07_PythonBev_Fuggvenyek-Gyakorlas

November 21, 2024

1 7. alkalom: Függvények gyakorlás

1.1 Feladat: Prímtesztelés

Készítsünk függvényt, amely eldönti egy természetes számról, hogy prím-e!

[1]:
```
# 1. változat: függvény nélkül
# Ez a [2, n-1] intervallumon minden számmal ellenőrzi az oszthatóságot
n = 23
eredm = True
for k in range(2,n):
    if n%k == 0:
        eredm=False
        break        # kilép a ciklusból ha eljut ide
print(eredm)
```

```
True
```

[2]:
```
# 2. változat: függvénnyel
# javított verzió: Ez a [2, sqrt(n)] intervallumon ellenőriz csak

def is_prime(n):
    eredm = True
    n0 = int(n**0.5)+1
    for k in range(2,n0):
        if n%k == 0:
            eredm=False
            break
    return eredm
```

[3]:
```
# előző "igényesebben"
def is_prime(n):
    n0 = int(n**0.5)+1
    for k in range(2, n0):
        if n % k == 0:
            return False
    return True
```

[4]:
```
is_prime(123),is_prime(7),is_prime(1993)
```

[4]:
```
(False, True, True)
```

1.2 Szervezzük át a korábban már megírt programjainkat függvénnyé

1.2.1 Példa: n-szer n-es háromszög * karakterekből.

[ ]:
```
## korábbi kódunk
n = 10
for hossz in range(1, n + 1):
    print("*" * hossz )
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

[6]:
```
# fügvénnyel amely stringet ad vissza
def haromszog(n):
    s = ""
    for elem in range(n):
        s +="*"*(elem+1)
        s += "\n"
    return s[0:-1]
```

[7]:
```
print(haromszog(10))
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

1.2.2 Példa: Magánhangzók statisztikája szótárral (angol kisbetűs szövegben).

[ ]:
```
## korábbi kódunk

text = 'This is a short text for testing the algorithm.'
vowels = 'aeiou'

stat = dict()

for mgh in vowels:
    mgh_db = 0
    for betu in text.lower():
        if betu == mgh:
            mgh_db += 1
    print(mgh, ":", mgh_db )
    stat[mgh] = mgh_db
```

```
o : 3
e : 3
i : 4
a : 2
u : 0
```

[9]:
```
stat
```

[9]:
```
{'o': 3, 'e': 3, 'i': 4, 'a': 2, 'u': 0}
```

[10]:
```
# függvénnyel

def mgh_stat(text):
    vowels = {'a', 'e', 'i', 'o', 'u'}

    stat = dict()

    for mgh in vowels:
        mgh_db = 0
        for betu in text.lower():
            if betu == mgh:
                mgh_db += 1
        stat[mgh] = mgh_db
    return stat
```

[11]:
```
mgh_stat("Ez egy proba szoveg")
```

[11]:
```
{'o': 2, 'e': 3, 'i': 0, 'a': 1, 'u': 0}
```

1.3 Lambda kifejezések

- A lambda kifejezés nem más, mint egysoros, névtelen függvény.
- Lambda kifejezések a dokumentációban

[ ]:
```
## Példa lambda kifejezésre.
f = lambda x: x + 42
f(3)
```

[ ]:
```
45
```

[13]:
```
# Egynél több bemenet is megengedett.
```

1.3.1 Lambda kifejezés alkalmazása rendezésnél

[14]:
```
# Párok listájának rendezése a második elem szerint.
pairs = [('alma', 22), ('körte', 11), ('barack', 33)]
sorted(pairs, key = lambda x: x[-1] )
```

[14]:
```
[('körte', 11), ('alma', 22), ('barack', 33)]
```

[15]:
```
sorted(pairs)
```

[15]:
```
[('alma', 22), ('barack', 33), ('körte', 11)]
```

[8]:
```
# Az előző feladat megoldása lambda kifejezés nélkül külön függvényt definiálva
pairs = [('alma', 22), ('körte', 11), ('barack', 33)]

def rend(x):
    return x[-1]

sorted(pairs, key=rend )
```

