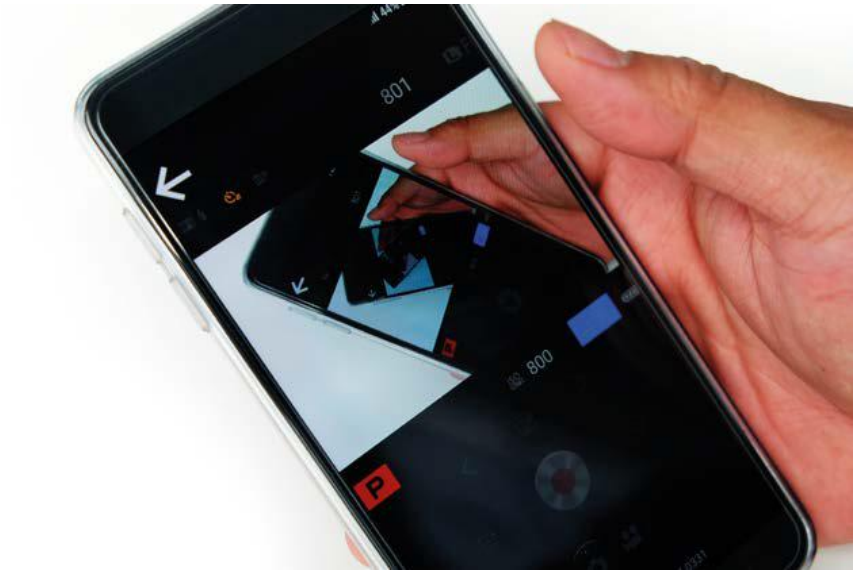
3. RAČUNALNO RAZMIŠLJANJE

3.1 Rekurzije

ŠTO JE SAMOSLIČNOST?

ŠTO JE SAMOSLIČNOST?

- **Samosličnost** je svojstvo objekta da sadrži dijelove u manjem mjerilu koji su slični cjelini.



REKURZIJA – ŠTO JE TO?

NABROJI PRIMJERE

REKURZIJA – ŠTO JE TO?

„Jeste li ikada vidjeli sliku koja sadrži manju kopiju same sebe, pa opet i opet?“

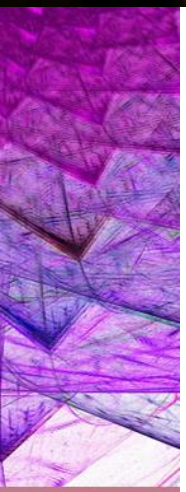
RUSKA BABUŠKA

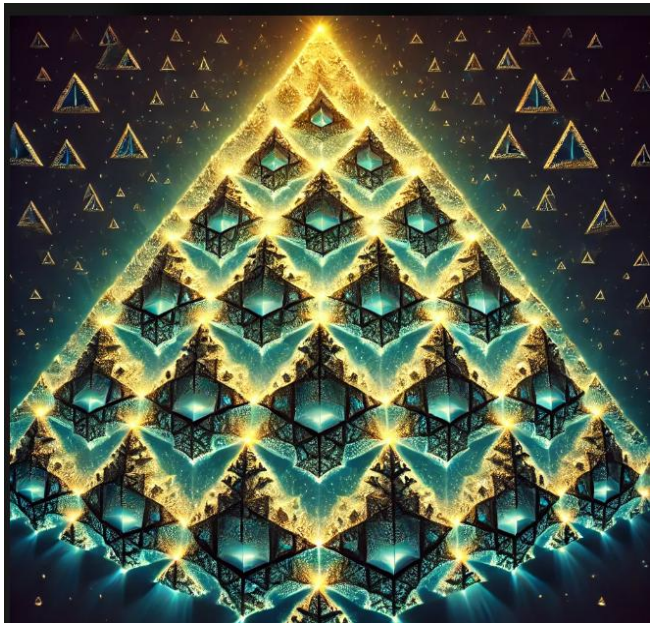
Ruska babuška (Matryoshka)

Ruske babuške su drvene lutke koje se otvaraju i otkrivaju manju verziju sebe unutar. Svaka manja lutka sadrži još manju lutku, sve do najmanje unutarnje lutke koja se više ne može otvoriti.

DVA OGLEDALO NASUPROT

BESKONAČNA REFLEKSIJA





KOJE SU FAZE RJEŠAVANJE PROBLEMA GEORGE POLYA?

RJEŠAVANJE PROBLEMA
GEORGE POLYA

1. Razumijevanje problema
2. Stvaranje plana rješavanja problema
3. Izvršavanje osmišljenog plana
4. Osvrt na rješenje i metodu rješavanja

ZADATAK

Riješi problem.

- Nora je voditeljica plesne skupine i nakon održane audicije mora izabrati *n* plesača/ plesačica koji će sudjelovati na natjecanju.
- Napiši algoritam za rješavanje tog problema.

1. RAZUMIJEVANJE PROBLEMA

- Koja si pitanja postavljamo u 1. fazi?

