l-system

December 4, 2023

1 Az L-rendszer

Az L-system (Lindenmayer rendszer) egy matematikai formális nyelv. Sorozatokat hoz létre azáltal, hogy ismételten alkalmaz bizonyos szabályokat egy kiindulási állapoton. Az "L" az L-rendszerben Aristid Lindenmayer magyar kutató nevéből származik. Lindenmayer élesztőgombákkal foglalkozott. Az L-rendszer használható biológiai struktúrák modellezésére is. Formálisan következőképpen adható meg:

G = (T, s, H), ahol T egy véges ábécé, s az axióma, H a szabályok halmaza.

Az L-rendszerek rendkívül hatékonyak lehetnek a növények struktúrájának modellezésére. A növények ágazódása, leveleinek elrendeződése és számos más aspektusa leírható szabályokkal és iterációkkal. Például, egy egyszerű L-system segítségével lehet egy faágakat és leveleket modellezni. Játékfejlesztésben is gyakran alkalmazzák, például procedurális térképgenerálás, terep, növényzet létrehozásánál.

1.0.1 A három fő komponens:

- Axióma (kiinduló állapot): Ez a kezdő állapot vagy kezdőszimbólumok halmaza, amelyekből kiindulunk.
- Szabályok: Ezek a szabályok írják le, hogy hogyan változtassuk meg az aktuális szimbólumokat vagy szekvenciákat az iterációk során. Például egy adott szimbólum cserélhető egy másik szekvenciára vagy szimbólumra.
- Iterációk: Az iterációk (ismétlések) során alkalmazzuk a szabályokat az aktuális sorozatra, létrehozva egyre bonyolultabb és részletesebb struktúrákat.

1.0.2 L-rendszer megvalósítása:

1. szabályok értelmezése
2. nyelvtant alkalmazó függvény elkészítése - kimenete az adott szó
3. függvény elkészítése, ami képes az adott nyelvtan szavait ábrázolni

1.1 1. Példa - egyszerű szabályrendszer

Például, ha egy L-rendszer axiómája "A" és az egyik szabály az, hogy minden "A" cserélődik "AB" sorozatra, akkor az iterációk során a sorozat így fejlődik:

0. Iteráció: A (kezdeti állapot, maga az axióma)
1. Iteráció: AB (az "A" helyettesítve "AB"-vel)
2. Iteráció: ABB
3. Iteráció: ABBB

Pszeudokód:

A program bekéri a végrehajtandó iterációk számát, utána végigmegy az aktuális szó minden karakterén és a “new” nevű változóhoz szabályok alapján hozzáadja a szükséges karaktereket. Minden iteráció végén kiírja a szót, “axiom” nevű változót aktualizálja és a “new” változót kiüríti.

[37]:
```
#({A,B}, A, {A->AB, B->B})

axiom = 'A'
new = ''

n = int(input("Adja meg az iterációk számát: "))

for i in range (0,n):
    print(str(i)+". iteráció: " + axiom)
    for letter in axiom:
        if (letter=='A'):
            new += 'AB'
        elif (letter=='B'):  #lehetne else is
            new += 'B'
    axiom = new
    new = ''
```

```
0. iteráció: A
1. iteráció: AB
2. iteráció: ABB
3. iteráció: ABBB
4. iteráció: ABBBB
```

##

1. Feladat:

Az előző kód alapján készítse el a következő rendszer megvalósítását:

Kezdeti állpot: X

Szabályok: (X → F+[[X]-X]-F[-FX]+X), (F → FF)

Ahol: - “F” jelentése “előre rajzol” - “−” jelentése“jobbra fordul 25°-kal” - “+” jelentése“balra fordul 25°-kal” - “X” jelentésenem felel meg semmiféle rajzolási műveletnek, és a görbe alakulásának irányítására szolgál - “[” jelentése a jelenlegi pozíció és szög értékek mentése - ”]” jelentése az utolsó mentett pozíció és szög visszaállítása és a mentett adatsor törlése