[8]:
```
[('körte', 11), ('alma', 22), ('barack', 33)]
```

[10]:
```
# Szótárkulcsok rendezése az értékek szerint.
words = {'king': 203, 'denmark': 24, 'queen': 192}

def rend_kulcs(kulcs):
    return words[kulcs]

sorted(words, key=rend_kulcs )
```

[10]:
```
['denmark', 'queen', 'king']
```

1.4 Feladat:

Készítsünk függvényt, amely kiszámítja egy ferdén elhajított labda helyét tetszőleges időpontban.

A labdát indítsuk egy 25 méter magas épület tetejéről, 20 m/s kezdősebességgel, amely a vízszintessel 35°-os szöget zár be.

Hol lesz a labda 1.5 másodperc múlva?

A függvényeket felhasználva gyűjtse ki a test hely adatait egy-egy listába. A kezdeti 0 időponttól, t_max másodpercig, adott dt lépésekkel. Az idő adatokat is tárolja el egy listában.

Mikor érkezik meg a labda a talaj szintre? Milyen messze van ekkor vízszintes irányban a kiindulási helytől?

Milyen magasra jutott a labda?

[ ]:
```
import math
```

[ ]:
```
# megadott adatok
v0 = 20    # m/s  kezdősebesség nagysága
alfa_deg = 35   # szög fokban
alfa  = math.radians(alfa_deg)   # szög radiánban

h = 25     # m kezdeti mgasság

g = 9.81   # m/s^2

t1 = 1.5
```

[ ]:
```
# kezdeti hely és sebesség
x0 = 0.0
y0 = 0.25
vx0 = v0*math.cos(alfa)
vy0 = v0*math.sin(alfa)
print(vx0, vy0)
```

```
16.383040885779835 11.471528727020921
```

[ ]:
```
def x_irany(t):
    return x0 + vx0 * t

def y_irany(t):
    return y0 + vy0 * t - g/2 *t**2
```

[ ]:
```
# hol vagyunk t1 = 1.5 s-nál?
print("x koordináta: ", x_irany(t1) )
print("y koordináta: ", y_irany(t1) )
```

```
x koordináta:  24.574561328669752
y koordináta:  6.421043090531379
```

1.5 Teknős grafika függvényekkel

[18]:
```
#import turtle
```

[19]:
```
!pip3 install ColabTurtle
```

```
Requirement already satisfied: ColabTurtle in
/home/domotor/anaconda3/envs/pylatest/lib/python3.9/site-packages (2.1.0)
```

[20]:
```
import ColabTurtle.Turtle as turtle
```

[ ]:
```
## lássuk, hogyan működik a teknőc
turtle.initializeTurtle(initial_speed = 3 )
turtle.bgcolor(1,1,1)
turtle.color("red")    # red = piros színű a toll
turtle.forward(200)    # forward = előre megyünk 200 pixelt
turtle.left(90)        # left = balra fordulunk 90 fokot
turtle.forward(200)    # forward = előre megyünk 200 pixelt

turtle.penup()         # most felvesszük a tollat
turtle.forward(100)    # forward = előre megyünk 100 pixelt
turtle.pendown()       # és letesszük a tollat
```

```
<IPython.core.display.HTML object>
```

[22]:
```
# írjunk függvényt ami megvalósít egy adott méretű továbbosonoást
def oson(tavolsag):
    turtle.penup()
    turtle.forward(tavolsag)
    turtle.pendown()
```

[23]:
```
# írjunk függvényt amely négyzetet rajzol adott színnel és mérettel
def negyzet(meret : int, szin: str) -> None:
    '''
    adott meretü és színű négyzetet rajzol teknősgrafikával.
    '''
    turtle.color(szin)
    for i in range(4):
        turtle.forward(meret)
        turtle.left(90)
```

