Bevezetés a programozásba

8. Gyakorlat

Ismétlés

- Alprogramok
  - Eljárások
  - Függvények
- Változók
  - Lokális
  - Globális
- Paraméterátadás
- Opcionális paraméterek
- `main()` függvény

Main függvény

- Program belépési pontja:

```
def main():
    pass

if __name__ == "__main__":
    main()
```

Rekurzív függvények

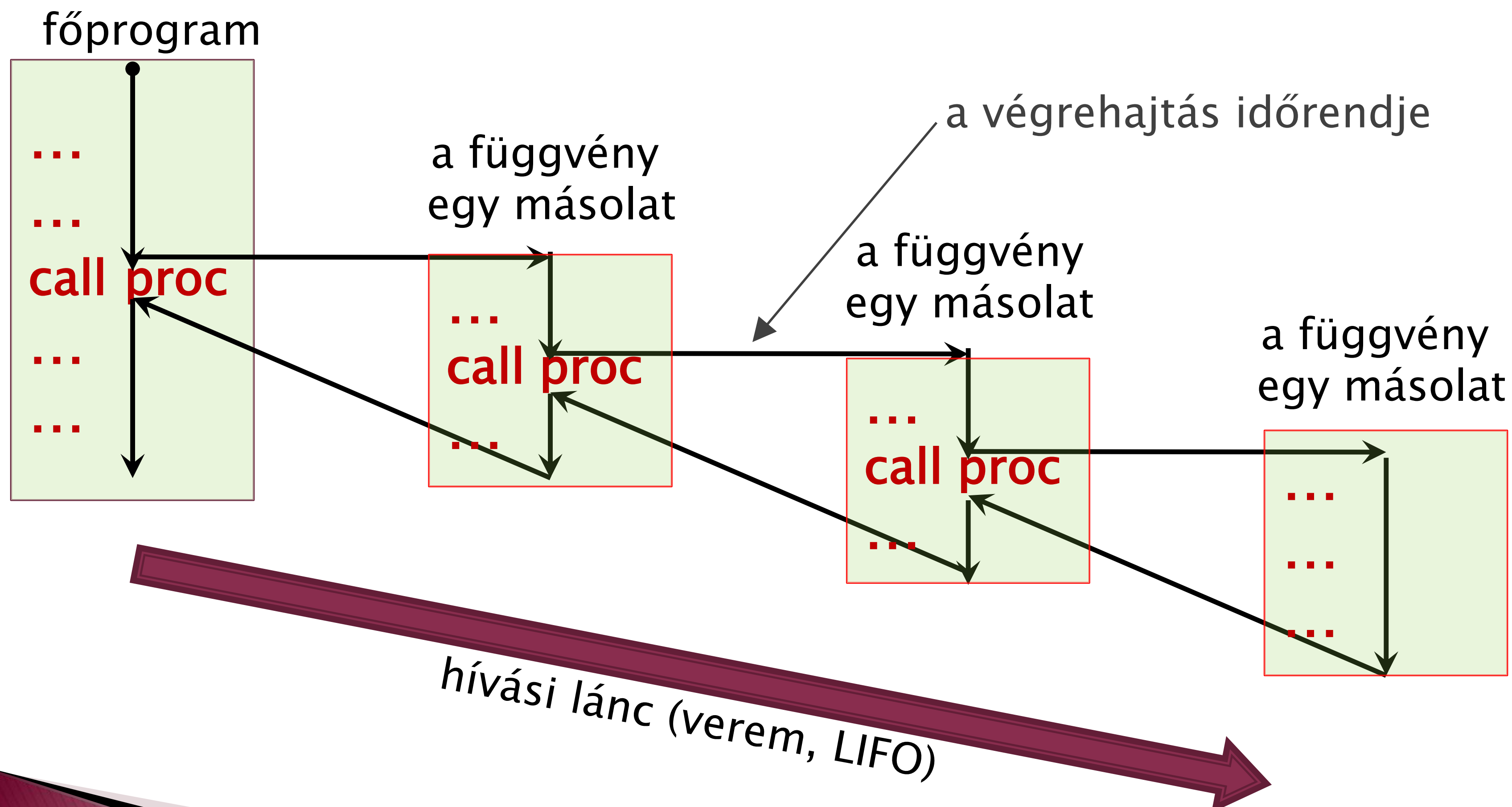
- A függvény **meghívja önmagát**.
- A függvény törzsében van legalább egy önmagára való hivatkozás.
- Ciklus helyett **elágazás**:
  - Igaz ág – leállási feltétel
  - Hamis ág – rekurzív hívás
- Rekurzív algoritmusokat általában akkor használunk, ha az alapfeladat túl bonyolult, azonban rekurzív hívásokkal ezt vissza tudjuk vezetni egyszerűbb (rész)feladatok megoldására.
- Ennek megfelelően egy rekurzív feladatot általában ehhez hasonlóan definiálunk:
  - triviális megoldás
  - általános eset egyszerűsítése

