
# Rekurzija u jeziku Python

Dr Milan Paroški
mparoski@singidunum.ac.rs
Univerzitet Singidunum
2024/2025

# Sadržaj

# 1.Uvod

Rekurzija ?

**omogućava definisanje nekog pojma pomoću njega samog.**

**Rekurzija u programiranju je?**

**poseban način ponavljanja grupa naredbi pomoću funkcija koje pozivaju same sebe.**

Primeri rekurzivnih struktura, gde se u delovima ponavlja celina su:

- u svakodnevnom životu?

slika ogledala u ogledalu ili snimak ekrana koji prikazuje sliku same kamere;

- u matematici?

mnoge definicije su rekurzivne, npr. definicija prirodnog broja?:

(a) broj 1 je prirodni broj;

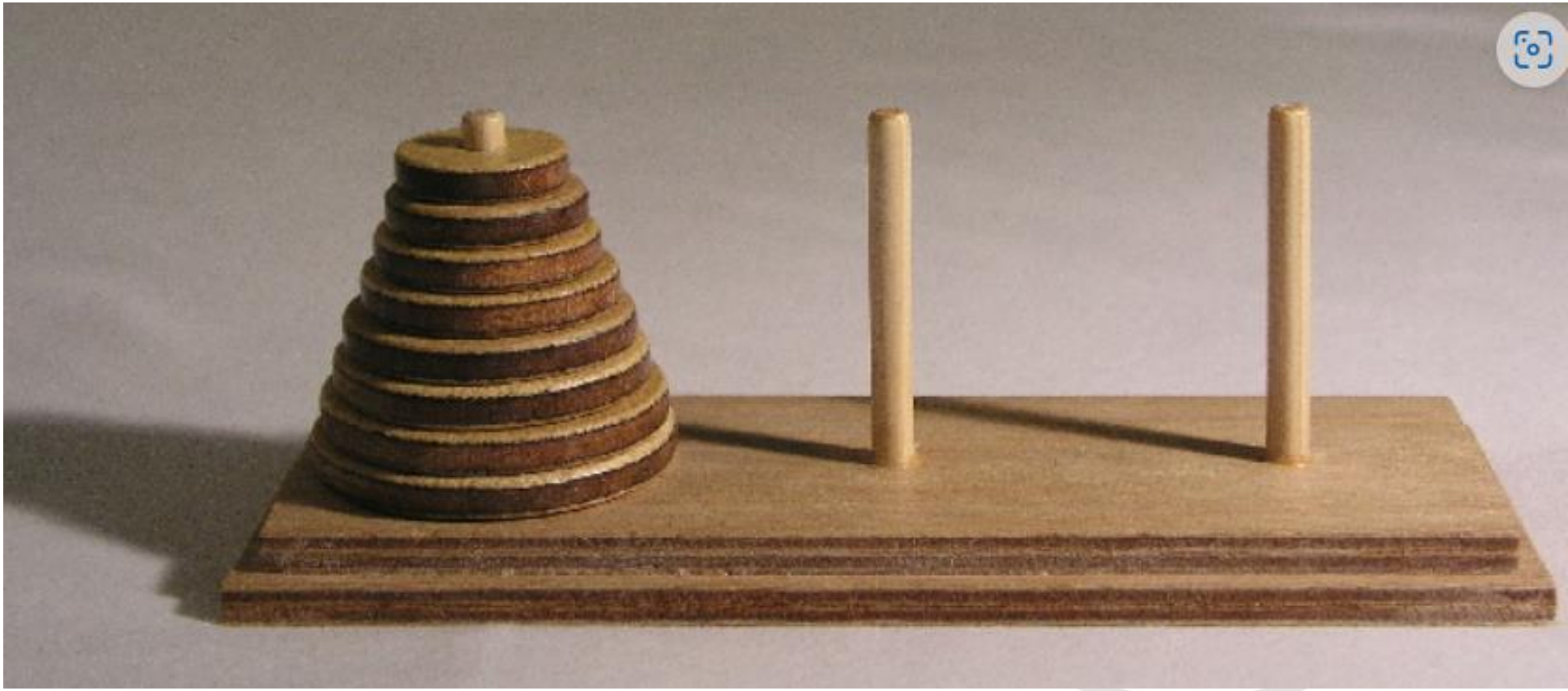
(b) ako je n prirodni broj, onda je n+1 takođe prirodni broj.

- u prirodi?

biološke strukture stabla i cveta itd.

# Kule Hanoja / matematička igra

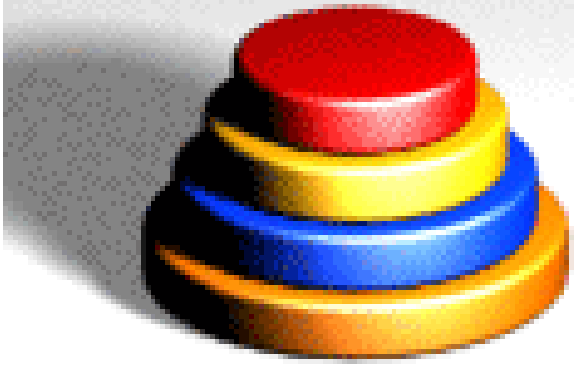
**Indijska legenda -** sveštenici treba da prenesu kulu od 64 zlatna diska sa jednog mesta na drugo, disk po disk, tako da veći uvek bude ispod a manji iznad, koristeći jedno privremeno mesto za spuštanje. Rečeno je da će se, pre nego što sveštenici završe svoj zadatak, hram rasuti u prah i svet nestati. Da li je moguće?



Problem kula Hanoja pojednostavljen glasi ovako: date su tri kule i na prvoj od njih n diskova opadajuće veličine; zadatak je prebaciti sve diskove sa prve na treću kulu (koristeći i drugu) ali tako da nikada nijedan disk ne stoji iznad manjeg

Animacija rešenja:

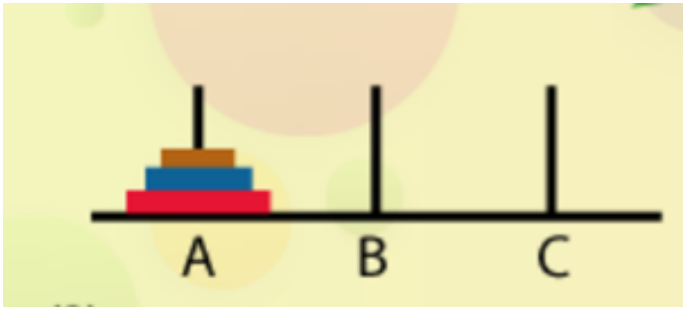
Škola koda | Kule Hanoja (skolakoda.github.io)

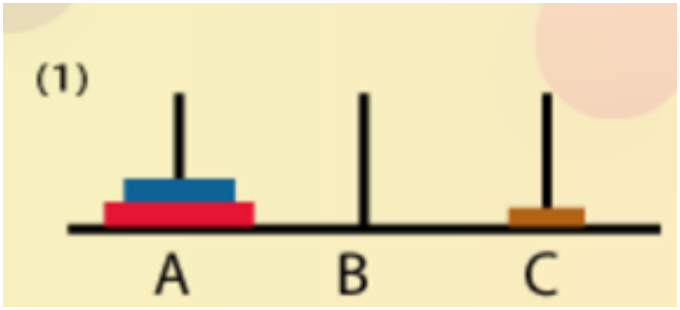


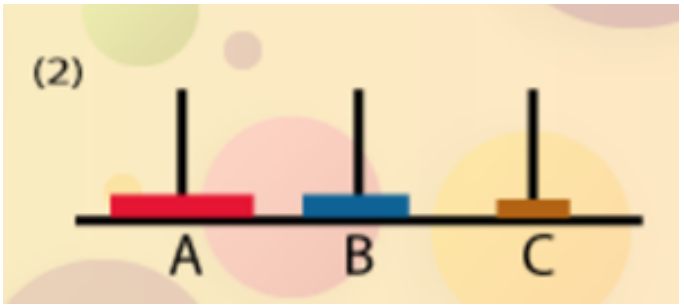
**Iterativno rešenje ovog problema je veoma kompleksno, a rekurzivno prilično jednostavno**

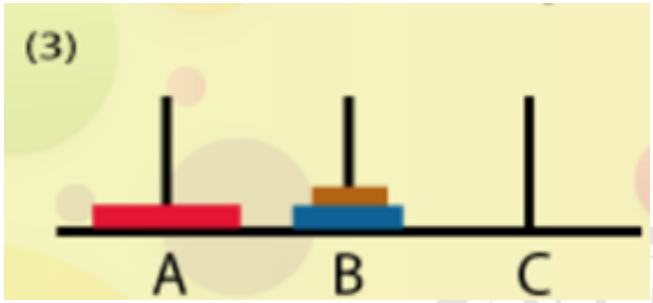
1. **prebaci (rekurzivno) n−1 diskova sa polazne na pomoćnu kulu (korišćenjem dolazne kule kao pomoćne),**
2. **prebaci najveći disk sa polazne na dolaznu kulu i, konačno,**
3. **prebaci (rekurzivno) n − 1 diskova sa pomoćne na dolaznu kulu (korišćenjem polazne kule kao pomoćne).**

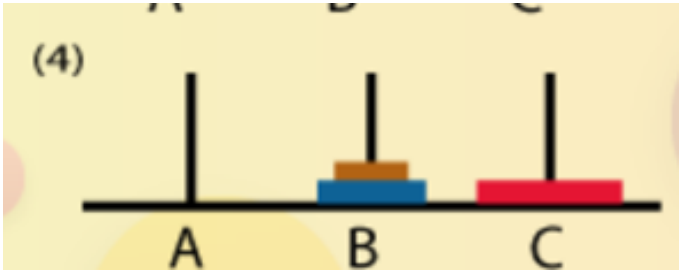
# Premeštati disk po disk, tako da veći uvek bude ispod a manji iznad : Za 3 diska :7 pomeranja



