Tartalomjegyzék

Python futtatása

Jupyterhub

- Jelentkezzetek be a jupyter.math.bme.hu-ra a **`leibniz`-es felhasználónévvel és jelszóval**
- **Python 3**-at fogunk használni!
- Ez a notebook hasonlít ahhoz, mint amikor saját gépr?l ezt futtatod:

```
jupyter notebook
```

leibniz

- A konzol-ba ezt írjuk be:

```
python3
```

- kilépni az így lehet:

```
exit()
```

Saját gépr?l

Installáljuk az Anaconda-t, **3.7**-es verzió!

- hogyan Installáljuk az Anacondat Windows-on
- Más disztribúciót is lehet használni, úgymint:
  - python.org
  - WinPython

Ha ezt megtettük, akkor több parancs segítségével is interakcióba léphetünk a Python-nal:

- parancssor: `python` vagy `ipython`
- Spyder
- `idle`
- `jupyter notebook`

Feladatok

factorial

Írjunk egy függvényt, ami kiszámolja *n* faktoriális értékét.

Összehasonlítás

Írjunk python függvényt, ami két paraméter? és az els? paramétert összehasonlítja a második paraméterrel.

A függvény neve legyen **hasonl**, kett? paramétere legyen: *x*,*y*
Ha *x* = *y*, akkor 'Megegyeznek' szöveget printeljen,
Ha *x* > *y*, akkor 'Az els? nagyobb, mint a második' szöveget printeljen,
Ha *x* < *y*, akkor 'Az els? kisebb, mint a második' szöveget printeljen.

Próbáljuk meg az if függvényt elif és else használatával is megírni.

celsiusra

Írjunk python függvényt, ami egy Fahrenheitben megkapott hőmérsékletet átvált Celsius fokra. A függvény neve legyen **celsiusra**, és paraméterként egy **fahrenheit** nevű számot kapjon. Úgy lehet kiszámolni ezt az értéket, hogy a Fahrenheit-ben mért hőmérsékletből kivonunk 32-t, majd az így kapott számot megszorozzuk 5/9-el.

- https://hu.wikipedia.org/wiki/Fahrenheit
- példák itt

Másodfokú egyenlet megoldóképlete

El?ször töltsük be az
import math
paranccsal azt a csomagot, amivel majd gyököt tudunk vonni az
math.sqrt()
parancs segítségével. A függvény 3 paramétere legyen *a*,*b*,*c* az együtthatók és kimenete legyen a másodfokú

egyenlet valós gyökeinek listája. Ha nincs valós gyök, akkor üres listával térjen vissza a függvény.

Szorzótábla

Printeljük ki a következ? szorzótáblát:

```
1: 1 2 3 4 5 6 7 8 9
2: 2 4 6 8 10 12 14 16 18
3: 3 6 9 12 15 18 21 24 27
4: 4 8 12 16 20 24 28 32 36
5: 5 10 15 20 25 30 35 40 45
6: 6 12 18 24 30 36 42 48 54
7: 7 14 21 28 35 42 49 56 63
8: 8 16 24 32 40 48 56 64 72
9: 9 18 27 36 45 54 63 72 81
```

Prím-e

Írjunk python függvényt, ami megmondja, hogy egy pozitív egész szám prím-e.

A függvény neve legyen **prime**, egy paramétere legyen:

- **x**, a vizsgálandó szám

A függvény **`True`**-val vagy **`False`**-al térjen vissza attól függően hogy a szám prím vagy sem.

A biztonság kedvéért érdemes ellen?rizni, hogy az argumentum valóban pozitív integer-e; ha nem, akkor térjen vissza a függvény a **`None`** értékkel.

Prímek között

Írj egy 2 argumentumú függvényt primek_között() néven úgy, hogy a primek_között(m,n) térjen vissza a prímekkel az [*m*,*n*] intervallumból.

Hatvány

Írjunk egy függvényt max_exp() néven úgy, hogy *m*,*n* természetes számok esetén max_exp(m,n) térjen vissza a legnagyobb *k* természetes számmal, melyre $m^k \mid n$. Feltehet?, hogy $m > 1$.

Oszthatóság

1.

Írjunk egy 2 paraméter? függvényt osztható() néven a következ? módon: Az els? bemenete legyen egy lista, a második pedig egy természetes szám. A függvény térjen vissza a lista azon elemeinek listájával, amelyek oszthatók ezzel a természetes számmal.
Például:

```
osztható(list(range(30,50)),7)
[35, 42, 49]
```

2.

Definiáljunk egy osztók() függvényt, ami egy természetes szám valódi osztóinak listáját adja vissza.

Tökéletes számok

Írjunk programot, mely bekér egy pozitív egész számot és leellen?rzi, hogy tökéletes szám-e.

Listák

1.

Írjunk egy függvényt dupla() néven, ami bemenetnek 1 listát kap és visszatér **True**-val, ha van benne olyan elem, ami legalább kétszer szerepel, egyébként visszatér **False**-szal.

2.

Definiáljunk egy függvényt rendezett_e() néven, aminek egy paramétere egy egész számokból álló lista, és eldönti, hogy rendezett-e a lista. Ha rendezett, akkor térjen vissza **True**-val, egyébként **False**-szal. Ha listában egy elem is többször szerepel, akkor térjen vissza **None**-nal.

3.

Írjunk egy függvényt részrendezés() néven, aminek 2 paramétere egy lista és egy természetes szám. A függvény térjen vissza a lista els? *n* rendezett elemével.
Például:

```
részrendezés([6,4,5,2,1],3)
[1,2,4]
```

Prímfaktorizáció

Írjunk egy függvényt prím_faktorizáció() néven, aminek bemenete egy természetes szám és egy olyan listával tér vissza, amiben párok vannak. A pár els? tagja adja vissza, hogy melyik prím szerepel a faktorizációban, a második tagja pedig, hogy milyen hatvánnyal szerepel.
(Tipp: Felhasználható erre a célra a max_exp() függvény.)
Például:

```
prime_decomp(90)
[(2, 1), (3, 2), (5, 1)]
```

Tuple

Definiáljunk egy függvényt lookup() néven, aminek 2 argumentuma van. A második argumentuma egy lista, ami 2 hosszú tuple-ket tartalmaz, az els? argumentum pedig a kulcs. A lookup(kulcs, lista) hívás térjen vissza az els? olyan tuple második tagjával, aminek az els? tagja megegyezik a kulcs bemenettel. Ha nincs ilyen tuple a listában, akkor térjen vissza None-nal.

Legnagyobb közös osztó

Definiáljuk az lnko() függvényt, aminek paramétere két természetes szám és visszatér a legnagyobb közös osztójukkal. (Ehhez felhasználható a prím_faktorizáció() függvény.)