Rekurzív függvények

- A rekurzió előnyei:
  - A rekurzív függvények segítségével a kód tiszta és elegáns lesz.
  - A komplex feladat rekurzióval egyszerűbb részproblémákra bontható.
  - Szekvencia generálása könnyebb a rekurzióval, mint néhány beágyazott iteráció használata.
- A rekurzió hátrányai:
  - Néha a rekurzió mögött rejlő logikát nehéz követni.
  - A rekurzív hívások nem hatékonyak, mivel sok memóriát és időt igényelnek.
  - A rekurzív függvényekben nehéz hibát keresni.

Rekurzió

- Egy alprogram saját magát hívja meg



Feladatgyűjtemény

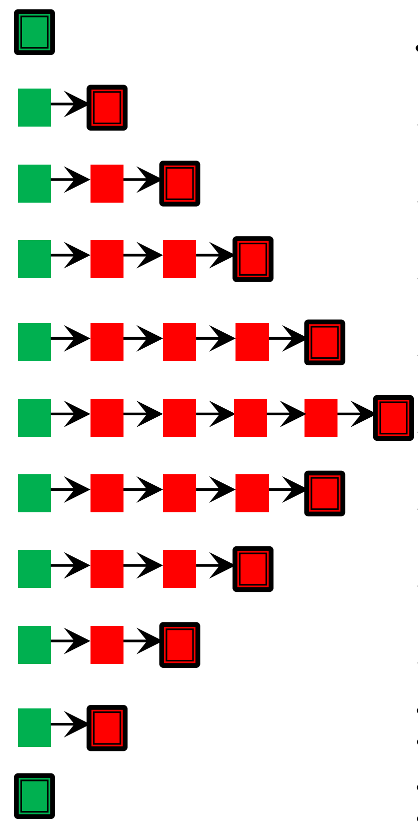
- https://viskillz.inf.unideb.hu/prog/#/
- Rekurzív függvények
  - https://viskillz.inf.unideb.hu/prog/#/?week=P1033

Feladat

- Írjatok rekurzív függvényt, mely meghatározza N faktoriálisát.

Rekurzió példa

- Faktoriális:
  - Ha a szám (N) az 1, akkor a faktoriálisa is 1.
  - Különben N! = N * (N–1)!



5!
5*4!
5*4*3!
5*4*3*2!
5*4*3*2*1!
5*4*3*2*1
5*4*3*2
5*4*6
5*24
120
120

Megoldás1

```
def fact(n: int) -> int:
    if n == 1:
        return 1
    else:
        return n * fact(n - 1)


def main():
    print(fact(5))

if __name__ == "__main__":
    main()
```

Megoldás2

```python
def fact(n: int) -> int:
    return 1 if n == 1 else n * fact(n - 1)

def main():
    print(fact(5))

if __name__ == "__main__":
    main()
```

```python
import sys

def fact(n: int) -> int:
    if n == 1:
        return 1
    else:
        return n * fact(n - 1)

def fact1(n: int) -> int:
    return 1 if n == 1 else n *
           fact1(n - 1)

def read() -> None:
# állományvégjelig olvas
    while True:
        try:
            n = int(input())
            print(fact(n))
        except EOFError:
            break
```

```python
def read1() -> None:
# állományvégjelig olvas
    for line in sys.stdin:
        n = int(line)
        print(fact(n))

def read2() -> None:
# N teszteset beolvasása
    n=int(input())
    for i in range(n):
        num=int(input())
        print(fact(num))

def read3() -> None:
# [0,50] közötti számok beolvasása
    while True:
        n=int(input())
        if n<0 or n>50:
            break
        print(fact(n))

def main():
    read3()

if __name__ == '__main__':
    main()
```

Feladat

- Írjatok rekurzív hatvány **függvényt**, amely meghatározza X^n–t.

