

PROGRAMIRANJE 1

Rekurzija

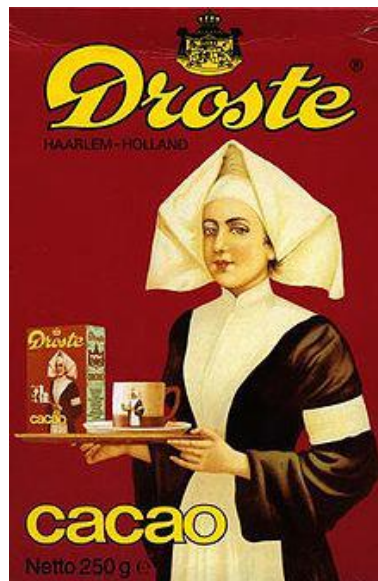
- Rekurzija je u matematici i računarstvu metoda definiranja funkcija u kojima se definirajuća funkcija primjenjuje unutar definicije.
- Na latinskom, re- = nazad + currere = izvršavati, događati se iznova.
- Pojednostavljeno rečeno rekurzija je algoritam koji poziva sam sebe, pri čemu do rješenja problema dolazimo rješavanjem podproblema.
- Bolje rečeno pojednostavljujemo veliki problem kroz jedan ili više manjih pod problema i njihovo rekurzivno rješavanje dok ne dosegemo osnovno rješenje koje možemo riješiti direktno bez rekurzije.

- u skupu prirodnih brojeva, možemo uočiti rekurziju:
 - prvi član je 1, a svaki sljedeći dobijemo tako da prethodni uvećamo za 1
- Jezična rekurzija bitno je obilježje ljudskoga jezika koje omogućuje beskonačnu produkciju novih izričaja. Na primjer,

Dječak nije mogao spavati pa mu je majka ispričala priču o žabici,
koja nije mogla spavati pa joj je majka ispričala priču o medi,
koji nije mogao spavati pa mu je majka ispričala priču o maci
...
i maca je zaspala
i medo je zaspao
i žabica je zaspala
i dječak je zaspao.

Vizualni oblici rekurzije

- Droste efekt
 - Naziv Droste dobio je po njemačkoj vrsti tople čokolade, na čijem je pakovanju bila slika sestre koja na pladnju poslužuje šalicu i pakovanje iste vrste tople čokolade. Na tom drugom pakovanju tople čokolade je opet jednaka slika sestre i to se ponavlja unedogled



- Gotovo paralelna ogledala



- Rekurzivna slika zaslona



Primjer jednostavne rekurzije

- Zadana je cjelobrojna funkcija f s dva cjelobrojna parametra x i y .
- Koja će biti vrijednost cjelobrojne varijable t nakon izvršavanja naredbe $t = f(f(2, 3), 4)$

```
funkcija f (x, y)  
vrati x * (y + 1)
```

$$\begin{aligned}t &= f(f(2, 3), 4) \rightarrow x=f(2,3), y=4 \\&= f(2,3) * (4 + 1) \\&= f(2,3) * 5 \\&= 8 * 5 \\&= 40\end{aligned}$$

$$f(2,3) = 2 * (3 + 1) = 2 * 4 = 8$$

40

Primjer složenije rekurzije

- Zadana je cjelobrojna funkcija fpr s dva cjelobrojna parametra x i y.
- Koju će vrijednost imati funkcija fpr nakon izvršenja ako se poziva s cjelobrojnim vrijednostima $m = 42$ i $n = 48$?

```
funkcija fpr (m, n)
    ako je m < n onda
        vrati fpr (n, m)
    inace
        ako je n=0 onda
            vrati m
        inace
            vrati fpr (n, m mod n)
```


Korak broj	Aktivnost	Korak broj	Aktivnost
1	$\text{fpr}(42,48)=?$ $m=42, n=48$ $m < n$ $42 < 48 \text{ T}$ $\text{fpr}(42,48)=\text{fpr}(48,42)$	2	$\text{fpr}(48,42)=?$ $m=48, n=42$ $m < n$ $48 < 42 \text{ F}$ $n=0$ $42=0 \text{ F}$ $\text{fpr}(48,42)=\text{fpr}(42, 48 \bmod 42)$ $=\text{fpr}(42,6)$
3	$\text{fpr}(42,6)=?$ $m=42, n=6$ $m < n$ $42 < 6 \text{ F}$ $n=0$ $6=0 \text{ F}$ $\text{fpr}(42,6)=\text{fpr}(6, 42 \bmod 6)$ $=\text{fpr}(6,0)$	4	$\text{fpr}(6,0)=?$ $m=6, n=0$ $m < n$ $6 < 0 \text{ F}$ $n=0$ $0=0 \text{ T}$ $\text{fpr}(6,0)=6$