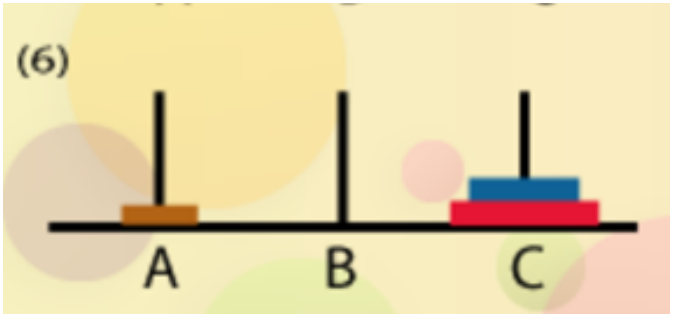



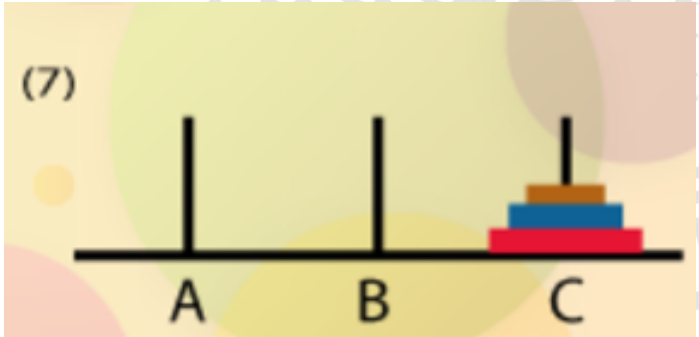



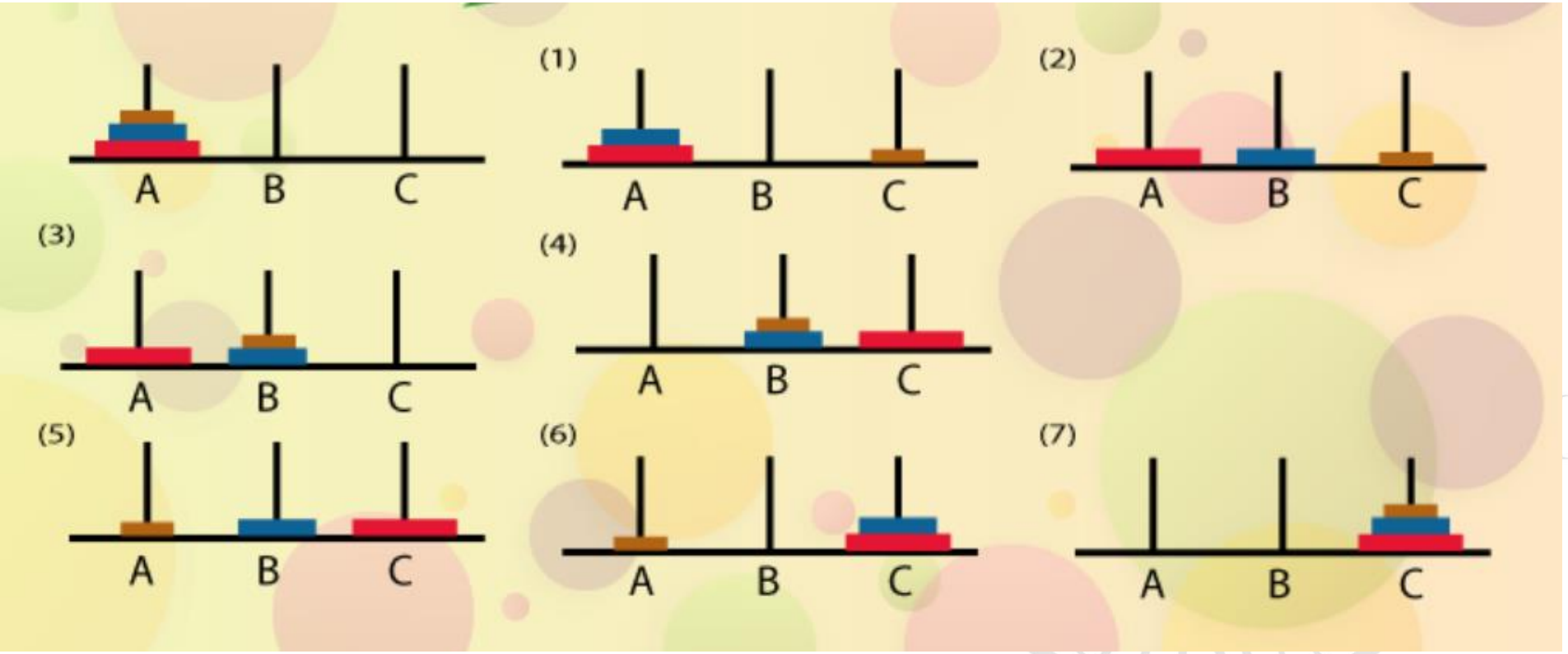








A
B
C
(1)
A
B
C
(2)
A
B
C
(3)
A
B
C
(4)
A
B
C
(5)
A
B
C
(6)
A
B
C
(7)
A
B
C

| Broj diskova | Broj pomeranja |
|---|---|
| 1? | 1 |
| 2? | 3 |
| 3? | 7 |
| 4? | 15 |
| 5? | 31 |

Kako se ova zakonitost može matematički izraziti?

| Broj diskova (n) | Broj pomeranja |
|---|---|
| 1 | $2^1-1=2-1=1$ |
| 2 | $2^2-1=4-1=3$ |
| 3 | $2^3-1=8-1=7$ |
| 4 | $2^4-1=16-1=15$ |
| 5 | $2^5-1=32-1=31$ |

Formula? $2^n-1$

# Primer 2

Isplata duga preko šahovskih polja..

Prvo šahovsko polje = 1 zrno pirindža

Drugo šahovsko polje = 2 zrna pirindža

Nadalje se duplira...

treće šahovsko polje = 4 zrno pirindža

četvrto šahovsko polje = 8 zrna pirindža

64-to šahovsko polje ?

Formula?

$$2^{n-1}$$

Excel fajl

| broj | 2 | 2 |
|---|---|---|
| n=eksponent | 4 | 64 |
| Rezulta $2^n$ | 16 | 18.446.744.073.709.600.000 |
| Rezulta $2^{n-1}$ | 8 | 9.223.372.036.854.780.000 |

Koja se funkcija u excelu koristi za izračunavanje eksponenta?

POWER

# 2 Definicija rekurzivne funkcije

Osnovni oblik definicije rekurzivne funkcije je:

def <nazivFunkcije>(<lista_parametara>):
<blok naredbi, s pozivima nazivFunkcije(...)>

Primer rekuzivne funkcije je funkcija :

```
def odbrojavanje(n):
    print(n)
    if n > 1:
        odbrojavanje(n-1)
print(odbrojavanje(5))        #izvršenje funkcije odbrojavanje
```

```
5
4
3
2
1
None
>>>
```

Šta radi ova funkcije?

odbrojavanja unazad:

Funkcija štampa redom brojeve n, n-1, ..., 2, 1. Rekurzivno poziva samu sebe s izmenjenom vrednošću argumenta.

**Kako se završava?**

**Za vrednost argumenta n=1 funkcija okončava bez novog rekurzivnog poziva**.

Nakon toga okončavaju sve rekurzivno pozvane funkcije, obrnutim redom od redosleda pozivanja, sve dok ne okonča i originalna funkcija.

```
def odbrojavanje(n):
    print(n)
    if n > 1:
        odbrojavanje(n-1)
    print("n=",n)
print(odbrojavanje(5))
```

Šta će se sada ispisavati?

Znači ovo može biti i funkcija za?

**Odbrojavanje unapred**

Šta bi se desilo da oduzimanje pretvorimo u sabiranje:

```
def odbrojavanje(n):
    print(n)
    if n > 1:
        odbrojavanje(n+1)
    print("n=",n)
print(odbrojavanje(5))
```

```
= RESTART: C:/Singidunum/Predmet Osnove programiranja - Python/2021-2022/Predavanja/primeri/Odbrojavanje unazad izmedju 2 broja.py
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

Funkcija se može modifikovati tako da odbrojavanje bude između dva zadana broja, unapred ili unazad.

**Algoritam rekurzivne funkcije za brojanje unapred je:**

prvo se rekurzivno pozove funkcija za prikaz sledećeg broja,

a zatim prikaže broj n2 > n1;

za n2 ≤ n1 samo se prikaže broj i završava proces pozivanja.

Implementacije funkcije u jeziku Python je:

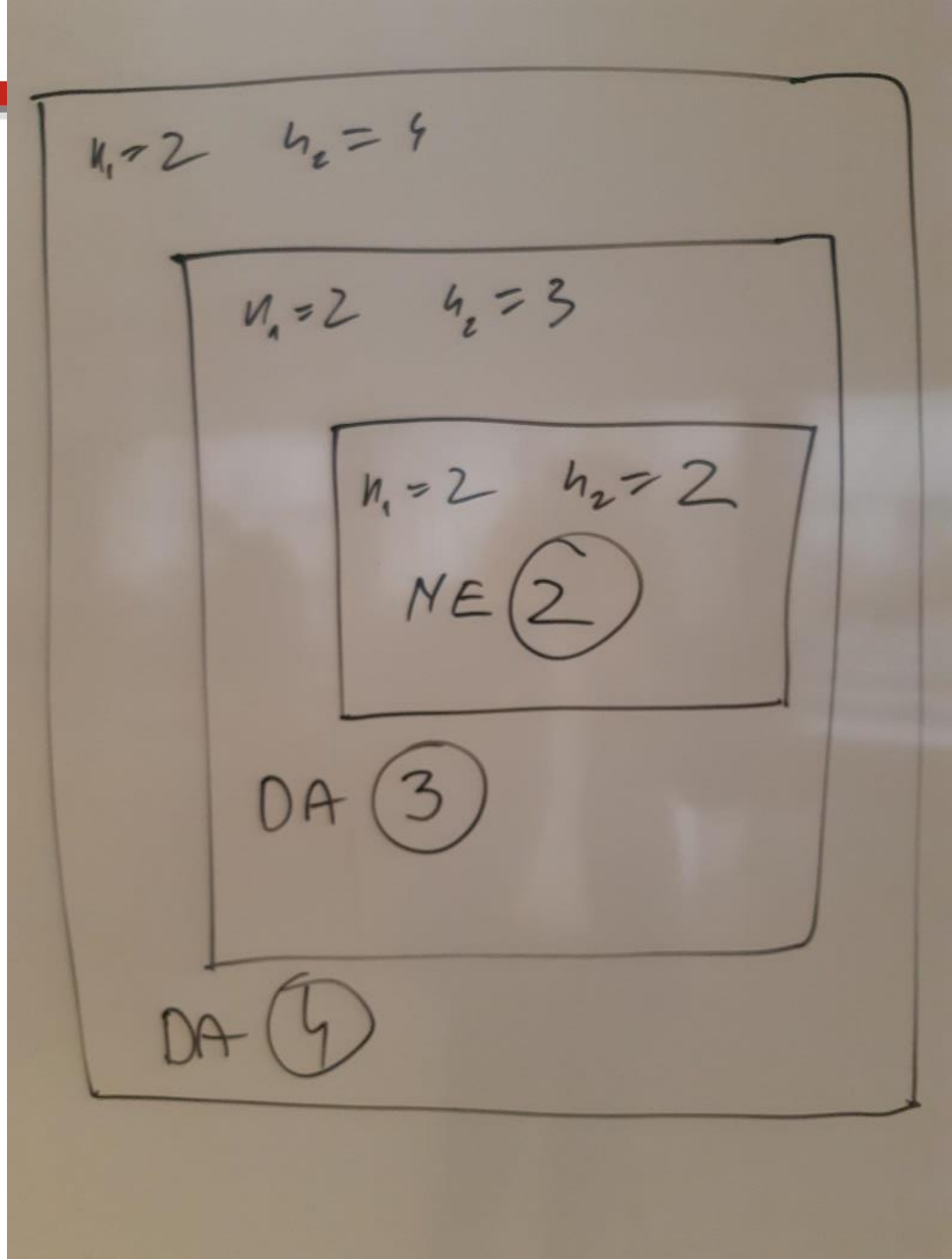
```
def broj_unapred(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print("DA",n2)
    else:
        print("NE",n2)
print(broj_unapred(2,4))                #izvršenje funkcije
```

***n1= 2 n2= 4***
***n1= 2 n2= 3***
***n1= 2 n2= 2***
***NE 2***
***DA 3***
***DA 4***

# Brojanje unapred

```
def broj_unapred(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print("DA",n2)
    else:
        print("NE",n2)
print(broj_unapred(2,4))
```

```
n1= 2 n2= 4
n1= 2 n2= 3
n1= 2 n2= 2
NE 2
DA 3
DA 4
None
```



Pitanje: Zašto se štampa **None**?