Megoldás

```
def power(x: int, n: int) -> int:
    if n == 1:
        return x
    else:
        return x * power(x, n - 1)

def main():
    x = int(input("x = "))
    n = int(input("n = "))
    print(power(x, n))

if __name__ == "__main__":
    main()
```

Megoldás

```python
def power(x: int, n: int) -> int:
    return x if n == 1 else x * power(x, n - 1)

def main():
    x = int(input("x = "))
    n = int(input("n = "))
    print(power(x, n))

if __name__ == "__main__":
    main()
```

```python
import math
def power_it(x: int, n: int) -> int:
    p = 1
    for i in range(n):
        p = p * x
    return p

def power_rek1(x: int, n: int) -> int:
    if n == 1:
        return x
    else:
        return x * power_rek1(x,n-1)

def power_rek2(x: int, n: int) -> int:
    return x if n == 1 else x * power_rek2(x,n-1)

def main():
    x=int(input("x="))
    n=int(input("n="))
    print(power_it(x,n))
    print(power_rek1(x, n))
    print(power_rek2(x, n))
    print(int(math.pow(x,n)))
if __name__ == '__main__':
    main()
```

Rekurzió

- Írjatok rekurzív függvényt, mely meghatározza 1-től N-ig a számok összegét.

$$\sum_{i=1}^{n} i$$

Megoldás1

```
def sum(n: int) -> int:
    if n == 1:
        return 1
    else:
        return n + sum(n - 1)

def main():
      print(sum(n))

if __name__ == "__main__":
    main()
```

Megoldás1

```python
def sum1(n: int) -> int:
    return 1 if n == 1 else n + sum1(n - 1)

def main():
    print(sum1(n))

if __name__ == "__main__":
    main()
```

Rekurzió

- Írjatok rekurzív függvényt, mely meghatározza 1–től N–ig a számok négyzet összegét.

$$\sum_{i=1}^{n} i^2$$

Megoldás

```
def sum2(n: int) -> int:
    if n == 1:
        return 1
    else:
        return n * n + sum2(n - 1)

def sum2(n: int) -> int:
    return 1 if n == 1 else n * n + sum2(n - 1)
```

Feladat

- Írjatok rekurzív függvényt, mely meghatározza a következő összeget:

$$\sum_{i=1}^{n}(i^2 - 2)$$

Megoldás

```
def sum3(n: int) -> int:
    if n == 1:
        return -1
    else:
        return n * n - 2 + sum3(n - 1)

def sum3(n: int) -> int:
    return -1 if n == 1 else n * n - 2 + sum3(n - 1)
```

Feladat

- Írjatok rekurzív függvényt, mely meghatározza a következő összeget:

$$\sum_{i=3}^{n} (i^2 - 2)$$

Megoldás

```
def sum4(n: int) -> int:
    if n == 3:
        return 7
    else:
        return n * n - 2 + sum4(n - 1)
```

```
def sum4(n: int) -> int:
    return 7 if n == 3 else n * n - 2 + sum4(n-1)
```

Feladat

- Írjatok rekurzív függvényt, mely meghatározza a következő összeget:

$$\sum_{i=4}^{n} i(i^2 - 1)$$

Megoldás

```
def sum5(n: int) -> int:
    if n == 4:
        return 60
    else:
        return n * (n * n - 1) + sum5(n - 1)


def sum5(n: int) -> int:
   return 60 if n == 4 else n * (n * n - 1) + sum5(n-1)
```