6

- Karakteristike rekurzije
 - Osnovni slučajevi: uvijek moraju postojati osnovni slučajevi koji se rješavaju bez rekurzije
 - Korak rekurzije: Za slučajeve koji se rješavaju rekurzivno, svaki sljedeći rekurzivni poziv mora se približiti osnovnim slučajevima.
- Na primjer, sljedeće je rekurzivna definicija predaka osobe:
 - Nečiji roditelji su njegovi pretci (osnovni slučaj);
 - Roditelji bilo kojeg pretka su također pretci osobe koju promatramo (korak rekurzije).

```
>>> def rfunc(n):  
    print(n)  
    if n > 0:  
        rfunc(n - 1)  
  
>>> rfunc(4)  
>>> ??? 4, 3, 2, 1, 0  
  
>>> rfunc(0)  
>>> ??? 0  
  
>>> rfunc(100)  
>>> ??? 100, 99, ..., 1, 0
```

```
>>> def rfunc(n):  
    if n == 1:  
        return 1  
    else:  
        return n + rfunc(n - 1)  
  
>>> print(rfunc(1))  
>>> ??? 1  
  
>>> print(rfunc(3))  
>>> ??? 6  
  
>>> print(rfunc(100))  
>>> ??? 5050
```

- Općenito za rekurziju vrijedi:
 - Postupak poziva funkcija je vremenski „skuplji“ nego izvođenje iterativnih postupaka u petlji,
 - Rekurzija zahtijeva više vremena (instrukcije za pozivanje funkcije i povratak su sporije)

- Funkcioniranje rekurzije
 - Djeluje jednostavno:
 - Kad uđemo u algoritam, prvo pitamo da li smo dosegli osnovni slučaj ili rješenje.
 - Ako jesmo, vraćamo se iz funkcije.
 - Ako nismo, smanjujemo osnovni ulaz i ponovo pozivamo algoritam.
 - Ponavljamo to dok se ne počnemo vraćati sa rezultatima tih pojednostavljenih slučajeva.
 - U praksi je to složenije:
 - Prvo moramo uvijek odrediti rekurzivnu funkciju i osnovni slučaj, a to baš ne mora kod složenijih algoritama biti očigledno.

- Većina ljudi prolazi tri faze u procesu učenja rekurzije:
 1. Prvo, oni je mrze, jer je ne mogu razumjeti.
 2. Zatim, je vole jer su razumjeli ovaj misteriozni proces.
 3. I na kraju ponovo je mrze jer misle da je neučinkovita.
- Definicije rekurzije:
 1. Rekurzija
Vidi Rekurzija
 2. Rekurzija je nepropisno parkiran pauk.
 3. Rekurzija je kada pozivaš samog sebe da bi odradio nešto što si i sam mogao napraviti.
 4. Rekurzija je kada se račun za kreditnu karticu plaća istom kreditnom karticom.

Primjer

- izračunavanje sume prvih N prirodnih brojeva:

$$\text{Suma}_N = 1 + 2 + 3 + \dots + N$$

$$\text{Suma}_{N-1} = 1 + 2 + 3 + \dots + (N-1)$$

- $\text{Suma}_N = \text{Suma}_{N-1} + N$

```
def suma(n):  
    if n==1:  
        return 1  
    else:  
        return suma(n-1)+n
```

- Treba izračunati produkt prvih n prirodnih brojeva koristeći se rekurzivnom funkcijom

```
def produkt(n) :  
    if n==1:  
        return 1  
    else:  
        return produkt(n-1)*n
```


- Što je rezultat izvršavanja sljedeće rekurzivne funkcije za bilo koji prirodan broj n ?

```
def rfunc(n):  
    if n == 0:  
        return 0  
    else:  
        return rfunc(n - 1) + 2
```

2^n

- Za sljedeće rekurzivne funkcije navedite koja od karakteristika rekurzije nije ispunjena:

```
def rfunc1(n):  
    return n + rfunc1(n - 1)
```

Nema osnovnog slučaja

```
def rfunc2(n):  
    if n == 0:  
        return 1  
    return n + rfunc2(n + 1)
```