```
def broj_unapred(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print("DA",n2)
    else:
        print("NE",n2)
print(broj_unapred(2,4))
```

```
n1= 2 n2= 4
n1= 2 n2= 3
n1= 2 n2= 2
NE 2
DA 3
DA 4
None
```

**None** se štampa kada funkcija ne vrati eksplicitno vrednost, odnosno kad je rezultat poziva funkcije podrazumevani povratak None.

To se dešava kada:

**1.Funkcija nema return izjavu**: Ako funkcija završi bez return izraza, Python automatski dodeljuje None kao povratnu vrednost.

Na primer:

```
def pozdrav():
    print("Zdravo!")

rezultat = pozdrav()
print(rezultat)
```

```
Zdravo!
None
```

Kada prethodna funkcija neće ispisati None?

```
def pozdrav():
    print("Zdravo!")

rezultat = pozdrav()
print(rezultat)
```

Kada se umesto dva poslednja reda napiše:

```
def pozdrav():
    print("Zdravo!")

#rezultat = pozdrav()
#print(rezultat)
pozdrav()
```

Šta će se sada ispisati?

```
def pozdrav():
    print("Zdravo!")
    return "Pozdrav je završen"

rezultat = pozdrav()
print(rezultat)
```

```
Zdravo!
Pozdrav je završen
```

# 2.čas

```
def broj_unapred(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print("DA",n2)
    else:
        print("NE",n2)
broj_unapred(2,4)
```

```
n1= 2 n2= 4
n1= 2 n2= 3
n1= 2 n2= 2
NE 2
DA 3
DA 4
```

Brojanje unapred bez pomoćnih štampi:

```
def broj_unapred(n1, n2):
    #print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print(n2)
    else:
        print(n2)
broj_unapred(2,4)
```

Ovo je rezultat:

2
3
4

Kako prethodni program da broji unazad ?

jednostavnom zamenom redova Bold/Italic ?

4
3
2

```
def broj_unapred(n1, n2):
    #print(n1,n2)
    if n2 > n1:
        print(n2)
        broj_unapred(n1, n2-1)
    else:
        print(n2)
broj_unapred(2,4)
```

Sa prethodnog slajda:

```
def broj_unapred(n1, n2):
    #print("n1=",n1,"n2=",n2)
    if n2 > n1:
        broj_unapred(n1, n2-1)
        print(n2)
    else:
        print(n2)
broj_unapred(2,4)
```

Algoritam rekurzivne funkcije za brojanje unazad: je:

prvo se prikaže broj n2>n1, a zatim rekurzivno pozove funkcija za prikaz sledećeg broja;

za n2 ≤ n1 samo se prikaže broj i završava proces pozivanja.

Implementacije ove funkcije u jeziku Python je:

```
def broj_unazad(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        print("DA",n2)
        broj_unazad(n1, n2-1)
        print("Kraj poziva",n2)
    else:
        print("NE",n2)
broj_unazad(2,4)
```

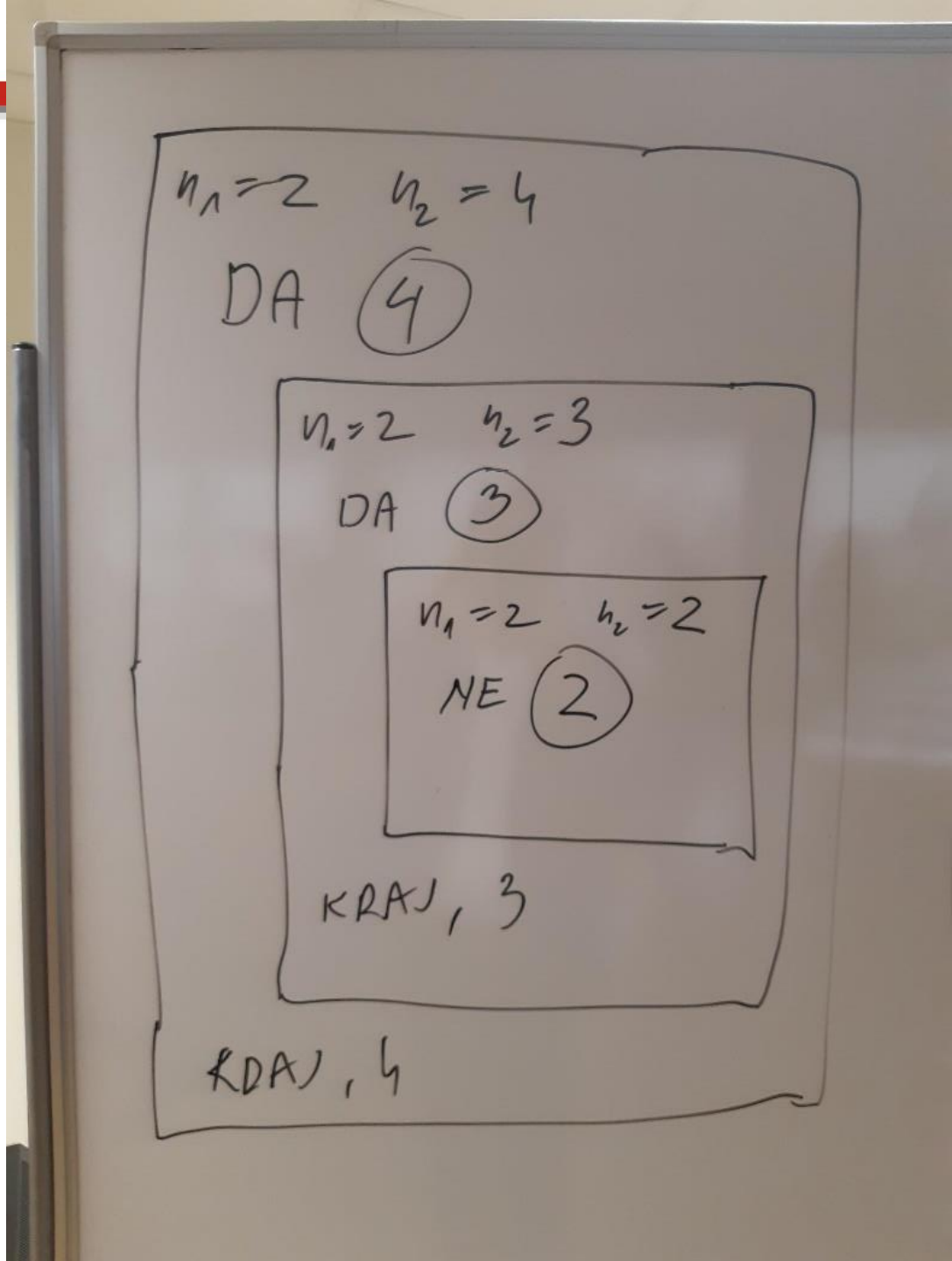
```
n1= 2 n2= 4
DA 4
n1= 2 n2= 3
DA 3
n1= 2 n2= 2
NE 2
Kraj poziva 3
Kraj poziva 4
>>>
```

# Brojanje unazad

```
def broj_unazad(n1, n2):
    print("n1=",n1,"n2=",n2)
    if n2 > n1:
        print("DA",n2)
        broj_unazad(n1, n2-1)
        print("Kraj poziva",n2)
    else:
        print("NE",n2)
broj_unazad(2,4)
```

**REZULTAT IZVRŠAVANJA PROGRAMA:**

```
n1= 2 n2= 4
DA 4
n1= 2 n2= 3
DA 3
n1= 2 n2= 2
NE 2
Kraj poziva 3
Kraj poziva 4
>>>
```



# Faktorijel

Funkcija faktorijel u matematici se obično definiše rekurzivno. Zašto?

$n! = n \cdot (n-1)!$

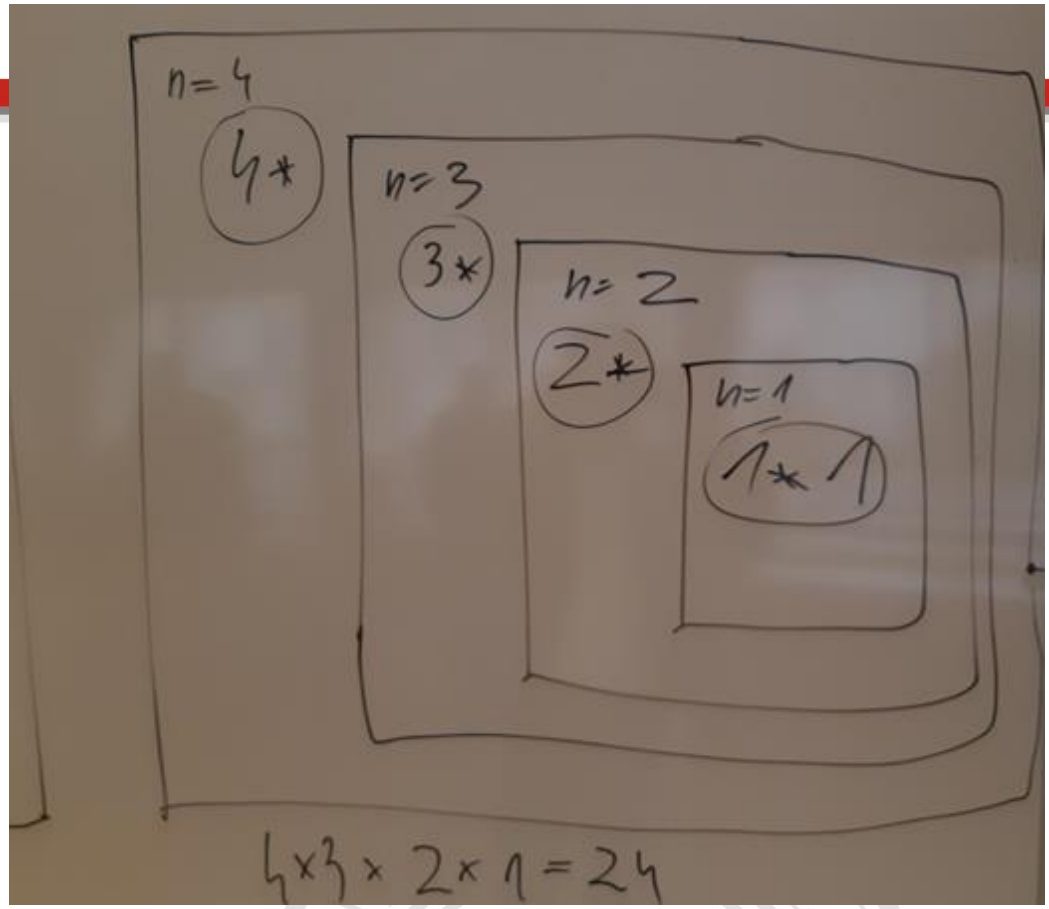
U jeziku Python rekurzivna funkcija n! može se realizovati direktno na osnovu ove rekurzivne definicije kao:

```
def faktorijel(n):
    if n == 0:
        return 1  #Zašto se ovo piše?
                  #0! = 1
    else:
        return n*faktorijel(n-1)
print(faktorijel(5))
```

```
def faktorijel(n):
    if n == 0:
        return 1
    else:
        print(n)
        return n*faktorijel(n-1)


print("Rezultat je ",faktorijel(13))
```

# FAKTORIJEL

```
def faktorijel(n):
    if n == 0:
        return 1
    else:
        print(n)
        return n*faktorijel(n-1)
print("Rezultat je ",faktorijel(4))
```




**REZULTAT IZVRŠAVANJA PROGRAMA:**

```
4
3
2
1
Rezultat je  24
```

Rekurzivno računanje faktorijela pomoću funkcije iz prethodnog primera može se prikazati sledećom sekvencom:

faktorijel(5) =

= 5 · faktorijel(4)

= 5 · (4 · faktorijel(3))

= 5 · (4 · (3 · faktorijel(2)))

= 5 · (4 · (3 · (2 · faktorijel(1))))

= 5 · (4 · (3 · (2 · 1)))

= 5 · (4 · (3 · 2))

= 5 · (4 · 6)

= 5 · 24

= 120

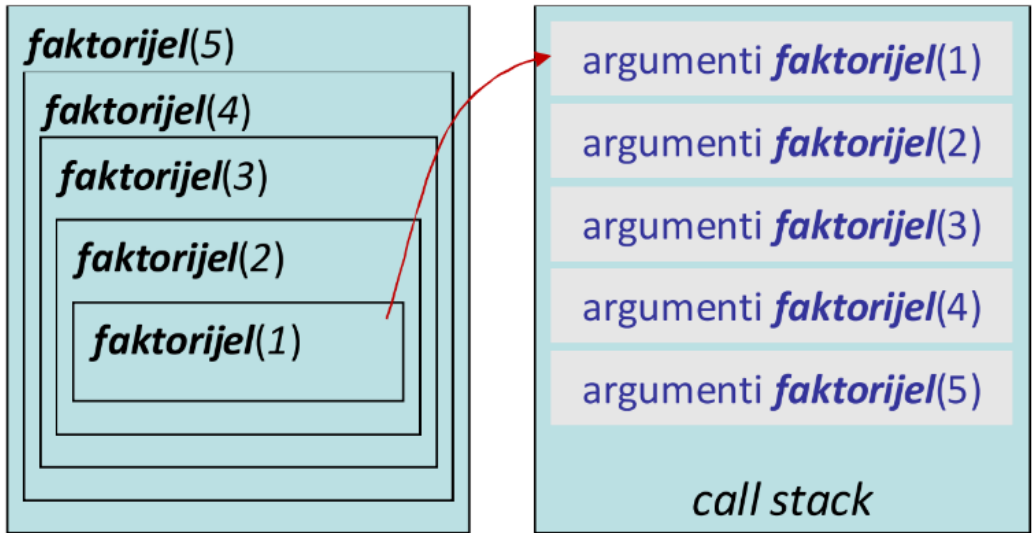
- U prvoj fazi računanja vrednost aktuelnog argumenta **n** se pamti u steku, a zatim se rekurzivnim pozivom istoj funkciji prenosi vrednost n-1.
- Proces rekurzivnog pozivanja okončava kad vrednost argumenta n postane nula, kada se prvoj funkciji pozivniku vraća vrednost 1.
- Vraćena vrednost se množi s argumentom iz steka i proizvod vraća kao rezultat.

Postupak se nastavlja, sve dok se stek poziva ne isprazni, odnosno dok se ne vrati konačni rezultat funkcije.

Prikazan je izgled steka u fazi rekurzivnog pozivanja, pre početka vraćanja kontrole i množenja argumenata s vrednostima iz steka.

Svaki rekurzivni poziv dodaje na vrh steka vrednosti argumenata funkcije iz poziva.

```
def faktorijel(n):
    if n == 0:
        return 1
    else:
        print(n)
        return n*faktorijel(n-1)
print("Rezultat je ",faktorijel(5))
```



# 3. Izvršavanje rekurzivne funkcije

U jeziku Python su dozvoljeni rekurzivni pozivi funkcija.

Radi sprečavanja namerne ili slučajne beskonačne rekurzije, u interpreter jezika Python definisana je maksimalna dubina rekurzivnih poziva funkcija, koja je inicijalno 1.000 poziva.

Trenutna vrednost ograničenja dubine rekurzivnih poziva uvek se može ustanoviti pomoću ugrađene funkcije sys.getrecursionlimit().

```
import sys
print("rezultat ogranicenja je=",sys.getrecursionlimit())
```

Rezultat ogranicenja je= 1000

Pošto je u jeziku Python dubina rekurzivnih poziva ograničena, funkcija koja predstavlja beskonačnu rekurziju ipak okončava.

Kako ovo simulirati?

```
def rekurzija():
    rekurzija()

rekurzija()
```

Poziv ovako definisane funkcije daje rezultat:

```
>>> rekurzija()
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
rekurzija()
File "<pyshell#1>", line 2, in rekurzija
rekurzija()
...
RecursionError: maximum recursion depth exceeded
>>>
```

# 4. Primeri rekurzivnih algoritama

Radi ilustracije praktične upotrebe rekurzivnih algoritama, daju se tipični primeri upotrebe rekurzivnih funkcija:

1. Fibonačijev niz
2. Vektorska grafika u jeziku Python (turtle graphics)
3. Fraktali i fraktalne funkcije

# 4.1 Fibonačijev niz

Italijanski matematičar Leonardo Bonači (Fibonači) razmatrao je rast populacije zečeva u polju uz (veoma pojednostavljene) pretpostavke:

- ✓ zečevi su u stanju da se pare u dobu od mesec dana,
- ✓ na kraju svog drugog meseca ženka može da okoti još jedan par zečeva;
- ✓ zečevi ne umiru
- ✓ jedan par zečeva nakon parenja uvek daje novi par, od drugog meseca pa nadalje, i to uvek samo po jednog muškog i jednog ženskog potomka.

Fibonačija je zanimalo: Početak je jedan par zečeva. Koliko parova zečeva će biti u polju nakon godinu dana?

Na kraju prvog meseca ?

postoji samo **jedan par.**, ali na kraju prvog meseca mogu da se pare

Na kraju drugog meseca?

još uvek **1 par**, jer nakon parenja se uvek dobija novi par, od drugog meseca pa nadalje

Na kraju trećeg meseca?

Prva ženka okoti PRVI par, tako da postoje **dva para** zečeva u polju.

Na kraju četrvrtog meseca?

prva ženka okoti svoj DRUGI par zečeva, a sada drugi par može da se pari pa je svega **tri para** u polju.

Na kraju petog meseca?

prva ženka okoti svoj TREĆI par, a ženka rođena pre dva meseca okoti svoj PRVI par, tako da je ukupno **pet parova** zečeva u polju.

gde je svaki sledeći broj zbir prethodna dva.

Na kraju n-tog meseca?

broj parova jednak je broju postojećih parova (od meseca n-1) uvećan za broj novih parova (od meseca n-2). Rekurzija?

DA. Fibonačijev n-ti broj definiše se rekurzivno, uz pretpostavke:

F(1) =?

1, **na kraju prvog meseca zečevi tek mogu da se pare, tako da na kraju prvog meseca imamo 1 par početnih zečeva**

**F(2) = ?**

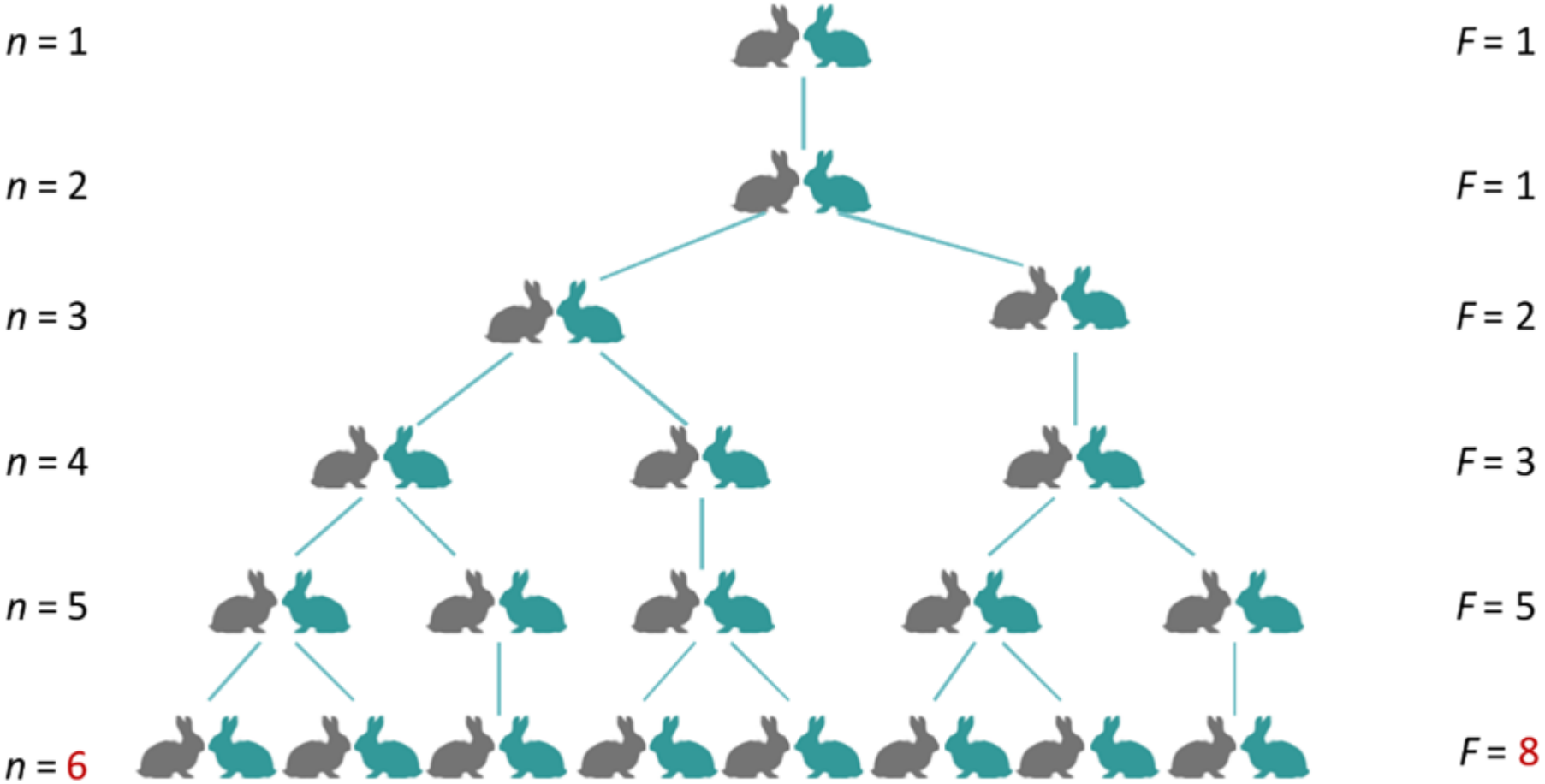
**1, na kraju drugog meseca imamo takođe 1 par zečeva, jer se zečevi pare na kraju prvog, ali nakon parenja se dobija novi par tek od drugog meseca**

F(n) = ?

F(n-1) + F(n-2)

F(1)=1
F(2)= F(1)=1
F(3)=F(2)+ F(1)=1+1=2par
F(4)=F(3)+ F(2)=2+1=3para
F(5)=F(4)+ F(3)=3+2=5para
svaki sledeći broj je zbir prethodna dva.

# 3.čas




za 6 meseci $F(6) = 8$

za godinu dana $F(12) = 144$

Fibonačijev broj: prikaz rasta populacije zečeva za n=6 meseci

Jednostavni program u jeziku Python za računanje populacije nakon određenog vremenskog perioda je :