1. RAZUMIJEVANJE PROBLEMA

- Među plesačima jedne skupine Nora mora izabrati *n* plesača/plesačica.
- Naravno, broj *n* manji je od ukupnog broja plesača/plesačica koji/koje su sudjelovali/sudjelovale na audiciji.

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

- Što radimo kada stvaramo plan?

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

- Na koja dva načina možemo riješiti ovaj problem?

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

- Što je iteracija ili petlja?

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

- Želi li Nora imati skupinu plesača, mora birati jednog po jednog plesača.
- Ponavljanje u algoritmu može se ostvariti dvojako. Prvi je način uporabom **petlje (iteracijom)** u kojoj će se *n* puta ponoviti naredba *Odaberi plesača/plesačicu.*
- **Iteracija (petlja)** je algoritamski postupak ponavljanja naredbi određeni, konačni broj puta.

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

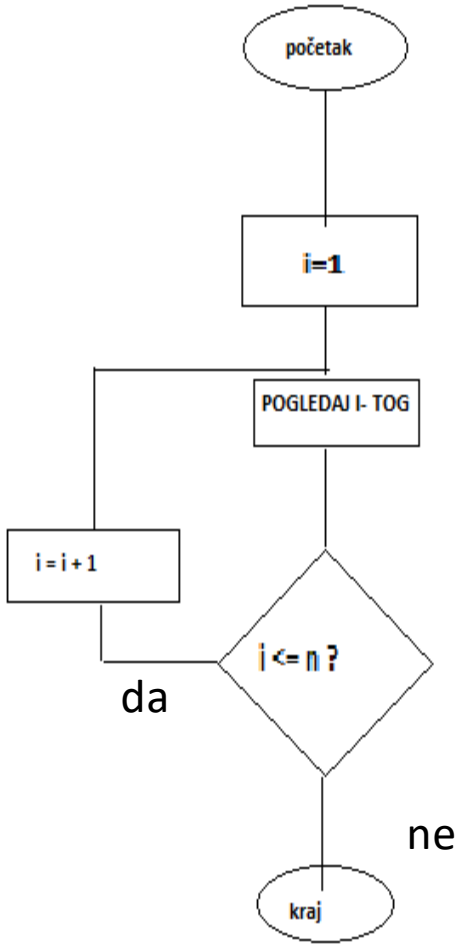
Kako glasi algoritam i izgleda dijagram toka ako rješavamo zadatak upotrebom petlje?

2. STVARANJE PLANA RJEŠAVANJA PROBLEMA

Algoritam:

Za i = 1 do n ponovi

Odaberi plesača/plesačicu sljedeći i



REKURZIVNI ALGORITAM

Što je Rekurzivni algoritam ili rekurzija?

Kako izgleda algoritam za ovaj zadatak?

REKURZIVNI ALGORITAM

Rekurzivni algoritam ili rekurzija je algoritamski postupak u kojem rekurzivna funkcija poziva samu sebe.

3. IZVRŠAVANJE OSMIŠLJENOG PLANA (uz rekurziju)

Algoritam:

Ples (n)