Korak rekurzije se udaljava od osnovnog slučaja

- Napišite rekurzivnu i nerekurzivnu funkciju koja provjerava da li je uneseni broj paran ili nije.

```
def paran(n):  
    if n % 2 == 0:  
        return "Paran"  
    else:  
        return "Neparan"
```

```
m=int(input())  
print(paran(m))
```

```
def paran(n):  
    if n == 2:  
        return "Paran"  
    elif n==1:  
        return "Neparan"  
    else:  
        return paran(n - 2)
```

```
m=int(input())  
print(paran(m))
```

- Napišite rekurzivnu i nerekurzivnu funkciju koja ispisuje sljedeće:

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

```
def zvjezdice(n):
    for i in range(n,1,-2):
        s=i*""
        print(s.center(n))
```

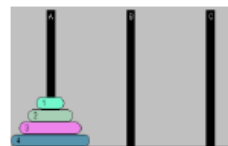
```
m=12
zvjezdice(m)
```

```
def zvjezdice(i,n):
    s=i*""
    print(s.center(n))
    if i == 0:
        return i*""
    else:
        return zvjezdice(i-2,n)
```

```
m=12
zvjezdice(m,m)
```

Poznatiji rekurzivni algoritmi

- Faktorijel $n! = 1 * 2 * 3 * \dots * n$
- Fibonaccijevi brojevi $F(n) = F(n-1) + F(n-2)$
- Euklidov postupak za pronalaženje najveće zajedničke mjere
- 8 kraljica:
 - U ovom problemu potrebno je napisati funkciju koja će pronaći položaj 8 kraljica na šahovskoj ploči tako da se one međusobno ne napadaju.
- Hanojski tornjevi:
 - Postoje štapovi A (source-izvor), C (destination-odredište), B (temporary-pomoćni). Na prvom štapu (A) (slika) ima n diskova različite veličine postavljenih tako da veći nikad ne dolazi iznad manjeg. Uz minimalni broj operacija preselite sve diskove na C, jedan po jedan. Disk se smije postaviti ili na prazan štap ili tako da je manji disk na većem.



Faktorijel

$$n! = n * (n-1) * (n-2) * \dots * 3 * 2 * 1$$

ako grupiramo članove na ovaj način:

$$n! = n * [(n-1) * (n-2) * \dots * 3 * 2 * 1]$$

tada se svaki korak sastoji od dva dijela:

$$n! = n * (n-1)!$$

1. jednog nezavisnog (pomnožiti sa n) i
2. jednog koji izgleda baš kao i početni zadatak $(n-1)!$ – korak

Osnovni slučaj je $1!=1$

Kada se prestaje sa rekurzivnim pozivima? Dok se ne dođe do osnovnog slučaja

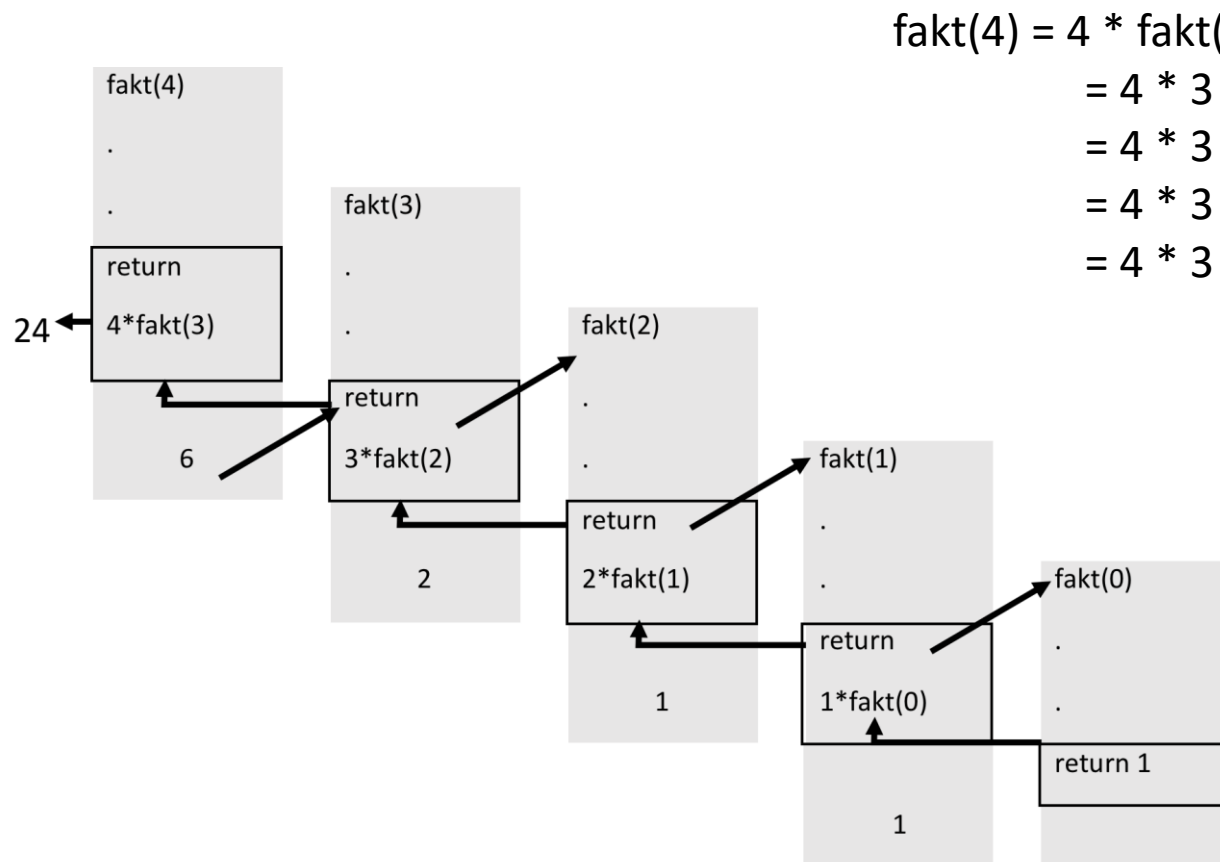
Npr.