Feladat

- Írjatok rekurzív függvényt, mely meghatározza a következő összeget:

$$\sum_{i=2}^{n}(i^3-5)$$

Megoldás

```python
def sum6(n: int) -> int:
    if n == 2:
        return 3
    else:
        return n * n * n - 5 + sum6(n - 1)


def sum6(n: int) -> int:
  return 3 if n == 2 else n * n * n - 5 + sum6(n-1)
```

Megoldás

```
def szum6(n: int) -> int:
    szum = 0
    for i in range(2, n + 1):
        szum += i * i * i - 5
    return szum
```

Feladat

- Írjatok rekurzív függvényt az n–ik Fibbonacci szám meghatározására!

$$F_n = \begin{cases} 0, & \text{ha } n = 0; \\ 1, & \text{ha } n = 1; \\ F_{n-1} + F_{n-2}, & \text{ha } n \geq 2. \end{cases}$$

Megoldás

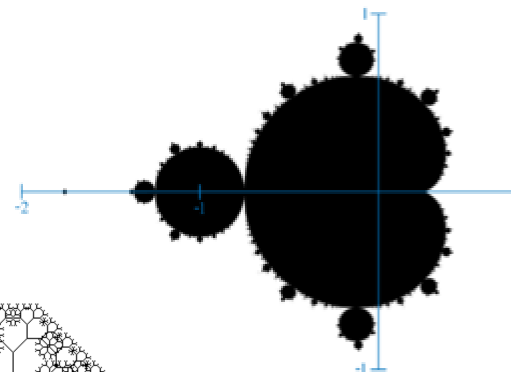
```python
def fibo(n: int) -> int:
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibo(n - 1) + fibo(n - 2)
```

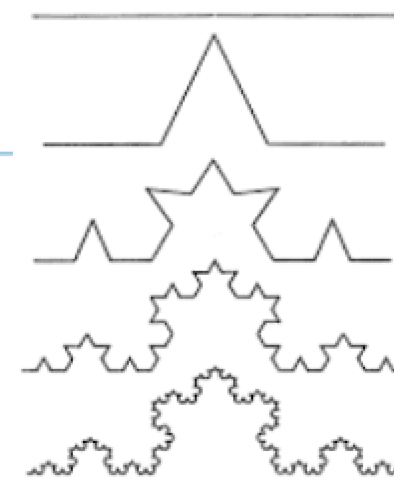
Megoldás

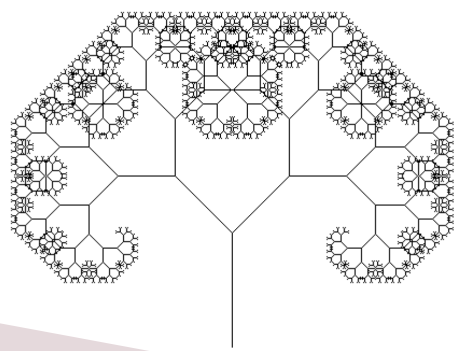
```
def fibonacci(n):
    f1 = 0
    f2 = 1
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        for i in range(1, n):
            f3 = f2 + f1
            f1 = f2
            f2 = f3
        return f3
```

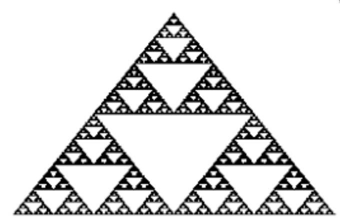
Rekurzió alkalmazási területe

- Fraktálok
- A fraktálok „önhasonló”, végtelenül komplex matematikai alakzatok, melyek változatos formáiban legalább egy felismerhető (tehát matematikai eszközökkel leírható) ismétlődés tapasztalható.
- Az elnevezést 1975-ben Benoît Mandelbrot adta, a latin fractus (vagyis törött; törés) szó alapján, ami az ilyen alakzatok tört számú dimenziójára utal, bár nem minden fraktál tört dimenziós (ilyenek például a síkkitöltő görbék).
  - Koch görbe
  - Sierpinski-háromszög
  - Mandelbrot-halmaz
  - Püthagorasz-fa
  - Newton-fraktál
  - Júlia halmaz