```
def fib(n):
    if n == 1: # prvi nivo
        return 1
    elif n == 2: # drugi nivo
        return 1
    else: # Rekurzivni pozivi
        return fib(n-1) + fib(n-2)
n = int(input("Unesite indeks za Fibonačijev broj: "))
print("Fibonačijev broj za indeks", n, "je", fib(n))
```

Primer računanja Fibonačijevog broja, odnosno populacije nakon 24 meseca je:

Unesite indeks za Fibonačijev broj: 24

Fibonačijev broj za indeks 24 je 46368

# 4.2 Vektorska grafika u jeziku Python (turtle graphics)

Vektorska grafika je način kreiranja grafičkih prikaza pomoću osnovnih geometrijskih objekata.

Poseban pristup kreiranju vektorske grafike je tzv. Turtle grafika:

- objekti se crtaju pomoću relativnih koordinata i
- Komande za crtanje geometrijskih objekata se izdaju zamišljenoj olovci, popularno nazvanoj "kornjača" (turtle).

Olovka ili "kornjača" se na ekranu prikazuje u obliku malog trougla, koji pokazuje smer crtanja .

Za korišćenje ovakvih grafičkih naredbi u jeziku Python potrebno je prethodno uključiti odgovarajuću programsku biblioteku naredbom **import turtle**.

Neke od osnovnih funkcija iz ove biblioteke su, npr.

**pendown() ?**

spuštanje olovke, nakon čega olovka prilikom pomeranja crta liniju odgovarajuće **dužine, boje i debljine,** koji se mogu podesiti;

**Left, right?**

**left(ugao), right(ugao)** - promena smera strelice u izabranom smeru, za zadani ugao;

**Forward, back?**

**forward(rastojanje), back(rastojanje)** - pomeranje olovke za zadani broj piksela u smeru strelice ili u suprotnom smeru.

Crtanje jednostavnih geometrijskih figura vrši se postavljanjem olovke na izabranu početnu tačku ekrana s koordinatama (x, y), nakon čega se linije crtaju promenom smera i pomeranjem olovke i izabranom smeru za određeni broj tački ekrana (piksela).

Zadatak:

Napraviti program za crtanje pravougaonika, čije su dužine stranica 100 piksela od podrazumevajuće početne pozicije olovke (0,0):

```
import turtle
turtle.pendown()
turtle.forward(100)     #zašto 100?
turtle.right(90)        #zašto 90?
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
turtle.forward(100)
turtle.right(90)
```

UZ POMOĆ PETLJE?

```
import turtle
turtle.pendown()
for i in range(4): #zašto 4?
        #broj stranica
   turtle.forward(100)  #zašto 100?
#dužina stranice
   turtle.right(90)        #zašto 90?
#ugao se menja nakon crtanja svake stranice
```

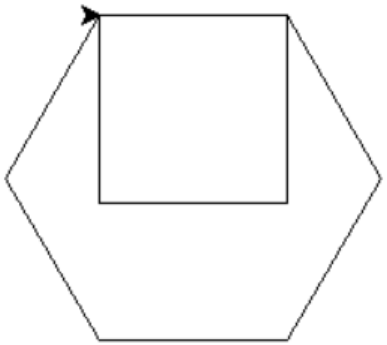
Zadatak:

Napraviti program za crtanje šestougla, čije su dužine stranica 100 piksela od početne pozicije olovke:

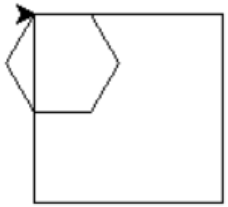
```
import turtle
turtle.pendown()   # sledeći korak je petlja sa koliko koraka?
for i in range(6):    #šta je sledeća komanda?
    turtle.forward(100)     #šta je sledeća komanda?
    turtle.right(60)
```

Spojiti ova dva programa i menjati dužinu stranice šestougaonika ?
Spaja se boldovani tekst
turtle.forward 100,
pa 30,
pa 150

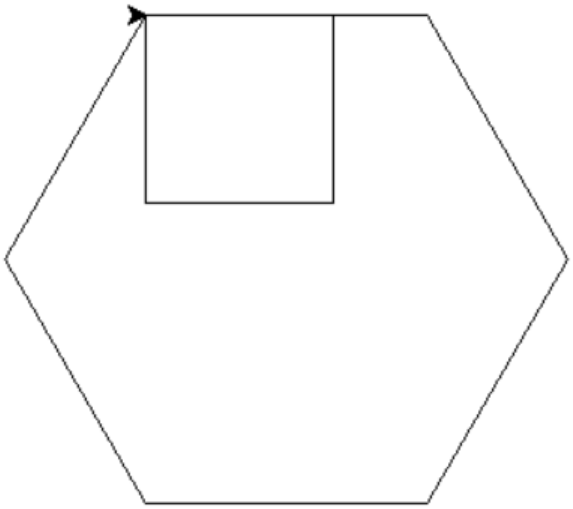
100,60



30,60

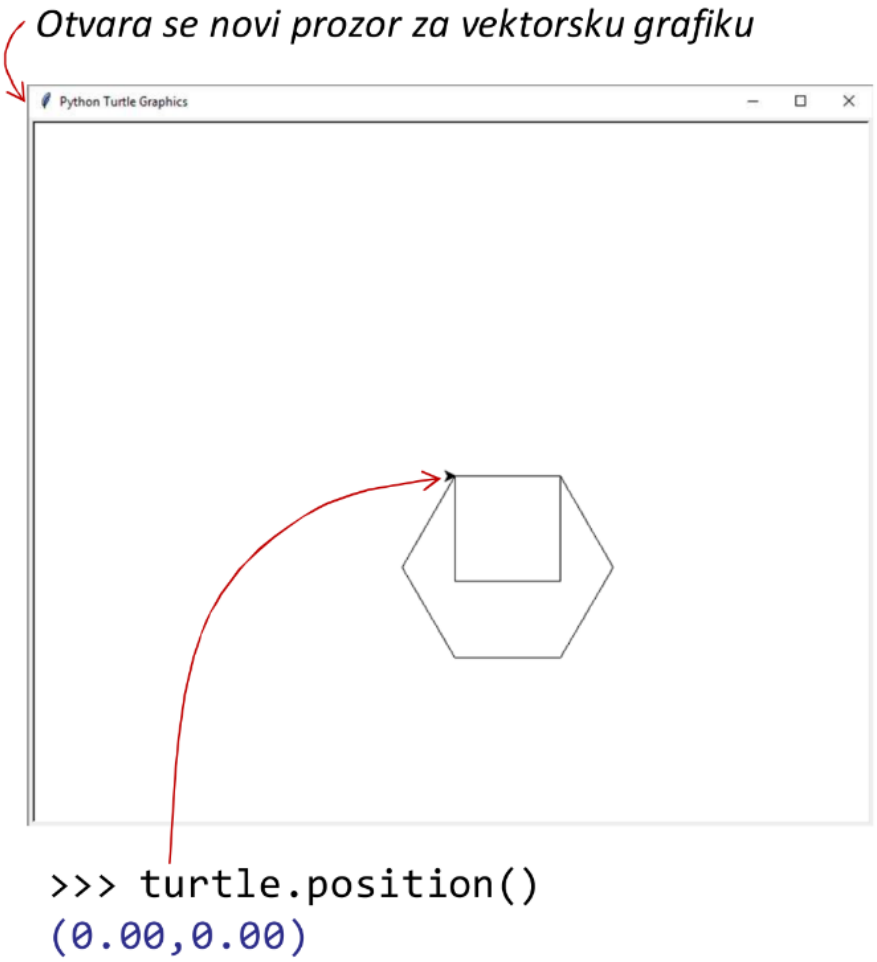


150,60



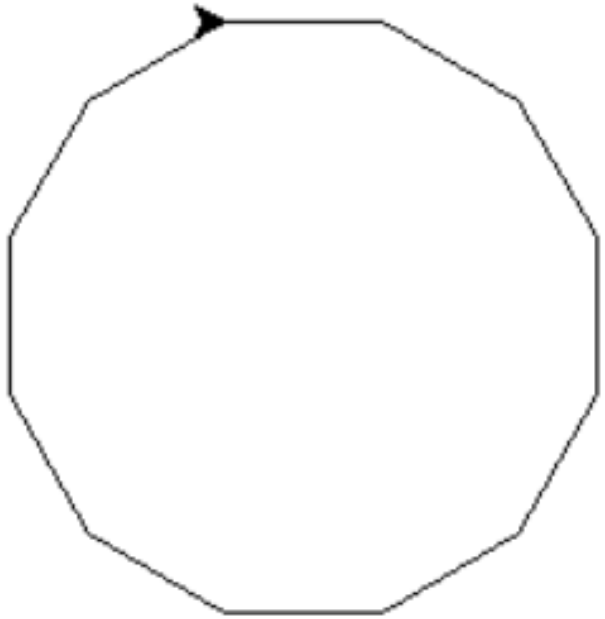
Inače, pokretanjem programa otvara se novi grafički prozor za prikaz vektorske grafike.

Pokazivač (olovka, "kornjača") na početku je u težištu grafičkog prozora, koje ima koordinate x=0 i y=0,



Kako nacrtati 12-ugao?