[38]:
```
#(X → F+[[X]-X]-F[-FX]+X), (F → FF)

axiom = 'X'
new = ''

n = int(input("Adja meg az iterációk számát: "))
```

```
for i in range (0,n):
    print(str(i)+". iteráció: "+ axiom)
    for letter in axiom:
        if (letter=='X'):
            new += 'F+[[X]-X]-F[-FX]+X'
        elif (letter=='F'):  #lehetne else is
            new += 'FF'
    axiom = new
    new = ''
```

```
0. iteráció: X
1. iteráció: F+[[X]-X]-F[-FX]+X
2. iteráció:
FFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+X
3. iteráció: FFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[
[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X
]-X]-F[-FX]+XFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+
XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF
+[[X]-X]-F[-FX]+X
4. iteráció: FFFFFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-
FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX
]+XF+[[X]-X]-F[-FX]+XFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]
-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F
[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]
-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F
[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X
]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFF+[[X]-
X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFFFFFFFFFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F
[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-
FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-
X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]
-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFF+[[X]-X]-F[-FX]+XF+[[X]-
X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X]-X]
-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFFFFFFFFFF+[[X]-X]-F[-FX]+XF+[
[X]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+XFFF+[[X]-X]-F[-FX]+XF+[[X
]-X]-F[-FX]+XFFFFF+[[X]-X]-F[-FX]+XF+[[X]-X]-F[-FX]+X
```

1.2 2. Példa - Koch-görbe leírása

- változók: F
- constansok: +,-
- kezdeti állapot: F
- szabályok: (F->F+F-F-F+F)

Ahol: - “F” jelentése “előre” - “+” jelentése “fordulj balra 90°-ot” - “-” jelentése “fordulj jobbra 90°-ot”

1.2.1 2.1. Alap probléma megoldása, nyelv generálása:

Figyeljük meg, hogy a szabály alapján minden egyes 'F' betű helyét 5db 'F' betű kerül. Ez azt jelenti, hogy a generált nyelv hossza 5n, ahol n az iterációk számát jelenti (pl.: 3. iteráció 125db 'F' karaktert eredményez). Könnyen belátható, hogy ez exponenciális növekedést eredményez. A kiindulási állapot megjelenítése miatt szükséges a ciklus első iterációját külön kezelni.

[22]:
```python
axiom = 'F'
new = ''

n = int(input("Adja meg az iterációk számát: "))

for i in range (0,n):
    if (i == 0):
        new = axiom
    else:
        for letter in axiom:
            if (letter == 'F'):
                new += 'F+F-F-F+F'
            else:
                new += letter
        axiom = new
        new = ''
    print(str(i)+". iteráció: "+ axiom)
```

```
0. iteráció: F
1. iteráció: FF+F-F-F+F
2. iteráció: F+F-F-F+FF+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F
3. iteráció: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+FF+F-F-F+F+F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F
4. iteráció: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F
+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F
+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F
+F-F+F-F-F+F+F+F-F-F+FF+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+
```

F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F

1.2.2 2.2. Nyelv generálása - függvény formájában

A Python nyelv szintaxisa alapján az előző rövid kódot függvénnyé alakítottuk. Ez segíti a kód későbbi felhasználhatóságát.

[23]:
```
axiom = 'F'
new = ''

def generateSentence(n):
    global axiom
    for i in range (0,n):
        if (i == 0):
            new = axiom
        else:
            for letter in axiom:
                if (letter == 'F'):
                    new += 'F+F-F-F+F'
                else:
                    new += letter
            axiom = new
            new = ''
        print(str(i)+". iteráció: "+ axiom)
    return axiom

n = int(input("Adja meg az iterációk számát: "))
sentence = generateSentence(n)
```

```
0. iteráció: F
1. iteráció: FF+F-F-F+F
2. iteráció: F+F-F-F+FF+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F
3. iteráció: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+FF+F-F-F+F+F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+
4. iteráció: F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F
+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F
+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F
+F-F+F-F-F+F+F+F-F-F+FF+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+
F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+
F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F+F+F-F-F+F+F-F+F-F-F+F-F+F-F-F+
```

```
F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F+F-F-F+F+F-F-F+F+F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+
F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F
```

1.3 3. Nyelvtan szavainak ábrázolása - Turtle Graphics

Turtle egy könnyen kezelhető könyvtár grafikai munkák elvégzésére

A turtleStart() függvény tartalmazza a grafikára vonatkozó alap beállításokat: középre helyezi a kurzort, engedélyezi a rajzolást és meghatározza a használt vászon(kép) méretét.

- penup() - megszünteti a rajzolást (következő pendown() eseményig nem rajzol vonalat)
- pendown() - előző utasítás párja, ez után következő parancsok rajzolnak
- speed() - meghatározza a rajzolás sebességét, értéke megadható “slowest” - “fastest” között, vagy egész számmal 1-10-ig.
- hideturtle() - elrejti kurzort
- screensize() - megjelnítéshez használt vászon méretét és színét lehet állítani, canvwidth, canvheight, bg paraméterek megadásával
- exitonclick() - bezárja az elkészített ábrát
- tracer() - alapértelmezett esetben (1) mutatja az animációt, 0 érték esetén a kész ábrát mutatja