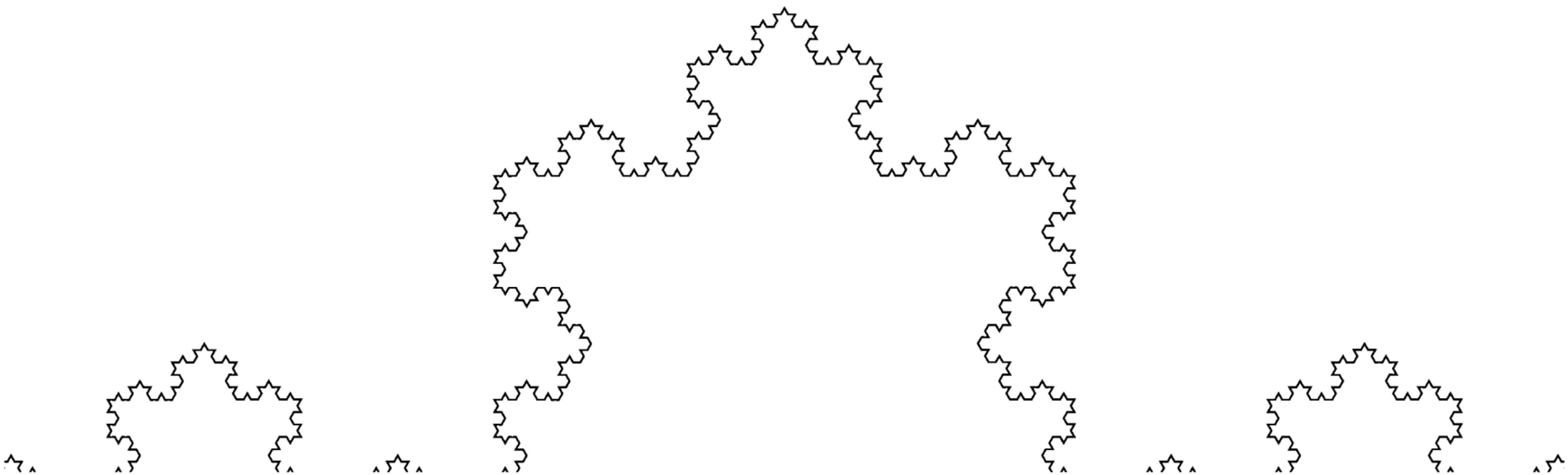






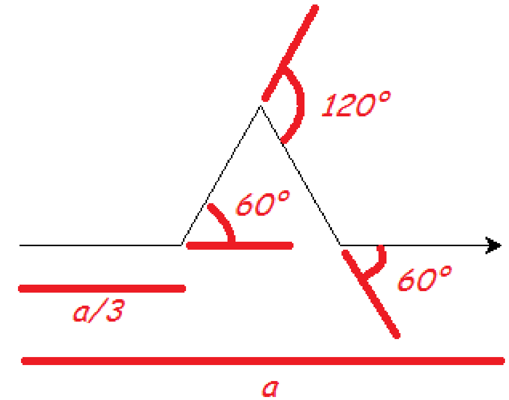


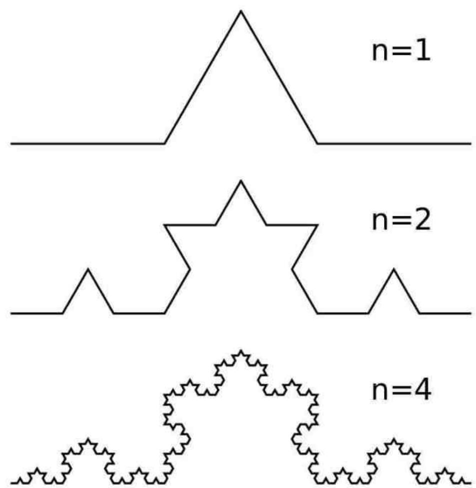
Koch görbe

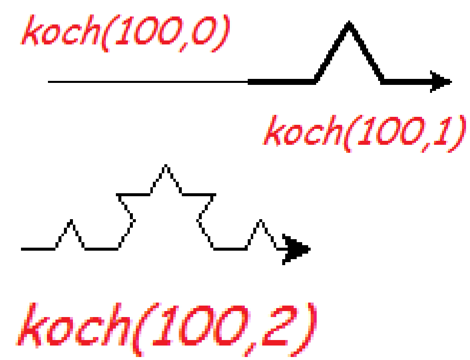


Koch görbe

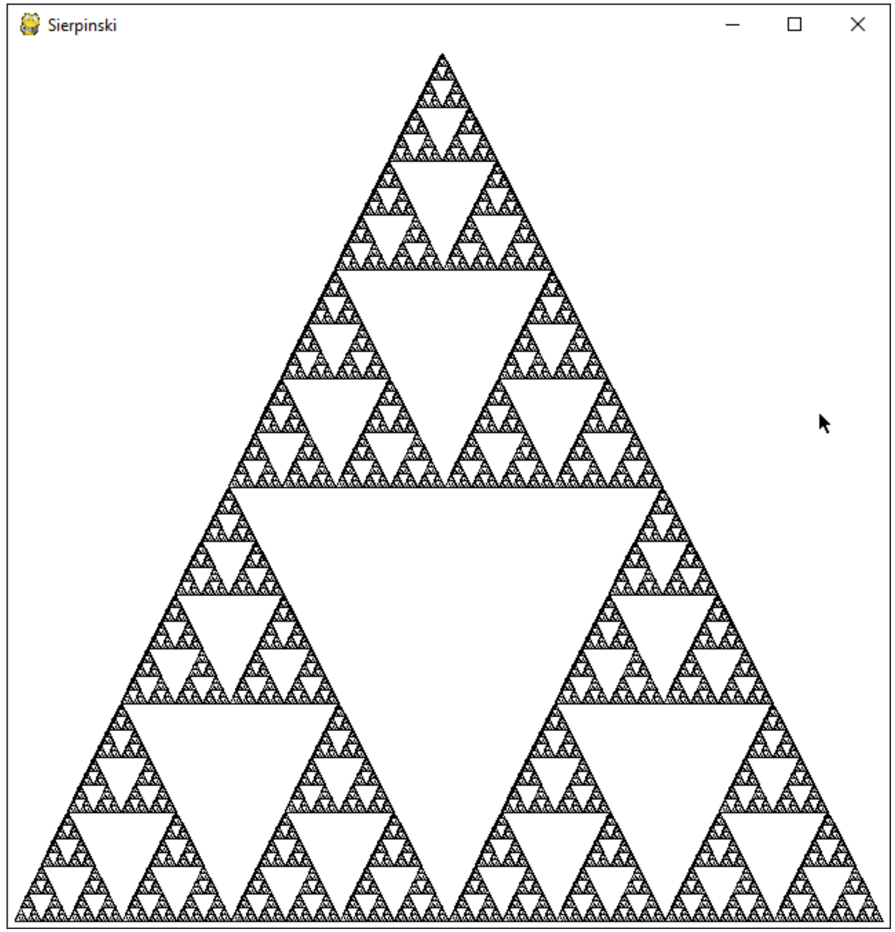
```python
from turtle import *

def koch(a, order):
    if order > 0:
        for t in [60, -120, 60, 0]:
            forward(a/3)
            left(t)
    else:
        forward(a)

def koch2(a, order):
    if order > 0:
        for t in [60, -120, 60, 0]:
            koch(a/3,order-1)
            left(t)

koch(100, 0)
pensize(3)
#koch(100,1)
koch2(100, 2)
```



