami az utána zárójelben megadott string karakterszámát adja.

```
>>> len("kutya")
5
```

A „RANGE()” FÜGGVÉNY

Amennyiben az adataink szekvenciálisak, és az iterációt ezen a sorozaton kell végrehajtani, jó szolgálatot tesz a „range()” függvény.

```
for i in range(5):
print(i)
```

Az adott végpont soha nem része a generált sorozatnak. A (10) tartomány 10 értéket generál, a 10-es érték nem lesz a sorozat eleme. Lehetőség van arra, hogy a tartomány egy másik számtól induljon, vagy akár megadhatunk 1-től eltérő növekményt is, ez akár negatív is lehet! A növekményt sokszor lépés-nek hívjuk.

range(kezdőérték, végérték, lépés)

```
range(5, 10)
5, 6, 7, 8, 9
range(0, 10, 3)
0, 3, 6, 9
range(-10, -100, -30)
-10, -40, -70
```

A sorozat elemein történő iteráláshoz a következőképpen kombinálhatjuk a „range ()” és a „len ()” függvényeket:

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
for i in range(len(a)):
print(i, a[i])
```

A futás eredménye:

```
0 Mary
1 had
2 a
3 little
4 lamb
```

BEÁGYAZOTT CIKLUSOK

A beágyazott hurok egy hurok belsejében lévő hurok. A “belső hurok” végrehajtása egyszer történik a “külső hurok” minden egyes iterációja esetén:

```
type = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in type:
for y in fruits:
print(x, y)
```

A futás eredménye:

```
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
```

```
tasty cherry
```

LISTA ALKALMAZÁSA A FOR CIKLUSBAN

A Python lista tulajdonképpen elemek, adatok felsorolása vesszővel elválasztva. Mondhatjuk azt is, hogy tulajdonképpen egy adattömb, ezekről a későbbikben lesz szó. A lista elemei lehetnek karakterek, stringek, számok. Egy lista elemeit könnyen ki tudjuk íratni egy „for” ciklus segítségével:

```
mylist = ["apple", "banana", "cherry"]
for x in mylist:
print(x)
```

A futás eredménye:

```
apple
banana
cherry
```

Amennyiben a lista pl. számokat tartalmaz:

```
mylist = [13, 22, 44, 55]
for x in mylist:
print(x)
```

Az futás eredménye:

```
13
22
44
55
```

Megfigyelhető, hogy a „for” ciklus és a lista kombinálásával kinyomtatott számsor elemei között nem egyenletes a növekmény.

A lista elemeinek egymás után történő kinyomtatása megoldható a „for” ciklus és a „range()” függvény használatával:

```
mylist = [13, 22, 44, 55, 65, 92]
for x in range(2,5):
print(mylist[x])
```

Az eredmény:

```
44
55
65
```

Ezt a megoldást használhatjuk fel például a Raspberry Pi GPIO portjára kötött LED-ek futófény szerű működtetésére. A kimeneti tüskék sorszámai nem egymás után következő értékek, de ha a pin sorszámokat egy listában adjuk meg, akkor könnyen megoldható a feladat:

Kapcsoljunk LED diódákat a Raspberry Pi 7, 11, 13, 15-ös kimeneteire. Ha sorban szeretnénk ki/be kapcsolgatni a kimeneteket, akkor célszerű a kimeneteket egy lista elemeiként definiálni:

```
gpio_list = [7,11,13,15]
```

A teljes program ezek után:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)
GPIO.setup(13,GPIO.OUT)
GPIO.setup(15,GPIO.OUT)
gpio_list = [7,11,13,15]
for i in range (0,4):
GPIO.output((gpio_list[i]),GPIO.HIGH)
time.sleep(0.5)
GPIO.output((gpio_list[i]),GPIO.LOW)
time.sleep(0.5)
GPIO.cleanup()
```

A fenti program egyszer fut le. Ha azt szeretnénk, hogy a futófény folyamatos legyen, alkalmazhatjuk a „while” ciklust:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7,GPIO.OUT)
GPIO.setup(11,GPIO.OUT)
GPIO.setup(13,GPIO.OUT)
GPIO.setup(15,GPIO.OUT)
gpio_list = [7,11,13,15]
while True:
for i in range (0,4):
GPIO.output((gpio_list[i]),GPIO.HIGH)
time.sleep(0.5)
GPIO.output((gpio_list[i]),GPIO.LOW)
time.sleep(0.5)
GPIO.cleanup()
```

A forráskódok letölthetőek:

- 1. program

- 2. program
- 3. program
- 4. program
- 5. program
- 6. program

A PYTHON PROGRAMOZÁSI NYELV – 5. – FÜGGVÉNYEK

Írta: K András | 2020.3.30. | Oktatási anyagok, Python, Raspberry Pi |

A függvény meghatározása nagyon egyszerű: ez egy kis kódrészlet, ami valami specifikus feladatot lát el. A függvény az utasítások olyan blokkja, ami végrehajtja a benne meghatározott művelet(ek)et, és miután lefutott, tetszőlegesen újra hívható, futtatható. A függvények modulárisabbá, áttekinthetőbbé teszik a kódot, lehetővé téve ugyanazon kódrészlet (függvény) újra és újra használatát.

A Python számos beépített függvénnyel rendelkezik, amikkel már megismerkedtünk, többek között:

- print (), ami adatokat nyomtat a terminálra
- int (), ami egy karakterlánc vagy szám adattípust egész adattípussá konvertál
- len (), ami egy objektum (pl. string) hosszát adja vissza

A függvénynevek zárójeleket tartalmaznak, és tartalmazhatnak paramétereket.

Ebben a leckében azt vizsgáljuk, meg hogy, hogyan készíthetünk saját függvényeket, amiket a kódolási projektjeinkben használhatunk.

A FÜGGVÉNYEK LÉTREHOZÁSA A DEF KULCSSZÓVAL TÖRTÉNIK, A KÖVETKEZŐ MÓDON:

```
def fuggveny(param1):
"""Ez egy függvény"""
# ... jelenleg nem sokat csinál
print "semmi"
return None
```

A def kulcsszót a függvény neve követi, majd utána zárójelben a függvénynek átadandó paramétereket soroljuk fel, jelen esetben ez csak egy, param1 nevű paramétert jelent. Ezután megadhatunk a függvényben egy ún. doc-stringet, amiben leírhatjuk pár szóval, hogy mit csinál, majd megírjuk a függvény törzsét. A függvény visszatérésekor a példánkban a „None” értéket adja.

Első példánk legyen egy klasszikus „Hello, World” függvény! Az első lépés a függvény létrehozása:

```
def hello():
```

Egyelőre a függvény neve után álló zárójelek üresek. A függvény létrehozása után következik az a kódrészlet, ami majd a függvény hívásakor lefut:

```
def hello():
print("Hello, World!")
```

A függvényünk elkészült, ha ezek után beillesztjük egy programba, akkor még semmit nem fog csinálni. Ahhoz, hogy működjön, meg kell hívnunk, ami a nevével történik:

```
def hello():
print("Hello, World!")
hello()
```

Ez a program még tényleg nem csinál sok hasznosat, a következővel azonban ki tudjuk számoltatni pl. egy háromszög kerületét:

```
def triangle(a,b,c):
print("perimeter: ",a+b+c)
triangle(3,4,5)
```

Sokkal rugalmasabb a program, ha az oldalhosszakat mi adhatjuk meg!