[24]:
```
# játszunk el a függvényeinkkel!
#Rajzoljunk több elfofgatott négyzetet.
#Rajroljunk egymás mellé más-más színű négyzeteket!

turtle.initializeTurtle(initial_speed=13)

for i in range(10):
```

```
    meret = (i+1)*20
    turtle.left(i*1)
    negyzet(meret,"red")
```

```
<IPython.core.display.HTML object>
```

[25]:
```
# írjunk függvényt ami testőleges szabályos sokszöget rajzol!
def Nszog(N, meret, szin):
    '''
    N oldalú szabályos sokszöget rajzol meret oldalhosszal
    '''
    turtle.color(szin)
    for i in range(N):
        turtle.forward(meret)
        turtle.left(360/N)
```

[26]:
```
turtle.initializeTurtle(initial_speed=13)

for i,szin in enumerate(["red","green","violet","orange","blue"]):
    meret = (i+1)*20
    turtle.left(i*5)
    oson(-i*5)
    Nszog(6,meret,szin)
```

```
<IPython.core.display.HTML object>
```

[27]:
```
# rajzoljunk kisházat a meglévő sokszögek segítségével
```

[28]:
```
def haz(meret):
    turtle.right(90)
    # házikó fő része: egy négyzet
    negyzet(meret, "red")
    # felmegyünk a tető rajzolásához:
    turtle.left(90)
    oson(meret)
    turtle.right(90)
    # most jön a tető
    Nszog(3, meret, "blue")
    # elmegyünk az ablak helyére:
    oson(meret*0.2)
    turtle.right(90)
    oson(meret*0.5)
    turtle.left(90)
    # most jön az ablak
    negyzet(meret*0.3, "green")
    # ... és vissza a kiindulópontra. Ha nem mennénk, összezavarodhatnánk!
    turtle.right(90)
```

```
    oson(meret*0.5)
    turtle.right(90)
    oson(meret*0.2)
    turtle.right(180)
```

[29]:
```
turtle.initializeTurtle(initial_speed=13)

haz(100)
```

```
<IPython.core.display.HTML object>
```

2 Feladatok

2.1 1.feladat Legnagyobb közös osztó

Készítsünk függvényt két természetes szám legnagyobb közös osztójának a meghatározására!

- Gondoljuk végigig, hogyan oldanánk ezt meg egyszerűen az oszthatóságot vizsgálva.
- Majd ismerkedjünk meg egy hatékonyabb megoldással, az Eulideszi algoritmussal: https://hu.wikipedia.org/wiki/Legnagyobb_k%C3%B6z%C3%B6s_oszt%C3%B3
- Nézzünk utána a beépített lehetőségeknek is. Például a `math` modulban

2.2 2.feladat Legkisebbb közös többszörös

Készítsünk függvényt két természetes szám legkisebb közös többszörösének a meghatározására!

2.3 3.Feladat: Háromszög oldalak bekérése

Írjon egy programot, ami bekéri egy háromszög 3 oldalhosszát. (a,b,c) A program csak a valóban szerkeszthető háromszög oldalakat fogadja el.

Az ellenőrzést függvények segítségével végezze el!

Azaz ellenőrizze, hogy

- lehetnek-e az adatok oldalhosszak? Azaz pozitívak-e a megadott számok?
- továbbá teljesül-e a háromszög egyenlőtlenség? (Bármely két oldal összege nagyobb a harmadiknál.)

Ha valamelyik feltétel sérül arról adjon visszajelzést!

[30]:
```
def is_positive(a,b,c):
    "True: ha mind a 3 oldal pozitív szám, különben False"
    pass

def is_triangle(a,b,c):
    "True: ha teljesül a 3szögegyenlőtlenség, különben False"
    pass
```

[ ]:
```

```