```
import turtle
turtle.pendown()
for i in range(12):
    turtle.forward(50)
    turtle.right(30)
```



Zašto je i do 12?

Zašto je right 30?

Podrazumevajuća boja pozadine ekrana je bela, a boja pera crna.

Pomoću metoda turtle.color() može se izabrati druga boja, koja se zadaje na tri načina:

1. **kao tekst (engleski naziv ili Veb kod boje)**
2. **numerički, pomoću RGB komponenti.**
3. ***Ako se postavi turtle.colormode(255)***

```
import turtle
turtle.color('red')
turtle.forward(100)
```

```
import turtle
turtle.color('#FF0000') # Veb kod crvene boje
turtle.forward(100)
```

```
import turtle
turtle.colormode(255)
turtle.color(255, 0, 0) # numerički kod crne boje
turtle.forward(100)
```

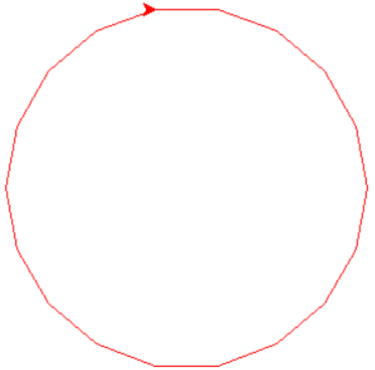
Primeri:

```
import turtle
turtle.colormode(255)
turtle.color(255, 0, 0) # numerički kod crvene boje
turtle.forward(100)
```

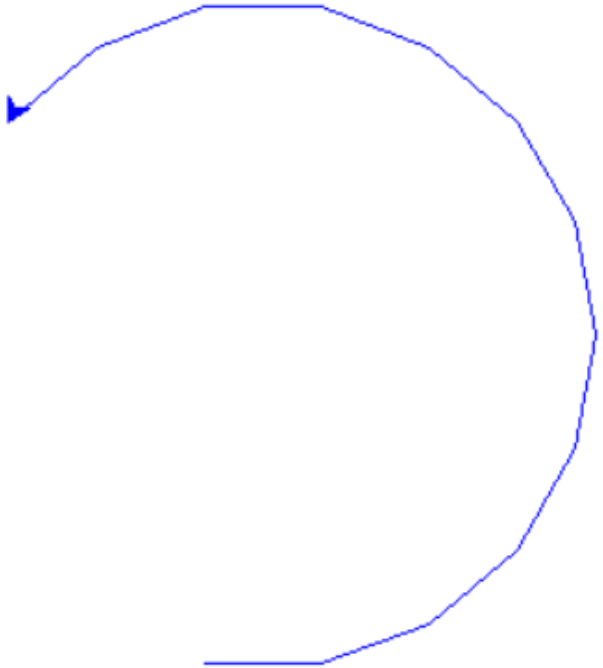
```
import turtle
turtle.color('#0000FF')
turtle.forward(100)
```

Kako nacrtati 18to ugaonik crvene boje?

```
import turtle
turtle.pendown()
turtle.color('red')
for i in range(18):
    turtle.forward(50)
    turtle.right(20)
```



```
import turtle
turtle.pendown()
turtle.color('blue')
for i in range(12):
    turtle.forward(50)
    turtle.left(20)
#šta će se nacrtati?
```



# Pozicija kursora

```
import turtle
print(turtle.position())
(0.00,0.00)
```

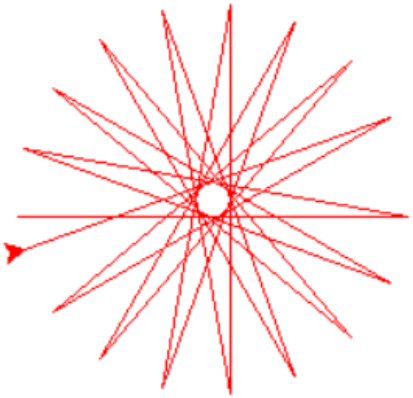
```
import turtle
turtle.forward(325)
print(turtle.position())
(325.00,0.00)
```

```
import turtle
turtle.forward(-75)
print(turtle.position())
(-75.00,0.00)
```

# Crtanje cveta

```
import turtle
turtle.color('red')
for i in range(18):
    turtle.forward(200)   #dužina
    turtle.left(170)      #ugao
    print(turtle.position())
```

```
(200.00,0.00)
(3.04,34.73)
(190.98,-33.67)
(17.77,66.33)
(170.98,-62.23)
(42.42,90.98)
(142.42,-82.23)
(74.02,105.71)
(108.75,-91.25)
(108.75,108.75)
(74.02,-88.21)
(142.42,99.73)
(42.42,-73.48)
(170.98,79.73)
(17.77,-48.83)
(190.98,51.17)
(3.04,-17.23)
(200.00,17.50)
>>>
```



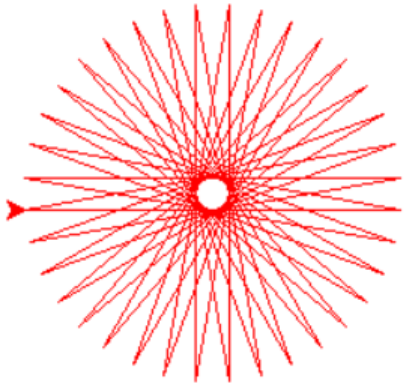
Šta bi bilo kada bi ugao promenuli sa 170 na 180?:

```
import turtle
turtle.color('red')
for i in range(18):
    turtle.forward(200)   #dužina
    turtle.left(180)      #ugao
    print(turtle.position())
```

# Debljina linije

```
import turtle
turtle.color('red')
turtle.pensize(4)
for i in range(18):
    turtle.forward(200)   #dužina
    turtle.left(170)      #ugao
    print(turtle.position())
```

Iterativni program za crtanje oblika koji podseća na cvet ili zvezdicu određene boje, zadane funkcijom turtle.color(), prikazan je na slici:



(200.00,0.00)
(3.04,34.73)
(190.98,-33.67)
(17.77,66.33)
(170.98,-62.23)
(42.42,90.98)
(142.42,-82.23)
(74.02,105.71)
(108.75,-91.25)
(108.75,108.75)
(74.02,-88.21)
(142.42,99.73)
(42.42,-73.48)
(170.98,79.73)
(17.77,-48.83)
(190.98,51.17)
(3.04,-17.23)
(200.00,17.50)
(0.00,17.50)
(196.96,-17.23)
(9.02,51.17)
(182.23,-48.83)
(29.02,79.73)
(157.58,-73.48)
(57.58,99.73)
(125.98,-88.21)
(91.25,108.75)
(91.25,-91.25)
(125.98,105.71)
(57.58,-82.23)
(157.58,90.98)
(29.02,-62.23)
(182.23,66.33)
(9.02,-33.67)
(196.96,34.73)
(0.00,-0.00)

```
import turtle
turtle.color('red')
for i in range(36):
    turtle.forward(200)   #dužina
    turtle.left(170)      #ugao
    print(turtle.position())
```

Ovo sporo crta

Obična brzina:turtle.speed()

Povećanje brzine:turtle.speed(neki broj)

```
import turtle
turtle.color('red')
turtle.speed(20)
for i in range(40):
    turtle.forward(200)   #dužina
    turtle.left(170)      #ugao
    print(turtle.position())
```

Univerzitet Singidunum
www.singidunum.ac.rs

# 18.11.2024.

# Popunjavanje pozadine

```
import turtle
turtle.pendown()
turtle.begin_fill()
for i in range(4):          #broj stranica
    turtle.forward(100)       #dužina
    turtle.right(90)          #ugao
turtle.end_fill()
```

```
import turtle
turtle.pendown()
turtle.color("yellow")
turtle.begin_fill()
for i in range(4):          #broj
stranica
    turtle.forward(100)       #dužina
    turtle.right(90)          #ugao
turtle.end_fill()
```

# 4.3 Fraktali i fraktalne funkcije

Fraktali su geometrijski objekti koji pružaju isti nivo detalja bez obzira na stepen uvećanja pod kojim se posmatraju.

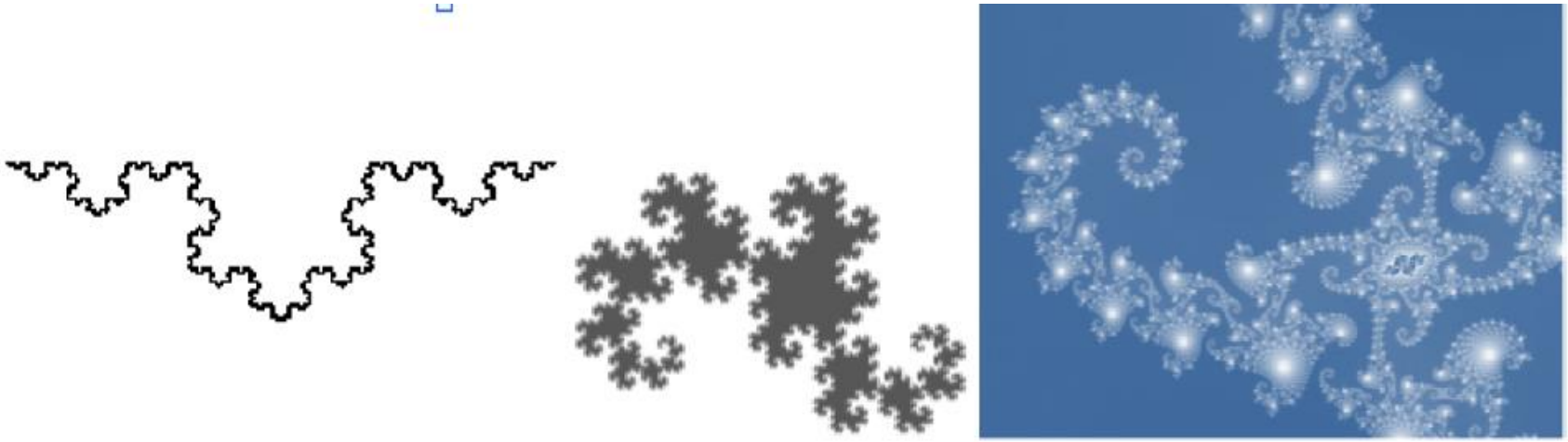
Imaju rekurzivnu strukturu, jer se sastoje od umanjenih verzija samih sebe, ali su suviše nepravilni da bi se opisali nekom jednostavnom geometrijom.

Neke od poznatih fraktalnih funkcija prikazane se na slici:

Kohova kriva,

Zmajeva kriva (dragon curve) https://youtu.be/m4-ILvsOFGo

Mandelbrotove krive.



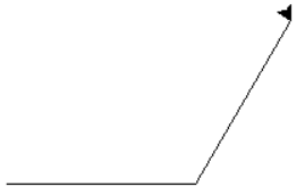
Kohova kriva, zmajeva kriva i uvećan detalj Mandelbrotove krive

# Postupno crtanje Kohove krive