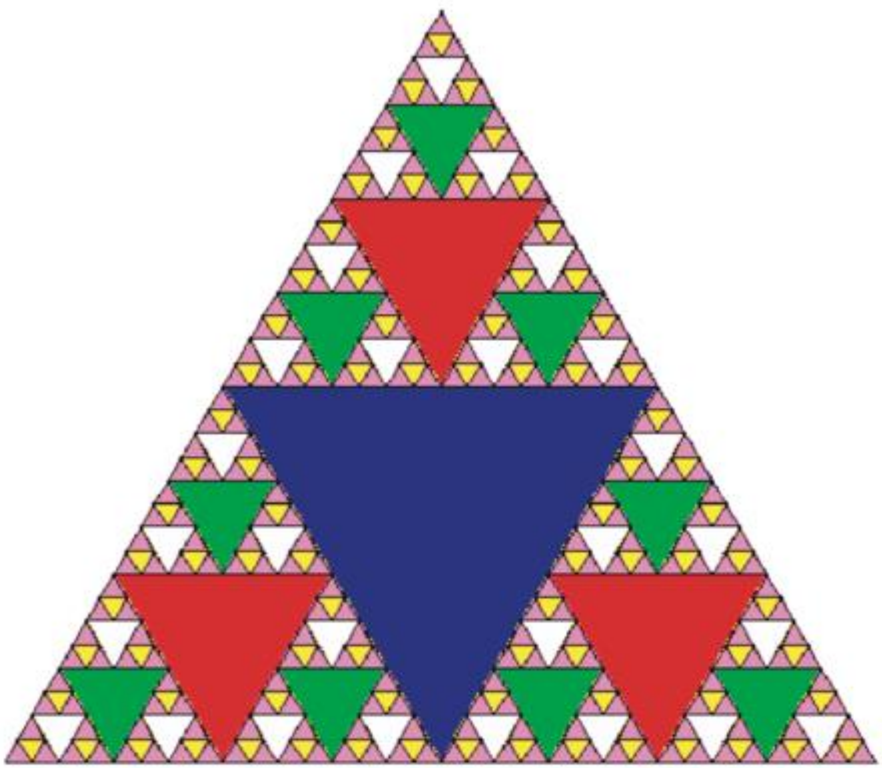




Sierpinski háromszög



Sierpinski háromszög



```
import turtle

def drawTriangle(points,color,myTurtle):
    myTurtle.fillcolor(color)
    myTurtle.up()
    myTurtle.goto(points[0][0],points[0][1])
    myTurtle.down()
    myTurtle.begin_fill()
    myTurtle.goto(points[1][0],points[1][1])
    myTurtle.goto(points[2][0],points[2][1])
    myTurtle.goto(points[0][0],points[0][1])
    myTurtle.end_fill()

def getMid(p1,p2):
    return ( (p1[0]+p2[0]) / 2, (p1[1] + p2[1]) / 2)

def sierpinski(points,degree,myTurtle):
    colormap = ['blue','red','green','white','yellow',
                'violet','orange']
    drawTriangle(points,colormap[degree],myTurtle)
    if degree > 0:
        sierpinski([points[0],
                        getMid(points[0], points[1]),
                        getMid(points[0], points[2])],
                    degree-1, myTurtle)
        sierpinski([points[1],
                        getMid(points[0], points[1]),
                        getMid(points[1], points[2])],
                    degree-1, myTurtle)
        sierpinski([points[2],
                        getMid(points[2], points[1]),
                        getMid(points[0], points[2])],
                    degree-1, myTurtle)


myTurtle = turtle.Turtle()
myWin = turtle.Screen()
myPoints = [[-100,-50],[0,100],[100,-50]]
sierpinski(myPoints,3,myTurtle)
myWin.exitonclick()
```

Fraktálok a természetben

1. Házi Feladat
https://viskillz.inf.unideb.hu/prog/#/?week=P1032

- Pitagorasz tétel (állományvégjelig olvasás)
- Pitagorasz tétel (n teszteset)
- Pitagorasz tétel (végjelig olvasás)

2. Házi Feladat
https://viskillz.inf.unideb.hu/prog/#/?week=P1043

- Lista valós osztása (állományvégjelig olvasás)
- Lista valós osztása (n teszteset)
- Lista valós osztása (végjelig olvasás)

3. Házi Feladat
https://viskillz.inf.unideb.hu/prog/#/?week=P1053

- Kisbetűk inkrementálása (állományvégjelig olvasás)
- Kisbetűk inkrementálása (n teszteset)
- Kisbetűk inkrementálása (végjelig olvasás)

3. Házi Feladat – Segítség

- chr() függvény
- ord() függvény

- Karakterkódok kezelése:

- ord("A")   # 65 a kódja
- chr(65)   # "A" betű

- chr(ord("A") + 3)   # "D", mert A → B → C → D