```
def triangle(a,b,c):
print("ker: ",a+b+c)
a = input("a oldal: ")
b = input("b oldal: ")
c = input("c oldal: ")
triangle(a,b,c)
```

A program futtatása után a következő, elsőre meglepő eredményt kapjuk:

```
a oldal: 5
b oldal: 6
c oldal: 7
ker: 567
```

A meglepő eredményt a hibás változó típus alkalmazása okozza, a program string-ként dolgozza fel az értékeket. A program helyesen:

```
def triangle(a,b,c):
print("ker: ",a+b+c)
a = float(input("a oldal: "))
b = float(input("b oldal: "))
c = float(input("c oldal: "))
```

```
triangle(a,b,c)
```

A fenti program a következő eredmény produkálja, ami már helyes:

```
a oldal: 5
b oldal: 6
c oldal: 7
ker: 18.0
```

A „RETURN” KUKLCSSZÓ ÉS A VISSZATÉRÉSI ÉRTÉK

A „return” kulcsszót arra használhatjuk, hogy segítségével kilépjünk a függvényből, és a főprogram azon pontjára kerüljünk vissza, ahol a hívás történt. A „return” tartalmazhat visszatérési értéket is, ami például egy számítás eredménye lehet.

A következő példa egy olyan függvény, ami a megadott szám abszolút értékét számolja ki, majd visszatérve a hívás helyére az eredményt megjeleníti a konzolon:

```
def absolute_value(num):
"""This function returns the absolute
value of the entered number"""
if num >= 0:
return num
else:
return -num
num = float(input("enter a number: "))
print(absolute_value(num))
```

A program futási eredménye:

```
enter a number: -12.6
12.6
```

Egy másik példa két szám összegzését végzi:

```
def add_numbers(x,y):
sum = x + y
return sum
num1 = float(input("num1: "))
num2 = float(input("num2: "))
print("The sum is", add_numbers(num1, num2))
```

Utolsó példánk egy négy alapműveletet végrehajtó „ számológép”, ami további függvények hozzáadásával bővíthető!

```
def add_numbers(x,y):
```

```
sum = x + y
return sum
def subt_numbers(x,y):
subt = x - y
return subt
def multi_numbers(x,y):
multi = (x*y)
return multi
def div_numbers(x,y):
div = (x/y)
return div
num1 = float(input("num1: "))
num2 = float(input("num2: "))
op = int(input("operation: 1:sum, 2:subtract, 3:multiplication, 4:divison "))
if op == 1:
print("The sum is", add_numbers(num1, num2))
elif op == 2:
print("The subt. is", subt_numbers(num1, num2))
elif op == 3:
print("The multi. is", multi_numbers(num1, num2))
elif op == 4:
print("The div. is", div_numbers(num1, num2))
else:
print("unknown operation")
```

FÜGGVÉNYEK:

A függvények olyan utasításkód blokkok, amelyek egy programon belül hajtanak végre műveleteket, elősegítve a kód újrafelhasználhatóságát és moduláris használatát.

A forráskódok letölthetőek:

- 1. program
- 2. program
- 3. program
- 4. program
- 5. program
- 6. program
- 7. program

A PYTHON PROGRAMOZÁSI NYELV – 6. – A PYTHON TURTLE (TEKNŐS) KÖNYVTÁR

Írta: K András | 2020.4.3. | Oktatási anyagok, Python, Raspberry Pi |

A „Turtle” aPythonelőre telepített könyvtára, ami lehetővé teszi a felhasználók számára képek és rajzok létrehozását úgy, hogy egy virtuális rajzfelülettel látja el a Python környezetet. A rajzhoz használt képernyő-tollat teknősnek hívják, innen a könyvtár neve. Ennek az az oka, hogy a Ateknős-könyvtár interaktív módon segít a kezdő programozóknak tesztelni a különböző alapvető programozási alapszerkezeteket mint pl. a ciklus, elágazás, eljárás.

A teknősbéka grafika kapcsán meg kell említenünk Seymour Papert nevét, aki matematikus a Massachusetts Institute of Technology-n, számítógéptudós, pszichológus és kiváló pedagógus volt. A mesterséges intelligencia kutatása úttörőinek és a Logo programozási nyelv megalkotóinak egyike.

A könyvtár elemeinek használata nagyon egyszerű, a parancsok felépítése egységes, mindegyik a turtle névvel kezdődik, majd ponttal elválasztva a konkrét parancs és a paraméterek.

```
turtle.right(90)
turtle.forward(100)
```

A leggyakrabban használt parancsokat az alábbi táblázatban láthatod:

PARANCS	PARAMÉTER	LEÍRÁS
Turtle()	nincs	Új „turtle” objektumot hoz létre és ad vissza
forward()	érték	A teknős előrehalad a megadott értékkel
backward()	értékt	A teknős hátrafelé mozog a megadott értékkel
right()	szög	A teknős az óramutató járásával megegyező irányban fordul
left()	szög	A teknős az óramutató járásával ellenkező irányba fordul
penup()	Nincs	Felemeli a teknős tollát
pendown()	Nincs	Leteszi a teknős tollát
up()	Nincs	Felveszi a teknős tollát

PARANCS	PARAMÉTER	LEÍRÁS
down()	Nincs	Leteszi a teknős tollát
color()	szín	Megváltoztatja a teknős toll színét
fillcolor()	szín neve/kódja	Megváltoztatja a teknős színét a sokszög kitöltéséhez
heading()	Nincs	Visszaadja a teknős aktuális irányát
position()	Nincs	Visszaadja az aktuális helyzetet
goto()	x, y	A teknőcöt az x, y pozícióba viszi
begin_fill()	Nincs	A színkitöltés kezdőpontja
end_fill()	Nincs	Lezárja a sokszöget, és kitölti az aktuális színnel
dot()	Nincs	Adott méretű és színű pontot rajzol az aktuális pozícióba
stamp()	Nincs	A teknős lenyomatát hozza létre a jelenlegi helyen
shape()	forma neve	A név lehet: ‘arrow’, ‘classic’, ‘turtle’ or ‘circle’

A turtle graphic programok elején egységesen a következő lépéseket kell elvégezni:

– könyvtár importálása:

```
import turtle
```

– rajzolóablak létrehozása

```
ablak = turtle.Screen()
```

– rajzoló teknős létrehozása:

```
fred = turtle.Turtle()
```

Nem kötelező, hasznos további parancsok:

– háttérszín beállítása:

ablak.bgcolor(„white”) - a parancs paramétere lehet színnév, vagy kód

– toll szín beállítás:

```
fred.pencolor(„black”)
```

– rajzolóablak méretezése:

```
ablak.setup (400,400)
```

– ablak címke beállítása:

```
ablak.title(„címke”)
```

Az alapokat összerakva, a programjaink első pár sora hasonlóan fog kinézni az alábbi példához:

```
import turtle
ablak = turtle.Screen()
ablak.setup(400,400)
ablak.bgcolor('white')
ablak.title("program_1")
fred = turtle.Turtle()
```

Teknősünk neve ’fred’ az ablaké pedig ’ablak’. Ezek az elnevezések szabadon választhatók, a többi elnevezés nem módosítható!

Első példaprogramunk egy 100 pixel oldalhosszúságú négyzetet rajzol:

```
import turtle
ablak = turtle.Screen()
ablak.setup(400,400)
ablak.bgcolor('white')
ablak.title("program_1")
fred = turtle.Turtle()
```

```
fred.fd(100)
fred.left(90)
fred.fd(100)
fred.left(90)
fred.fd(100)
fred.left(90)
fred.fd(100)
fred.left(90)
ablak.exitonclick()
```

Látható, hogy a rajzoló rész négyszer ismétli ugyan azokat a lépéseket. Ezt érdemes egy ciklussal megoldani!

A ciklus „fej” része bevezet egy ’i’ ciklusváltozót, és a range beállítja a négyszeri ismétlést. A további két sor az ún. ciklusmag, ami négyszer ismétlődik:

```
import turtle
ablak = turtle.Screen()
ablak.setup(400,400)
ablak.bgcolor('white')
ablak.title("program_1")
fred = turtle.Turtle()
for i in range(4):
fred.fd(100)
fred.left(90)
ablak.exitonclick()
```

Természetesen nem csak négyzetet tudunk a ciklussal rajzolni, hanem például ötágú csillagot is:

```
import turtle
ablak = turtle.Screen()
ablak.setup(400,400)
ablak.bgcolor('white')
ablak.title("program_3")
fred = turtle.Turtle()
for i in range(5):
fred.fd(100)
fred.right(144)
ablak.exitonclick()
```

A színek változtatásával érdekes geometriai ábrákat tudunk létrehozni! Ehhez a programhoz érdemes a háttérszínt feketére állítani (

```
ablak.bgcolor('black')
```

), illetve a rajzolás színét folyamatosan módosítani. Ehhez használhatjuk a python ’list’ függvényét, aminek segítségével a pencolor utasítás a listából veheti a színek megnevezését:

```
colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow']
```

Annak érdekében, hogy a színeket sorban vegye a program a listából, alkalmazzuk az „i%6” műveletet, ami az ún. modulo függvény. Ez az i aktuális értékét osztja 6-tal, és a maradékot adja eredményként. Ez rendre 0,1,2,3,4,5.

```
import turtle
ablak = turtle.Screen()
ablak.setup(400,400)
ablak.bgcolor('black')
ablak.title("program_4")
fred = turtle.Turtle()
colors = ['red', 'purple', 'blue', 'green', 'orange', 'yellow']
t = turtle.Pen()
for i in range(180):
t.pencolor(colors[i%6])
t.width(i/100 + 1)
t.forward(i)
t.left(59)
ablak.exitonclick()
```

A lecke utolsó mintaprogramja koordináta rendszert rajzol. A program négy ciklusból áll, ezek egymás után rajzolják meg a négy síknegyed tengelyeit, és készítik el azok beosztását. Új parancs a programban a fred.write („ „) függvény, ami az adott pozícióba írja az idézőjelek közötti stringet.

```
import turtle
ablak = turtle.Screen()
ablak.setup(600,600)
ablak.bgcolor('white')
ablak.title("Coordinates")
fred = turtle.Turtle()
fred.pencolor("black")
fred.speed(0)
#tengelyek - beosztás
for x in range (12):
fred.lt(90)
fred.fd(20)
fred.rt(90)
fred.fd(5)
fred.bk(5)
fred.penup()
fred.goto(0,0)
for x in range (12):
fred.pendown()
fred.fd(20)
fred.lt(90)
fred.fd(5)
fred.bk(5)
```

```
fred.rt(90)
fred.penup()
fred.goto(0,0)
for x in range (12):
fred.pendown()
fred.bk(20)
fred.lt(90)
fred.fd(5)
fred.bk(5)
fred.rt(90)
fred.penup()
fred.goto(0,0)
for x in range (12):
fred.pendown()
fred.rt(90)
fred.fd(20)
fred.lt(90)
fred.fd(5)
fred.bk(5)
fred.penup()
fred.goto(250,0)
fred.write("x")
fred.goto(0,250)
fred.write("y")
fred.goto(0,0)
ablak.exitonclick()
```

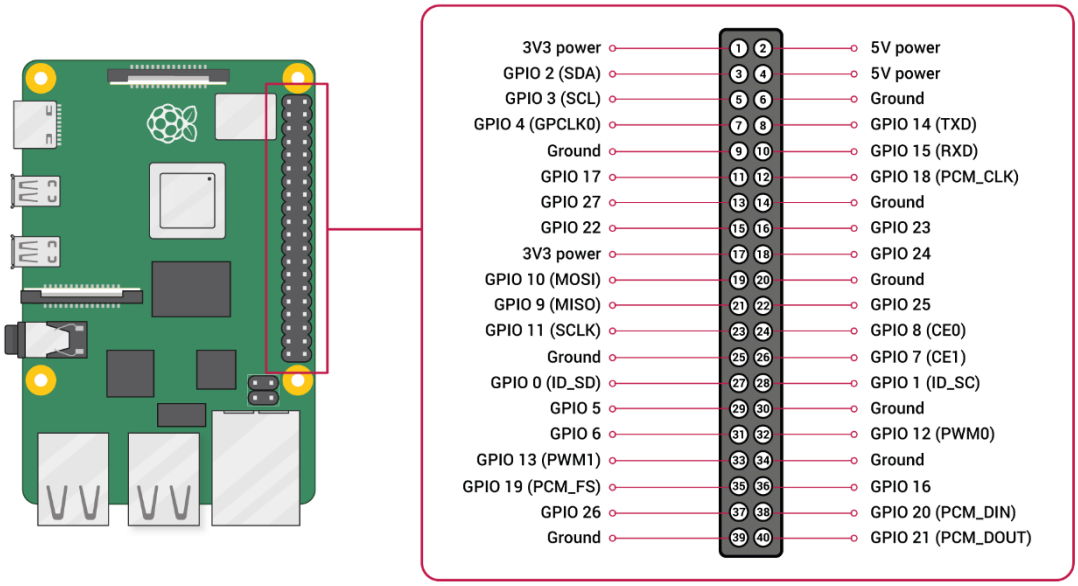
A forráskódok letölthetőek:

- 1. program
- 2. program
- 3. program
- 4. program
- 5. program

A PYTHON PROGRAMOZÁSI NYELV – 7. – HARDVER KÖZELI PROGRAMOZÁS – 1.

Írta: K András | 2020.4.14. | Oktatási anyagok, Python, Raspberry Pi |

A Raspberry Pi-t több tulajdonsága miatt is kiemelkedő képességű eszköznek tarthatjuk, a kis méret, az alacsony ár, a kiváló támogatottság mind-mind szuper tulajdonságok. Az ún. „maker”, „DIY”, illetve az otthoni vagy akár hivatásos fejlesztő szempontból az egyik legfontosabb tulajdonság az ún. GPIO (általános célú bemenet / kimenet) tüskék sora a panel felső széle mentén. A 40 tűs GPIO csatlakozó megtalálható az összes jelenlegi Raspberry Pi verzión. (A Pi 1 B + (2014) modell előtt a panelek rövidebb, 26 tűs csatlakozót tartalmaztak.)



GPIO interfész lábkiosztása

A tüskék között 17 olyan van, amiket különböző programozási környezetekből (Scratch, Python, C/C++) könnyen tudunk programozni. A többi tüske ún. dedikált kimenet, azaz konkrét funkciót rendeltek hozzájuk a Pi tervezői.

Dedikált tüskék:

PWM (pulse-width modulation – impulzus szélességmoduláció)

Szoftveres PWM az összes tüskén

Hardveres PWM: GPIO12, GPIO13, GPIO18, GPIO19

SPI

SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)

SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)

I2C

Data: (GPIO2); Clock (GPIO3)

EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)

Soros port

TX (GPIO14); RX (GPIO15)

A RASPBERRY GPIO PORTJÁNAK PROGRAMOZÁSA PYTHON NYELVEN

A port programozására jelenleg három ún. library segítségével van mód. Az első, a Gordon Henderson által fejlesztett Wiring Pi könyvtár (http://wiringpi.com), illetve két, többé-kevésbé hivatalosnak tekinthető modul, az RPi.GPIO illetve a legutóbbi kiadású GPIO Zero.

Amikor választunk a három elérhető modul közül, akkor a konkrét feladatot kell figyelembe vennünk, illetve azt, hogy a két hivatalos könyvtár fejlesztése folyamatosabb, jobban illeszkedik az éppen aktuális verzió hardver tulajdonságaihoz. Feltétlenül érdemes megemlíteni azonban, hogy a WiringPi tartalmaz egy gpio parancssori segédprogramot, ami felhasználható a GPIO tüskék direkt programozására és beállítására. Ezt használhatjuk a tüskék olvasására és írására, és akár parancsfájlokból történő vezérlésére is.

A következőkben a két, Raspberry Pi alapítvány által támogatott, illetve fejlesztett library használatát mutatjuk be.

1. AZ RPI.GPIO MODUL HASZNÁLATA

IMPORTÁLÁS

A library használatát importálásával kezdjük, ezt a műveletet célszerű összevonni egy ún. alias név adásával:

```
import RPi.GPIO as GPIO
```

Ilyen módon csak a „GPIO” névrekell hivatkoznunk a program további részében. A modul importálása egybevonva egy ellenőrzéssel a következőképpen történhet:

```
try:
import RPi.GPIO as GPIO
except RuntimeError:
print("Error importing RPi.GPIO!)
```

KIMENET SZÁMOZÁS

Az RPi.GPIO használatakor kétféle módon lehet számozni az IO-tüskéket. Az első az ún. BOARD számozási rendszer használata. Ez a Raspberry Pi panel P1 csatlakozóján található fizikai PIN-sorszámokra utal. Ennek a számozási rendszernek az az előnye, hogy nem kell ismernünk az RPiprocesszorának lábkiosztását, illetve nem kell változtatnunk a bekötésen ha esetleges hardveres fejlesztés történik.

A második számozási rendszer a BCM számozás. Ez egy alacsonyabb szintű működési mód – a Broadcom SOC (systemon chip, egylapkás rendszer) lábkiosztására utal. Ez annyival bonyolultabb, mint a BOARD számozás, hogy itt ismernünk kell az alkalmazott lapka lábkiosztását, és a sorszámok nem növekvő sorrendben következnek.

Annak beállítása, hogy programunkban melyik megoldást használjuk az a következő két paranccsal oldható meg:

```
GPIO.setmode(GPIO.BOARD)
```

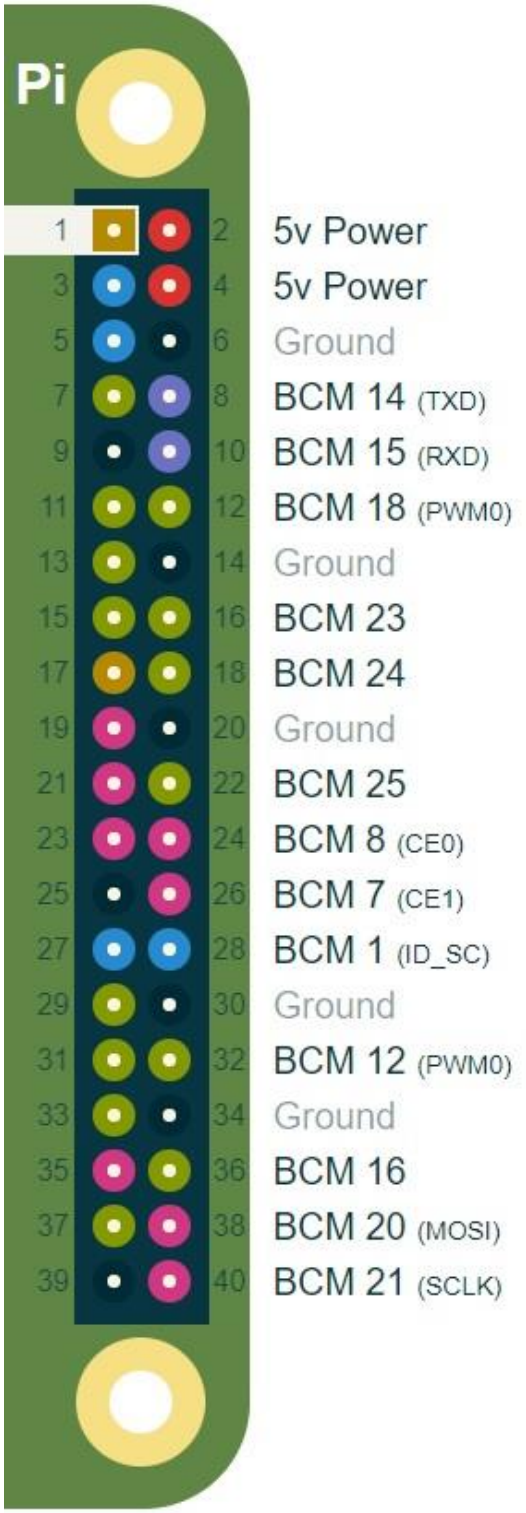
vagy:

```
GPIO.setmode(GPIO.BCM)
```

A beállított mód értéke le is kérdezhető:

```
mode = GPIO.getmode()
```

A „mode”értéke: GPIO.BOARD, GPIO.BCM vagy None



FIGYELMEZTETÉSEK

Lehetséges, hogy egynél több programot futtatunk a Pi-n, illetve többször újra indíthatjuk ugyan azt a programunkat. Ennek eredményeként, ha az RPi.GPIO azt észleli, hogy egy csatornát korábban használtunk, és az esetleg nincs „felszabadítva”, vagy, hogy ha egy kimenetet az alapértelmezetthez képest eltérő módon konfiguráltunk, akkor figyelmeztető jelzést kapunk. A figyelmeztetések letilthatók a következő paranccsal:

```
GPIO.setwarnings(False)
```

TÜSKE (PIN) BEÁLLÍTÁSA:

Minden egyes tüskéthasználata előtt be kell állítanunk, hogy kimenetként, vagy bemenetként szeretnénk alkalmazni az aktuális programban.

Beállítás bemenetként:

```
GPIO.setup (ch, GPIO.IN)
```

(ahol a „ch” a csatorna száma a megadott számozási rendszer alapján (BOARD vagy BCM)).

Csatorna beállítása kimenetként:

```
GPIO.setup (ch, GPIO.OUT)
```

(ahol a „ch” a csatorna száma a megadott számozási rendszer alapján (BOARD vagy BCM)).

Megadható a kimeneti csatorna kezdeti értéke is:

```
GPIO.setup (ch, GPIO.OUT, initial = GPIO.HIGH)
```

TÖBB CSATORNA BEÁLLÍTÁSA

Egyszerre több csatorna beállítására is lehetőség van, ha alkalmazzuk a Python lista (list) parancsát. Például:

Például:

```
chan_list = [11,12] # a listához tetszőleges számú további csatorna hozzáadható!
GPIO.setup (chan_list, GPIO.OUT)
```

BEMENET

A GPIO tüske értékének beolvasása:

```
GPIO.input (ch)
```

(ahol a ch a csatorna száma a megadott számozási rendszer alapján (BOARD vagy BCM)). Ez a parancs a 0 / GPIO.LOW / False vagy az 1 / GPIO.HIGH / True értéket adja vissza.

KIMENET

A GPIO tüske kimeneti állapotának (logikai szintjének) beállítása:

```
GPIO.kimenet (ch, szint)
```

(ahol a ch a csatorna száma a megadott számozási rendszer alapján (BOARD vagy BCM)).

A szint lehet 0 / GPIO.LOW / False vagy 1 / GPIO.HIGH / True.

Az egyes szintekhez a kimeneten 0 illetve 3,3V tartozik.

TÖBB CSATORNA BEÁLLÍTÁSA EGY PARANCCSAL

Lehetőségünk van több csatorna alapértékének egy paranccsal történő beállítására. Például:

```
chan_list = [11,12] # kimenetek beállítása a list paranccsal
GPIO.output (chan_list, GPIO.LOW) # az összes kimenetet alapértelmezetten alacsony szintre állítja
GPIO.output (chan_list, (GPIO.HIGH, GPIO.LOW)) # az első csatorna magas a második alacsony szint
```

ERŐFORRÁSOK FELSZABADÍTÁSA

Bármely program végén jó gyakorlat az esetlegesen használt erőforrások felszabadítása, a csatornák elengedése, azaz a programban meghatározott hozzárendelések megszüntetése. Ez így van az RPi.GPIO modul használatánál is. Ezzel a módszerrel elkerülhetjük, hogy a kimenetek a program leállítása után magas szinten maradjanak, és egyben megvédhetjük a Pi kimeneteit is. Fontos, hogy a parancs csak azokra a kimenetekre van hatással, amiket használtunk a futtatás során, viszont kiadásával töröljük a számozás (BOARD / BCM) beállítást is!

A parancsot több formában is használhatjuk:

```
GPIO.cleanup () # a program által használt összes csatorna felszabadítása
GPIO.cleanup(channel) # adott számú csatorna felszabadítása
GPIO.cleanup( [channel1, channel2] ) # a listában megadott számú csatornák felszabadítása
```

ALAPLAP ÉS GPIO INFORMÁCIÓK

Az alábbi parancsokkal lekérdezhető a Raspberry Pi panel, illetve a az RPi.GPIO verziója:

Az RPi-vel kapcsolatos információk felfedezése:

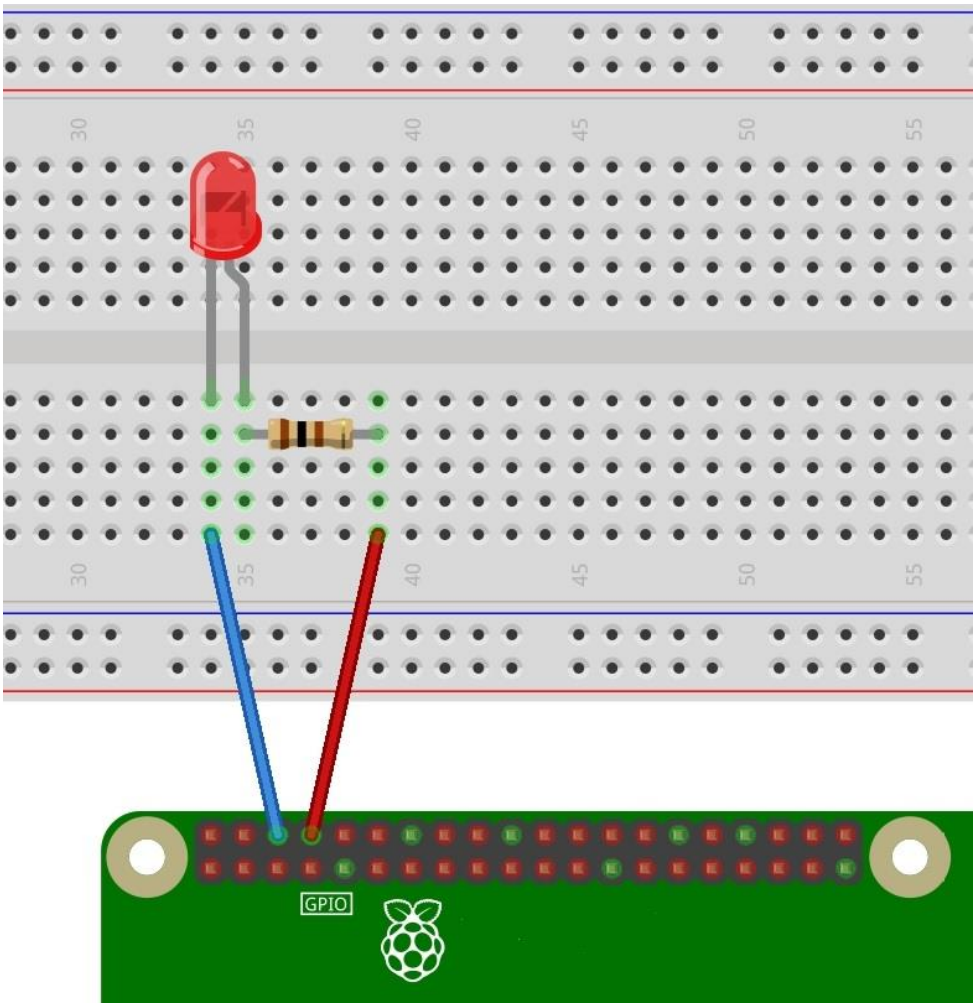
```
GPIO.RPI_INFO
GPIO.RPI_INFO['P1_REVISION']
```

GPIO HELLO WORLD

Végül nézzük meg a fentieket egy egyszerű példában összefoglalva, amit nevezhetünk a GPIO Hello World programjának!

A programhoz készítsük el az alábbi egyszerű kapcsolást, ami a BOARD számozás szerinti 8-as kimenetet használja egy LED villogtatására!

A kapcsolást breadboard-on készítsük el!



Egyszerű LED villogtató áramkör breadboardon

Az alkalmazott ellenállás 100 – 330 ohm közötti értékű legyen, a LED negatív kivezetését pedig a 6-os lábra kössük!

A MŰKÖDTETÉSRE HASZNÁLHATÓ PROGRAM:

– WHILE ciklussal:

```
import RPi.GPIO as GPIO # az RPi.GPIO könyvtár importálása, és az alias létrehozása

from time import sleep # az időzítéshez szükséges time library importálása

GPIO.setwarnings(False) # figyelmeztetések kikapcsolása (nem javasolt)

GPIO.setmode(GPIO.BOARD) # számozási mód beállítása

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # 8-as láb beállítása kimenetnek, kezdeti alacsony szint

while True: # végtelen ciklus indítása

GPIO.output(8, GPIO.HIGH) # 8-as kimenet magas logikai szintre állítása

sleep(1) # 1 másodperc várakozás

GPIO.output(8, GPIO.LOW) # 8-as kimenet alacsony logikai szintre állítása
```

```
sleep(1) # 1 másodperc várakozás
```

– FOR ciklussal:

```
import RPi.GPIO as GPIO # az RPi.GPIO könyvtár importálása, és az alias létrehozása

from time import sleep # az időzítéshez szükséges time library importálása

GPIO.setwarnings(False) # figyelmeztetések kikapcsolása (nem javasolt)

GPIO.setmode(GPIO.BOARD) # számozási mód beállítása

GPIO.setup(8, GPIO.OUT, initial=GPIO.LOW) # 8-as láb beállítása kimenetnek, kezdeti alacsony szint

for i in range (10): # for ciklus indítása

GPIO.output(8, GPIO.HIGH) # 8-as kimenet magas logikai szintre állítása

sleep(1) # 1 másodperc várakozás

GPIO.output(8, GPIO.LOW) # 8-as kimenet alacsony logikai szintre állítása

sleep(1) # 1 másodperc várakozás
```

A forráskódok letölthetőek:

- 1. program: LED villogtatás while ciklussal
- 2. program: LED villogtatás for ciklussal

A PYTHON PROGRAMOZÁSI NYELV – 8. – HARDVER KÖZELI PROGRAMOZÁS – 2.

Írta: K András | 2020.4.20. | Oktatási anyagok, Python, Raspberry Pi |

Ahogy az előző fejezetben megbeszéltük, a Raspberry Pi GPIO portjának programozására több Python könyvtár is rendelkezésre áll. Az RPi.GPIO mellett a másik hivatalosnak tekinthető modul a *GPIO Zero* 1 . A könyvtár nagyon friss, legutóbbi, v1.5.0-ás verziójáról a raspberrypi.org oldalon, egy 2019. februárjában megjelent cikkben olvashatunk (GPIO Zero v1.5 is here! 13th Feb 2019 Ben Nuttall) https://www.raspberrypi.org/blog/gpio-zero-v1-5/
A könyvtár sok egyszerűsítést tartalmaz az RPi.GPIO modulhoz képest, ennek ellenére nem lehet rangsort felállítani az egyes library-k között, újra érdemes hangsúlyozni, hogy a könyvtár kiválasztást mindig az adott feladat határozza meg. A Zero modul kiváló segédlet abban az esetben, ha gyorsan szeretnénk tesztelni, kipróbálni, fejleszteni egy alkalmazást. Egy javaslat, a hardver illetve a szoftver alapos(abb) megismerésének érdekében érdemes a GPIO programozást az RPi.GPIO modullal kezdeni.

1. A GPIO ZERO TELEPÍTÉSE

A GPIO Zero alapértelmezetten telepítve van a Raspbian lemezképen, csakúgy, mint a Raspberry Pi Desktop PC / Mac lemezképen is, mindkettő elérhető a raspberrypi.org weboldalon. Amennyiben a Raspbian Lite változatot használjuk, akkor a telepítés az alábbiak szerint végezhető el:

Először frissítsük az adattárak listáját (repositories list):

```
pi@raspberrypi: ~ $ sudo apt update
```

Ezután telepítsük a Python 3 csomagot:

```
pi@raspberrypi: ~ $ sudo apt install python3-gpiozero
```

Ha esetleg másik operációs rendszert használunk a Raspberry Pi-n, akkor a GPIO Zero telepítéséhez a „pip” használatára lesz szükség. Telepítsük a „pip”-et a get-pip használatával, majd írjuk be:

```
pi@raspberrypi:~$ sudo pip3 install gpiozero
```

2. ALAPPÉLDÁK

Az alábbi egyszerű példák bemutatják a GPIO Zero könyvtár használatának néhány lehetőségét. Fontos: az összes „recept” a Python 3 verzióhoz készült! A példák a Python 2 alatt is működhetnek,de ez nem garantált!

2.1. GPIO ZERO IMPORTÁLÁSA

A Python környezetben a programokban használt könyvtárakat és funkciókat név szerint kell importálni a fájl elején, kivéve az alapértelmezés szerint beépített modulokat.

Például ha csak a Button eljárást szeretnénk használni a GPIO Zero könyvtárból, akkor az import parancsot a következőképpen használjuk:

```
from gpiozero import Button
```

Ezek után a Button elem közvetlenül elérhető a programban:

```
button = Button(2)
```

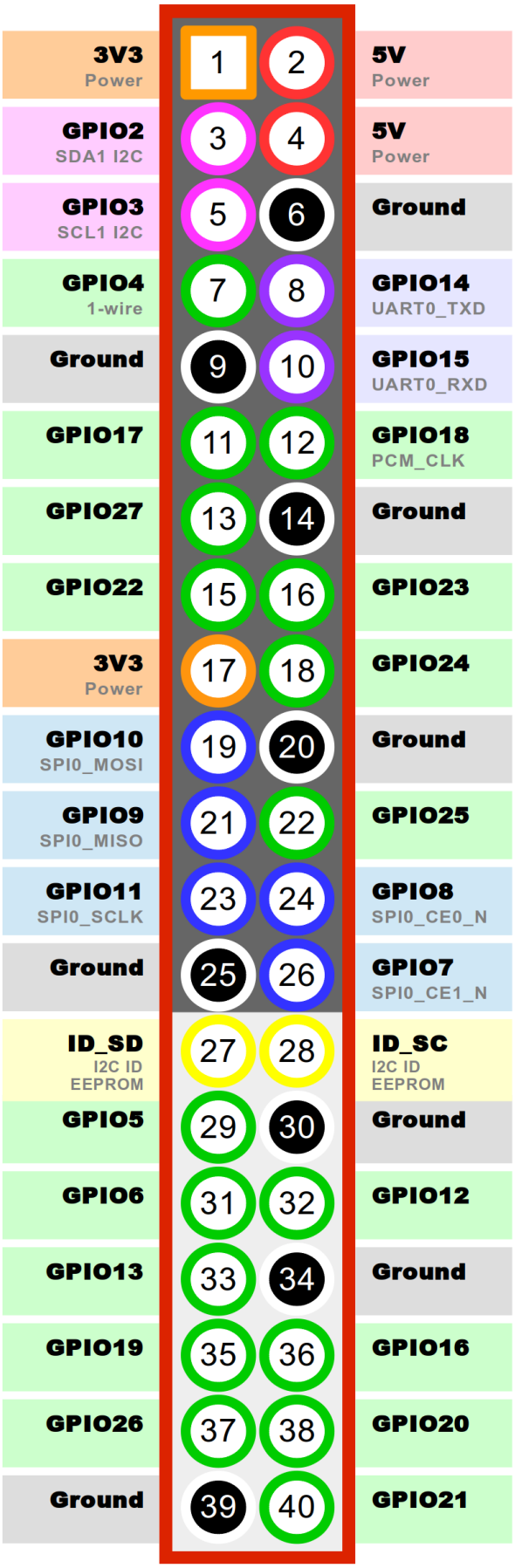
Természetesen, ha nincs okunk arra, hogy csak bizonyos elemeket importáljunk, (pl. szűkös memóriakeret) akkor célszerű a teljes modult importálni:

```
import gpiozero
```

Ebben az esetben a fenti Button hívás a következőképpen módosul:

```
button = gpiozero.Button(2)
```

2.2. PIN SZÁMOZÁS



Raspberry Pi 40 Pin Header

Ez a könyvtár a Broadcom (BCM) pin számozást használja a GPIO ki- illetve bemenetekre, szemben a fizikai (BOARD) számozással. Az RPi.GPIO könyvtárral ellentétben ez nem konfigurálható. A két számozás közötti átmenet egy kis trükkel megoldható!

Bármely, az alábbi ábrán „GPIO" jelzéssel ellátott csatlakozó használható PIN-sorszámmal. Például, ha egy LED-et csatlakoztatunk a "GPIO17" – kimenethez, akkor a PIN-kódot 17 helyett 11-re (BOARD számozás) állíthatjuk a következő paranccsal: BOARD11.

Hasonlóan használhatjuk azt a parancsot is, ami a Pi3, Pi4 verziók csatlakozósáv jelölésére hivatkozik, ahol a GPIO port a J8 ("header:number"). Tehát a „J8: 11" jelentése: a J8 csatlakozó 11. fizikai tüskéje.

A fentiek értelmében az alábbi parancsok egyenértékűen használhatóak:

```
>>> led = LED(17)
>>> led = LED("GPIO17")
>>> led = LED("BCM17")
>>> led = LED("BOARD11")
>>> led = LED("WPI0") # wiring pi szerinti számozás
>>> led = LED("J8:11")
```

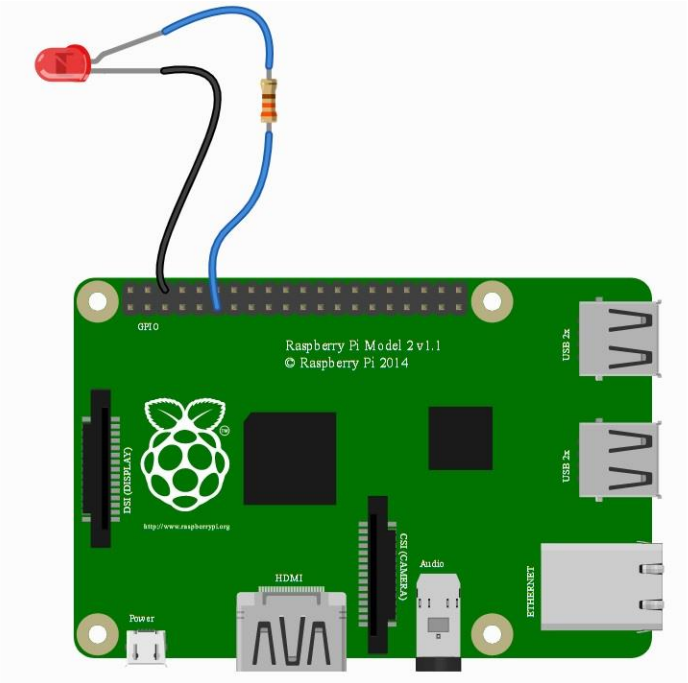
Vegyük figyelembe, hogy ezek a hozzárendelések csak az adott programon belül érvényesek, így ha egy kimenet állapotát lekérdezzük a programunkon belül, az „eredmény" mindig a Broadcom (BCM) számozás szerinti érték lesz:

```
>>> led = LED("BOARD11")
>>> led
<gpiozero.LED object on pin GPIO17, active_high=True, is_active=False>
```

2.3. LED-VILLOGTATÁS

Kezdjük a GPIO Zero könyvtár kipróbálását is a „kötelező" Hello World programmal, a led-villogtatással. Használjuk a 11-es tüskét (GPIO17)!

A fizikai elrendezés a következő:



A program:

```
from gpiozero import LED # a könyvtár importálása
from time import sleep # az időzítés importálása
red = LED(17) # az alkalmazott tüske beállítása
while True: # while ciklus fejléc
red.on() # led bekapcsolás
sleep(1) # várakoztatás
red.off() # led kikapcsolás
sleep(1) # várakoztatás
```

A könyvtár egyszerűsítési lehetőségeit kihasználva a programot az alábbi módon is megírhatjuk:

```
from gpiozero import LED
from signal import pause
red = LED(17)
red.blink()
pause()
```

2.4. VÁLTOZÓ FÉNYERŐSSÉGŰ LED

Bármelyik hagyományos LED fényerejét PWM (impulzus-szélesség-moduláció) segítségével beállíthatjuk. A GPIO Zero könyvtárban ez a PWMLED parancs használatával érhető el 0 és 1 közötti értékek megadásával:

```
from gpiozero import PWMLED
from time import sleep
led = PWMLED(17)
while True:
led.value = 0 # ki
```

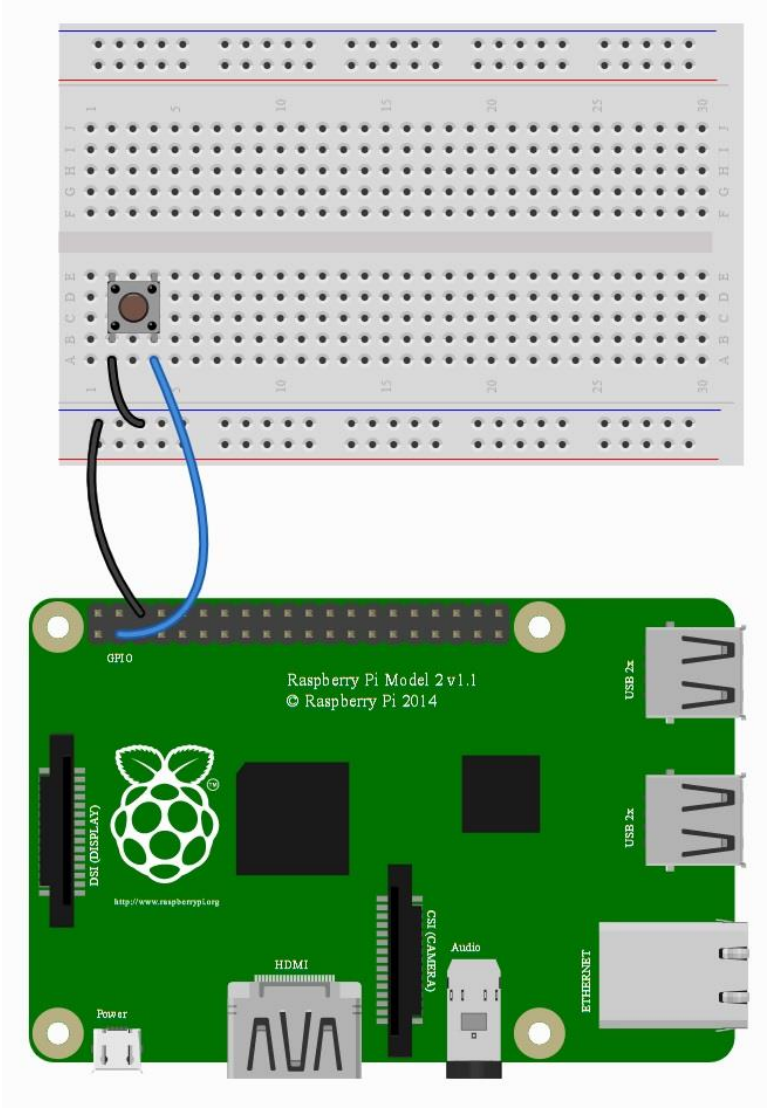
```
sleep(1)
led.value = 0.5 # fél fényerő
sleep(1)
led.value = 1 # teljes fényerő
sleep(1)
```

A folyamatos be- és kikapcsoláshoz hasonlóan a PWMLED paranccsal megoldható, hogy a fényerő folyamatosan nőjön, majd csökkenjen:

```
from gpiozero import PWMLED
from signal import pause
led = PWMLED(17)
led.pulse()
pause()
```

2.5. GOMB KEZELÉSE

A GPIO Zero könyvtár segítségével természetesen bemeneteket is tudunk kezelni, erre legegyszerűbb példa egy nyomógomb alkalmazása.



A gomb lenyomott állapotának lekérdezése:

```
from gpiozero import Button
button = Button(2)
while True:
if button.is_pressed:
print("Button is pressed")
else:
print("Button is not pressed")
```

Várakozás egy gomb megnyomására, mielőtt folytatnánk a programot:

```
from gpiozero import Button
button = Button(2)
button.wait_for_press()
print("Button was pressed")
```

A gomb minden megnyomásakor hívjon meg egy eljárást:

```
from gpiozero import Button
from signal import pause
def say_hello():
print("Hello!")
button = Button(2)
button.when_pressed = say_hello
pause()
```

Hasonlóképpen, az eljárás-hívását a gomb felengedéséhez is köthetjük:

```
from gpiozero import Button
from signal import pause
def say_hello():
print("Hello!")
def say_goodbye():
print("Goodbye!")
button = Button(2)
button.when_pressed = say_hello
button.when_released = say_goodbye
pause()
```

2.6. GOMBBAL VEZÉRELT LED

Az alábbi példaprogram bekapcsolja a LED-et a gomb lenyomásakor, és kikapcsolja a gomb felengedésekor:

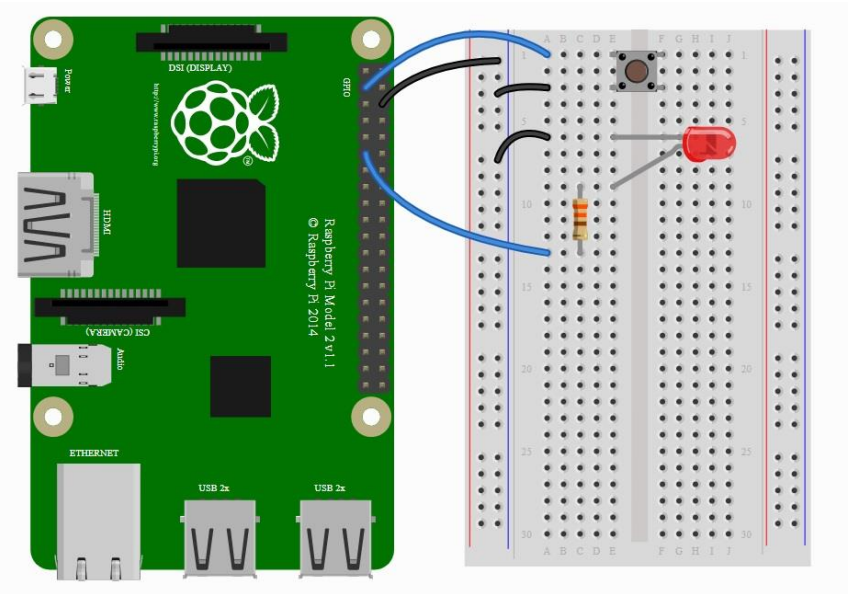
```
from gpiozero import LED, Button
from signal import pause
led = LED(17)
```

```
button = Button(2)
button.when_pressed = led.on
button.when_released = led.off
pause()
```

Alternatív lehetőség:

```
from gpiozero import LED, Button
from signal import pause
led = LED(17)
button = Button(2)
led.source = button
pause()
```

A példaprogramhoz használjuk az alábbi elrendezést:



Tesztkapcsolás BreadBoardon

Végül (de messze nem utolsónak!) egy érdekes alkalmazás, ami a könyvtár lehetőségeit kihasználva kikapcsoló gombot ad a Raspberry-nek!

2.7. KIKAPCSOLÓ GOMB

A Button osztály lehetővé teszi egy speciális parancs futtatását is, ha a kijelölt gombot egy bizonyos ideig lenyomva tartjuk. Az alábbi példa leállítja a Raspberry Pi-t, ha a gombot 2 másodpercig lenyomva tartjuk:

```
from gpiozero import Button
from subprocess import check_call
from signal import pause
def shutdown():
check_call(['sudo', 'poweroff'])
shutdown_btn = Button(17, hold_time=2)
shutdown_btn.when_held = shutdown
pause()
```

A forráskódok letölthetőek:

- 1. program
- 2. program
- 3. program
- 4. program
- 5. program
- 6. program
- 7. program
- 8. program
- 9. program
- 10. program