```
import turtle
turtle.forward(400)
```

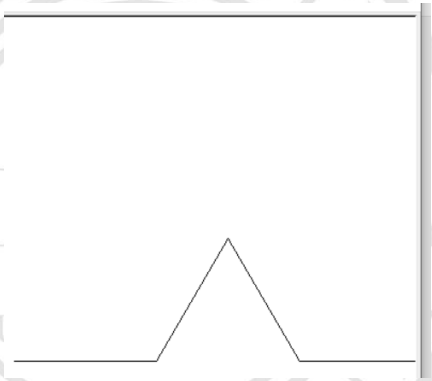
```
import turtle
duzina=400
for ugao in (60, -120):
        turtle.forward(duzina/3)
        turtle.left(ugao)
print(ugao)
```

```
60
-120
>>>
```



```
import turtle
duzina=400
for ugao in (60, -120,60,0):
        turtle.forward(duzina/3)
        turtle.left(ugao)
 print(ugao)
```

```
60
-120
60
0
```



Zašto se piše dužina/3?

Da se na unetoj dužine sve nacrta

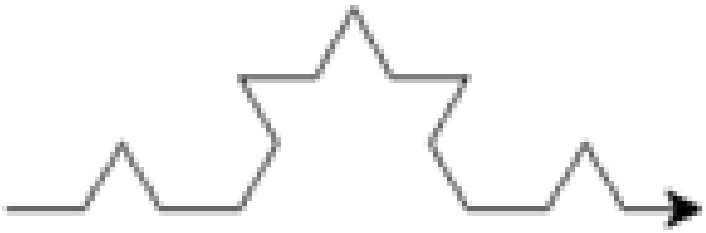
# Kohova kriva / ubacujemo rekurziju

```
import turtle
def koh(duzina,dubina):
    if dubina > 0 :
        for ugao in (60, -120,60,0):
            print("da",duzina,ugao,dubina)
            koh(duzina/3, dubina-1)
            turtle.left(ugao)
            print(ugao)
    else:
        turtle.forward(duzina)
        print("ne")
#Pozivanje funkcije koh
koh(100,1)
```

```
da 100 60 1
ne
60
da 100 -120 1
ne
-120
da 100 60 1
ne
60
da 100 0 1
ne
0
```

Koh(400,3)

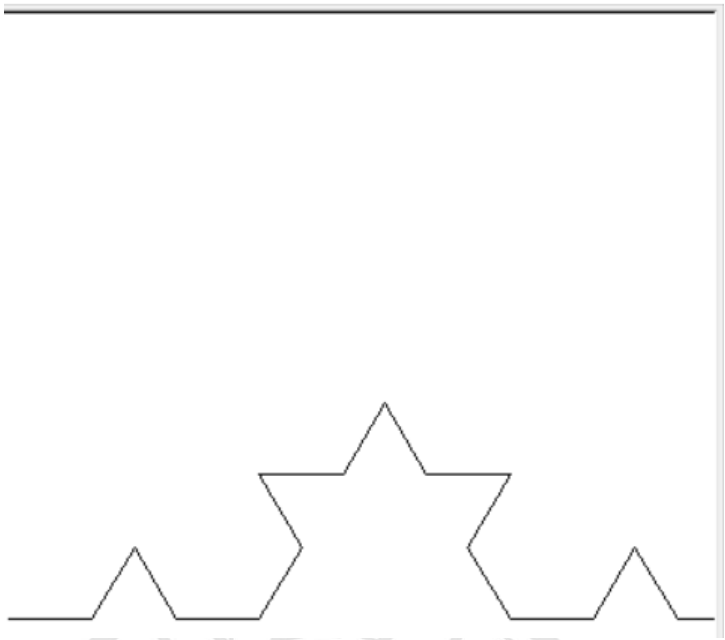
Koh(200,2)



Koh(100,1)

# Kako bi peške – bez petlje i rekurzije nacrtali Koh(100,2)?

```
import turtle
duzina=400
for ugao in (60, -120,60,0):
    turtle.forward(duzina/9)
    turtle.left(ugao)

turtle.left(60)
for ugao in (60, -120,60,0):
    turtle.forward(duzina/9)
    turtle.left(ugao)

turtle.left(-120)
for ugao in (60, -120,60,0):
    turtle.forward(duzina/9)
    turtle.left(ugao)

turtle.left(60)
for ugao in (60, -120,60,0):
    turtle.forward(duzina/9)
    turtle.left(ugao)
```

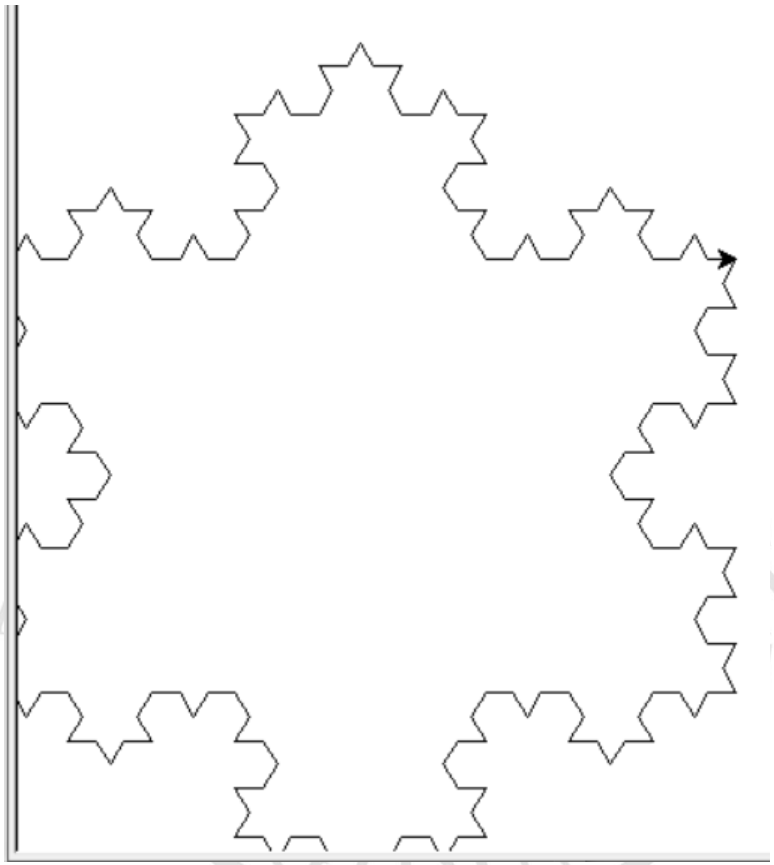


```
#isto samo preko petlje
import turtle
duzina=400

for ugao in (0,60, -120,60):
    turtle.left(ugao)
    for ugao in (60, -120,60,0):
        turtle.forward(duzina/9)
        turtle.left(ugao)
```

# Vraćamo se na Kohovu krivu. Kako dobiti pahulju?

```
import turtle
turtle.forward(-300)
def koh(duzina,dubina):
    if dubina > 0 :
        for ugao in (60, -120,60,0):
            koh(duzina/3, dubina-1)
            turtle.left(ugao)
    else:
        turtle.forward(duzina)

#Pozivanje funkcije koh
koh(100,3)

for i in range(3):
    turtle.right(120)
    koh(300,3)
```

Ovo smo imali malo pre bez 3 reda boldovana a sa koh(300,3)

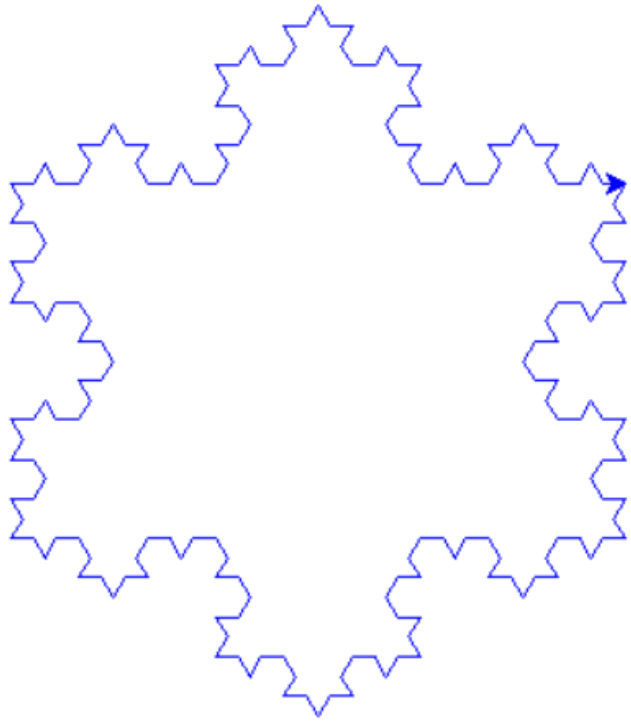


```
import turtle
turtle.forward(300)
def koh(duzina,dubina):
    if dubina > 0 :
        for ugao in (60, -120,60,0):
            koh(duzina/3, dubina-1)
            turtle.left(ugao)
    else:
        turtle.forward(duzina)

#Pozivanje funkcije koh
#koh(100,3)
for i in range(3):
    turtle.right(120)
    koh(300,3)
```

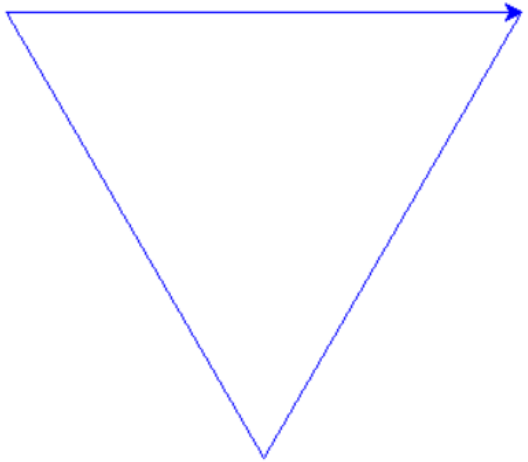
Ponovićemo primer sa prođlog časa:
Umesto poziva
koh (100,3)
stavljamo petlju

Kako da se ne vidi početna linija?