$$3! = 3 * 2! \dots \text{imamo rekurzivni poziv}$$

$$2! = 2 * 1! \dots \text{imamo rekurzivni poziv}$$

$$1! = 1 \dots \text{Osnovni slučaj!}$$


Korak broj	Aktivnost	Korak broj	Aktivnost
1	fakt(4)=? n=0 4=0 F fakt(4)=4*fakt(4-1)=4*fakt(3)	2	fakt(3)=? n=0 3=0 F fakt(3)=3*fakt(3-1)=3*fakt(2)
3	fakt(2)=? n=0 2=0 F fakt(2)=2*fakt(2-1)=2*fakt(1)	4	fakt(1)=? n=0 1=0 F fakt(1)=1*fakt(1-1)=1*fakt(0)
5	fakt(0)=? n=0 0=0 T fakt(0)=1	6	



$$\begin{aligned}\text{fakt}(4) &= 4 * \text{fakt}(3) \\ &= 4 * 3 * \text{fakt}(2) \\ &= 4 * 3 * 2 * \text{fakt}(1) \\ &= 4 * 3 * 2 * 1 * \text{fakt}(0) \\ &= 4 * 3 * 2 * 1 * 1 = 24\end{aligned}$$

- Napravi program koji pomoću funkcije izračunava n faktoriijela.

```
def fakt(n):  
    if n==0:  
        return 1  
    else:  
        return n*fakt(n-1)  
  
n=int(input("Unesite broj:"))  
print (n,"! =",fakt(n))
```



Osnovni slučaj

Fibonaccijevi brojevi

- $\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$,
- Osnovni slučajevi: $\text{fib}(0)=0$, $\text{fib}(1)=1$

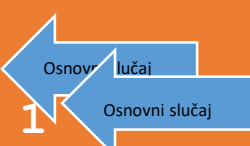
Korak broj	Aktivnost	Korak broj	Aktivnost
1	$f(13)=?$ $n=0$ $13=0$ F $n=1$ $13=1$ F $f(13)=f(13-1)+f(13-2)=f(12)+f(11)$	2	$f(12)=?$ $n=0$ $12=0$ F $n=1$ $12=1$ F $f(12)=f(12-1)+f(12-2)=f(11)+f(10)$
3	$f(11)=?$ $n=0$ $11=0$ F $n=1$ $11=1$ F $f(11)=f(11-1)+f(11-2)=f(10)+f(9)$	4	$f(10)=?$ $n=0$ $10=0$ F $n=1$ $10=1$ F $f(10)=f(10-1)+f(10-2)=f(9)+f(8)$
5	$f(9)=?$ $n=0$ $9=0$ F $n=1$ $9=1$ F $f(9)=f(9-1)+f(9-2)=f(8)+f(7)$	6	$f(8)=?$ $n=0$ $8=0$ F $n=1$ $8=1$ F $f(8)=f(8-1)+f(8-2)=f(7)+f(6)$