[43]:
```
import turtle

def turtleStart():
    turtle.screensize(canvwidth=1000, canvheight=1000, bg = "white")
    turtle.hideturtle()
    turtle.setposition(0,0)
    turtle.pendown()
    turtle.speed("fastest")
    turtle.tracer()
```

Alapvető vezérlési utasítások

- forward() - előre megy
- left() - balra fordul
- right() - jobbra fordul
- goto() - adott koordinátára rakja a kurzort

[44]:
```
import importlib
importlib.reload(turtle)

def drawSentence(s):
    for i in s:
```

```
        if (i == 'F'):
            turtle.forward(10)
        elif (i == '+'):
            turtle.left(90)
        elif (i == '-'):
            turtle.right(90)

turtleStart()
drawSentence(sentence)
turtle.exitonclick()
```

1.4 Példa

ide kéne szöveg

- változók: X, F
- konstansok: + , − , [,]
- kezdeti állapot: X
- szabályok: (X → F+[[X]-X]-F[-FX]+X), (F → FF)
- szög: 25°

[8]:
```
import turtle
import importlib
importlib.reload(turtle)

def turtleStart():
    turtle.screensize(canvwidth=1000, canvheight=800, bg = "#C7E2E7")
    turtle.hideturtle()
    turtle.penup()
    turtle.setposition(0,-250)
    turtle.left(90)
    turtle.pendown()
    turtle.speed("fastest")
    turtle.tracer(1)

def general_noveny(axiom, iteraciok):
    aktualis_kifejezes = axiom
    uj_kifejezes = ''
    for _ in range (0,iteraciok):
        for betu in aktualis_kifejezes:
            if betu == 'X':
                uj_kifejezes += 'F+[[X]-X]-F[-FX]+X'
            elif betu == 'F':
                uj_kifejezes += 'FF'
            else:
                uj_kifejezes += betu
        aktualis_kifejezes = uj_kifejezes
        uj_kifejezes = ''
```

```
    return aktualis_kifejezes

def rajzol_noveny(kifejezes, szog, lepes_hossz):
    verem = []
    turtle.speed(0)
    for szimbolum in kifejezes:
        if szimbolum == 'F':
            turtle.forward(lepes_hossz)
        elif szimbolum == '+':
            turtle.left(szog)
        elif szimbolum == '-':
            turtle.right(szog)
        elif szimbolum == '[':
            verem.append((turtle.pos(), turtle.heading()))
        elif szimbolum == ']':
            pozicio, szog_irany = verem.pop()
            turtle.penup()
            turtle.goto(pozicio)
            turtle.pendown()
            turtle.setheading(szog_irany)

turtleStart()
final_sentence = general_noveny('X', 5)
rajzol_noveny(final_sentence, 25, 3)

turtle.exitonclick()
```

1.5 Ágak ábrázolása és a verem adatszerkezet

A fa megrajzolása áganként történik olyan módon, hogy először a program megrajzol egy teljes ágat, utána visszatér a "törzshöz" és új ág rajzolásába kezd. Ebből adódik a probléma, hogy hogyan fog visszatérni a megfelelő helyre program? Az ágak kezdetét mindig elágazás (jelölést tekintve általában +,- szimbólumok) jelzi.

A verem egy LIFO (Last In First Out) adatszerkezet. Két jellegzetes hozzá kapcsolódó parancs a Push (Python esetén append) és a Pop:

Push - verembe helyezi a kijelölt adatot

Pop - az utoljára verembe helyezett adatot eltávolítja

Pythonnal egyszerű listával lehetséges a verem adatszerkezet megvalósítása:

[3]:
```
verem = []

verem.append(1)
verem.append(2)
verem.append(3)
```

```
print(verem)

verem.pop()
print(verem)
```

```
[1, 2, 3]
[1, 2]
```

A mi esetünkben az ág kezdeténél lévő pozíciót és szöget fogja eltárolni.

- turtle.pos() - teknős aktuális X és Y koordinátája
- turtle.heading() - szög, azaz merre néz a teknős
- append (hozzáad): Ha kigenerált nyelvben a soron következő karakter a '[' , akkor a jelenlegi állapotot (pozíciót és szöget) hozzáadjuk a veremhez. verem.append((turtle.pos(), turtle.heading()))
- pop (levesz): Ha kigenerált nyelvben a soron következő karakter a ']' , akkor kivesszük a veremből a legfelső elemét. pozicio, szog_irany = verem.pop() Ezek után visszatérünk a veremből kiszedett pozícióhoz és folytatjuk a rajzolást a tárolt szöggel.

turtle.goto(pozicio) - adott helyre mozgatja a teknőst

turtle.setheading(szog_irany) - beállítja a szöget