```
import turtle
turtle.color("white")
turtle.forward(300)
turtle.color("blue")
def koh(duzina,dubina):
    if dubina > 0 :
        for ugao in (60, -120,60,0):
            koh(duzina/3, dubina-1)
            turtle.left(ugao)
    else:
        turtle.forward(duzina)

for i in range(3):
    turtle.right(120)
    koh(300,3)
```



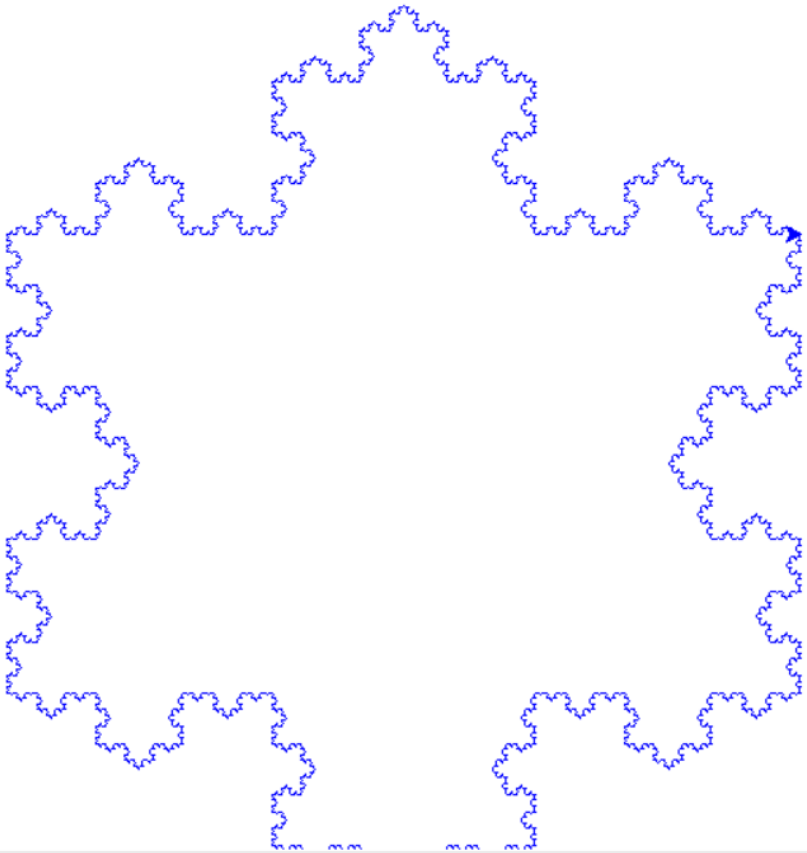
Šta se crta sa koh (300,0)?



Šta se crta sa :

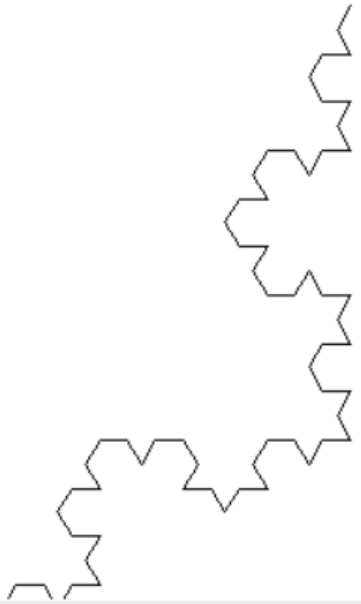
```
for i in range(3):
    turtle.right(120)
    koh(500,5)
```

?



Šta ce nacrtati za :

```
for i in range(1):
    turtle.right(120)
    koh(300,3)
```



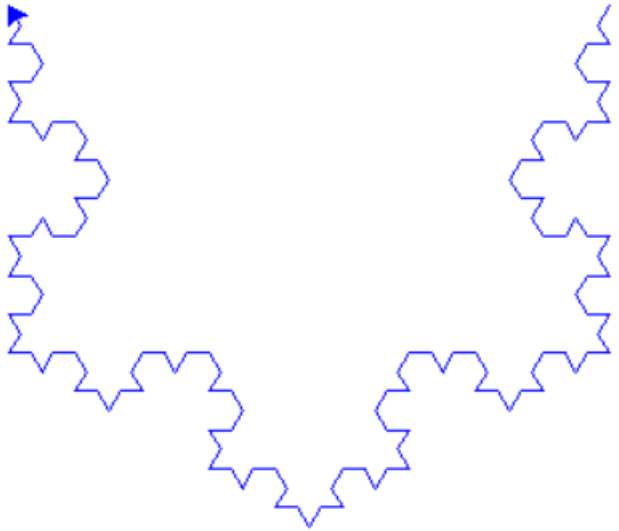
Šta ce nacrtati za :

For i in range(2)?

Šta ce nacrtati za :

For i in range(0)?

ništa



# 5. Rekurzivne i iterativne verzije algoritama

Rekurzivna definicija faktorijela n!=n· (n-1)! jednostavno se realizuje kao rekurzivna funkcija u jeziku Python.

Faktorijel se može definisati i iterativno, kao n!= n·(n-1)·(n-2)· ... ·2·1, odnosno, zbog komutacije operacije množenja, i kao n!= ?

1·2 · ... ·(n-2)·(n-1)·n.

Realizacija iterativnog algoritma za računanje funkcije faktorijel u jeziku Python je, npr.

```
def faktorijel(n):
    f = 1
    for i in range(1, n+1):
        f = f * i
    return f
print("Faktorijel je  ",faktorijel(5))
```

```
Faktorijel je   120
>>>
```

Izvršavanje funkcije je efikasnije, i ne zavisi više od ograničenja rekurzivne dubine.

>>> faktorijel(10)

3628800

>>> faktorijel(30)

265252859812191058636308480000000

>>>>

faktorijel(1115)

Ipak probati: faktorijel(3330)

```
    print('faktorijel je ',faktorijel(3330))
ValueError: Exceeds the limit (4300 digits) for in
ys.set_int_max_str_digits() to increase the limit
KeyboardInterrupt
```

**Iterativne verzije algoritama su efikasnije, jer nemaju potrebu za višestrukim čuvanjem vrednosti argumenata u sistemskom steku poziva, pošto se računanje obavlja u okviru jednog poziva funkcije.**

Rekurzivni algoritmi načelno se mogu pretvoriti u iterativne korišćenjem interne promenljive, koja predstavlja strukturu podataka tipa stek.

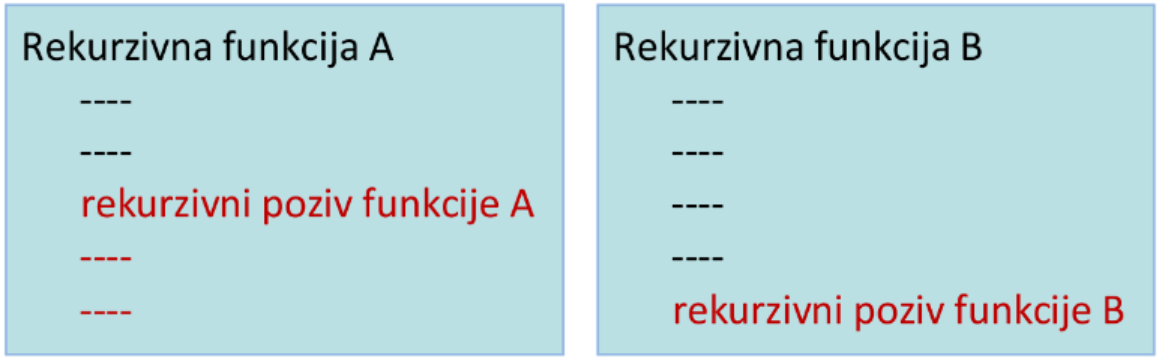
Interna struktura tipa stek koristi se za pamćenje redosleda izvršavanja naredbi, odnosno odgovarajućih vrednosti lokalnih promenljivih.

Algoritam se tada može realizovati pogodnom upotrebom petlji i strukture stek, umesto rekurzijom.

Funkcije su **rekurzivne na kraju (tail recursive**) kada se rekurzivno pozivanje vrši na kraju izvršavanja funkcije, nakon završetka svih ostalih operacija – **varijanta b.**

Funkcije koje **nisu rekurzivne na kraju**, zahtevaju čuvanje obimnijeg rekurzivnog konteksta u steku i nisu efikasne – **varijanta a**.

Na slici su prikazane  tipične strukture koda neefikasne i efikasne rekurzivne funkcije.




(a) neefikasna (b) efikasna

**Vidi se da funkcija A nakon povratka iz rekurzivnog poziva mora da izvrši još neke operacije pre završetka rada, pa je zbog toga neophodno čuvanje svih argumenata, međurezultata i povratnih informacija u sistemskom steku.**

Primer neefikasne rekurzivne funkcije je standardna realizacija funkcije

faktorijel:

```
# Funkcija koja nije rekurzivna na kraju
def faktorijel(n):
    if n == 0:
        return 1
    else:
        return n * faktorijel(n-1)
print(faktorijel(12))
```

Primer efikasne realizacije iste rekurzivne funkcije je funkcija faktorijel koja ima dodatni argument, u kome se rezultat formira u toku pozivanja :

```
# Funkcija rekurzivna na kraju
def faktorijel(n, rezultat):
    if n == 0:
        return rezultat
    else:
        return faktorijel(n - 1, n*rezultat)

print(faktorijel(7,1))
```

Funkcija se može koristiti kao faktorijel(n,1), ali se može definisati pomoćna funkcija, kako bi se pozivala samo s jednim argumentom.

# Kako možemo isprati formiranje rezultata:

```
# Funkcija rekurzivna na kraju
def faktorijel(n, rezultat):
    if n == 0:
        return rezultat
    else:
        print("n=",n,"  rezultat =",rezultat)
        return faktorijel(n - 1, n*rezultat)

print(faktorijel(7,1))
```

```
n= 7   rezultat = 1
n= 6   rezultat = 7
n= 5   rezultat = 42
n= 4   rezultat = 210
n= 3   rezultat = 840
n= 2   rezultat = 2520
n= 1   rezultat = 5040
5040
>>>
```

Koliko je:

print(faktorijel(7,0))?

0

print(faktorijel(7,2))?

10080