7	f(8)=? n=0 8=0 F n=1 8=1 F $f(8)=f(8-1)+f(8-2)=f(7)+f(6)$	8	f(7)=? n=0 7=0 F n=1 7=1 F $f(7)=f(7-1)+f(7-2)=f(6)+f(5)$
9	f(6)=? n=0 6=0 F n=1 6=1 F $f(6)=f(6-1)+f(6-2)=f(5)+f(4)$	10	f(5)=? n=0 5=0 F n=1 5=1 F $f(5)=f(5-1)+f(5-2)=f(4)+f(3)$
11	f(4)=? n=0 4=0 F n=1 4=1 F $f(4)=f(4-1)+f(4-2)=f(3)+f(2)$	12	f(3)=? n=0 3=0 F n=1 3=1 F $f(3)=f(3-1)+f(3-2)=f(2)+f(1)$

13	f(2)=? n=0 2=0 F n=1 2=1 F f(2)=f(2-1)+f(2-2)=f(1)+f(0)	14	f(1)=? n=0 1=0 F n=1 1=1 T f(1)=1
15	f(0)=? n=0 0=0 t f(0)=0		

```
def fib (n):
    if (n==0): return 0
    elif (n==1): return 1
    else: return fib(n-1)+fib(n-2)

n=int(input("Unesite broj:"))
print ("fib(",n,")=",fib (n))
```



Euklidov postupak

Algoritam rješenja:

ako je $b = 0$ onda

$$\text{nzm} = a$$

inace

$\text{nzm} = \text{najveca zajednicka mjera od } b \text{ i ostatka dijeljenja } a \text{ sa } b$

Korak broj	Aktivnost	Korak broj	Aktivnost
1	$\text{nzm}(22,8) = ?$ $a=22, b=8$ $b=0$ $8=0 \text{ F}$ $\text{nzm}(22,8)=\text{nzm}(8, 22 \bmod 8)=\text{nzm}(8,6)$	2	$\text{nzm}(8,6) = ?$ $a=8, b=6$ $b=0$ $6=0 \text{ F}$ $\text{nzm}(8,6)=\text{nzm}(6, 8 \bmod 6)=\text{nzm}(6,2)$
3	$\text{nzm}(6,2) = ?$ $a=6, b=2$ $b=0$ $2=0 \text{ F}$ $\text{nzm}(6,2)=\text{nzm}(2, 6 \bmod 2)=\text{nzm}(2,0)$	4	$\text{nzm}(2,0) = ?$ $a=2, b=0$ $b=0$ $0=0 \text{ T}$ $\text{nzm}(2,0)=2$

```
def nzm(x,y):  
    if y=0:  
        return x  
    else:  
        return nzm(y, x%y)  
  
a,b =int(input("Unesite dva broja:"))  
print (NZM(";a;", ";b;"="";nzm(a,b))
```


8 kraljica

- **Napomene:**

- šahovska ploča je veličine $8 * 8$
- postoje 92 različita rješenja
- u svakom retku i svakom stupcu mora se nalaziti samo jedna kraljica

- **Algoritam rješenja:**

- promatramo stupce na šahovskoj ploči od prvog prema zadnjem
- u svaki postavljamo jednu kraljicu
- promatramo ploču u situaciji kada je već postavljeno i kraljica (u i različitih stupaca) koje se međusobno ne napadaju
- želimo postaviti $i + 1$ kraljicu tako da ona ne napada niti jednu od već postavljenih kraljica i da se ostale kraljice mogu postaviti uz uvjet nenapadanja

Hanojski tornjevi

- **Napomene:**

- Štapovi S (*source*, izvor), D (*destination*, odredište), T (*temp*, pomoćni)
- Na prvom štapu (S) ima n diskova različite veličine postavljenih tako da veći nikad ne dolazi iznad manjeg.
- Preseliti sve diskove na D, **jedan po jedan**, uvijek postavljajući manji na veći

- **Algoritam rješenja:**

- Ignorirati donji (najveći) disk i riješiti problem za $n-1$ disk, ali sa štapa S na štap T koristeći D kao pomoćni.
- Sada se najveći disk nalazi na S, a ostalih $n-1$ na T.
- Preseliti najveći disk sa S na D.
- Preseliti $n-1$ disk sa T na D koristeći S kao pomoćni (problem je već riješen za $n-1$ disk).