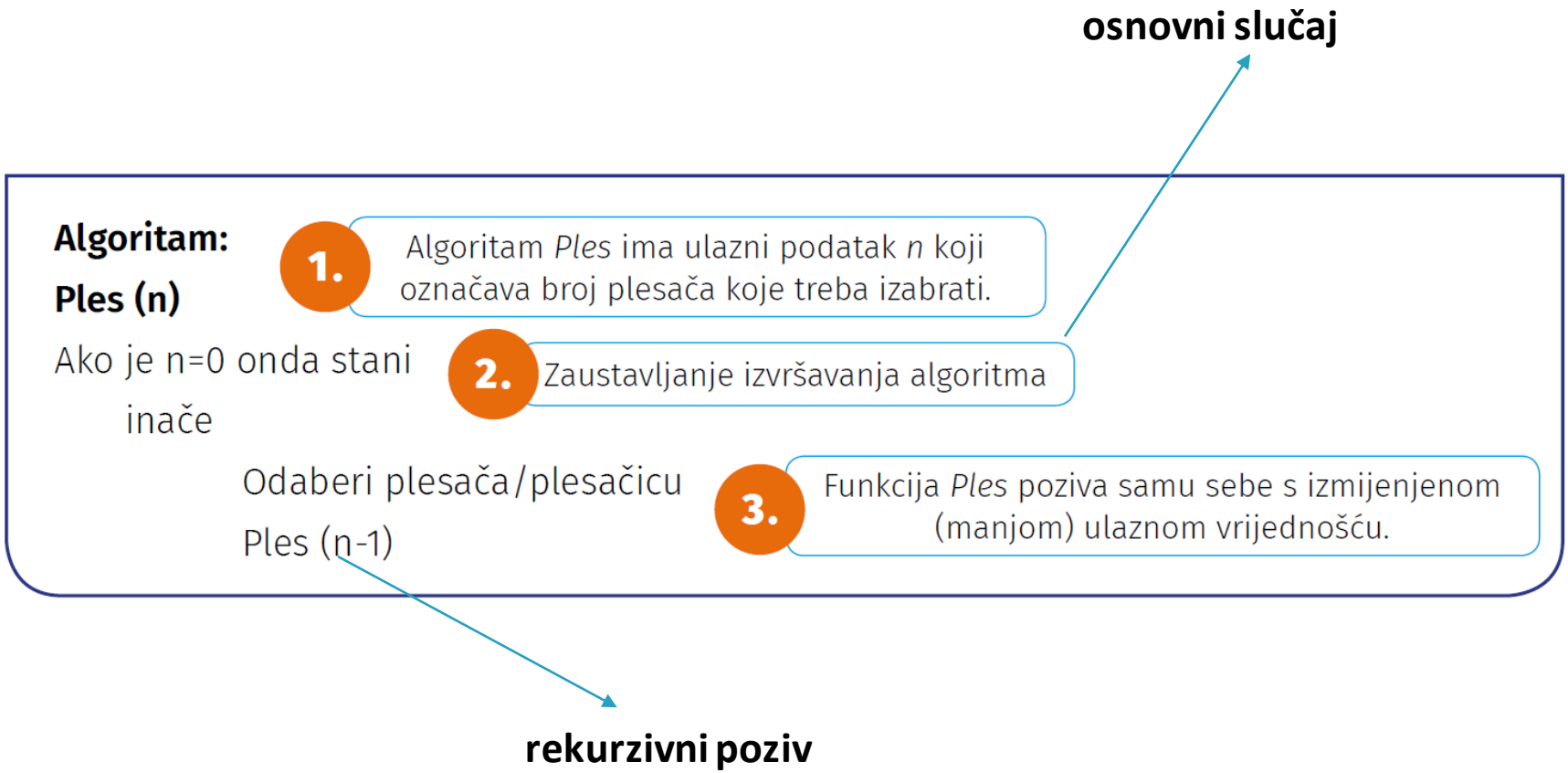
Ako je n=0 onda stani

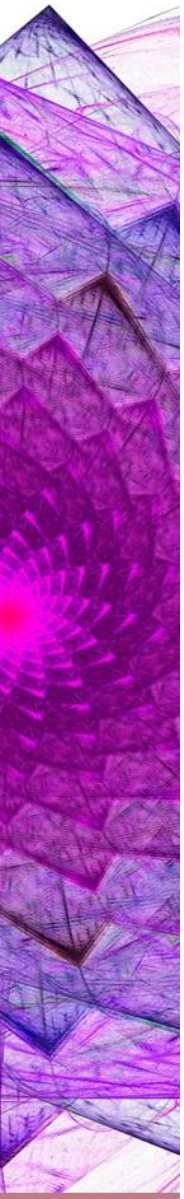
    inače

        Odaberi plesača/plesačicu

        Ples (n-1)

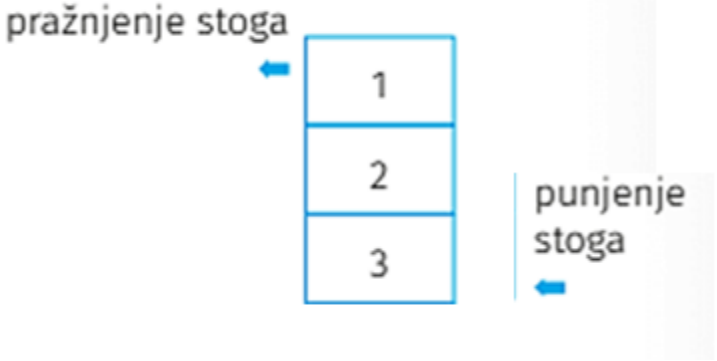
3. IZVRŠAVANJE OSMIŠLJENOG PLANA (uz rekurziju)





Što je stog ?
Što je LIFO način rada?

Stog je struktura podataka, smještaj u memoriji, u koju se pohranjuju varijable rekurzivne funkcije koja funkcionira na LIFO način rada.



1. plesač / 1. plesačica
2. plesač / 2. plesačica
3. plesač / 3 plesačica

LIFO – LAST IN FIRST OUT, zadnji ulazi, a prvi izlazi

- **Stog** je struktura podataka u koju se pohranjuju varijable rekurzivne funkcije koja funkcionira na LIFO način rada.
- **LIFO** (Last In First Out) način rada znači da podataka koji je zadnji ušao prvi izlazi van.

- U trenutku kad unutar rekurzivne funkcije dođemo do rekurzivnog poziva s izmijenjenom vrijednošću argumenta, privremeno se prekida izvršavanje tekuće funkcije i vrijednost varijable sprema se na stog.
- Prvi element stoga postavlja se na dno, zatim se na njega postavlja drugi element itd.
- Postupak se ponavlja sve dok se ne dođe do osnovnog slučaja čime se završavaju rekurzivni pozivi. Tim se postupkom povećava (raste) sadržaj stoga.

4. OSVRT NA RJEŠENJE I METODU RJEŠAVANJA

- Što bi se dogodilo da nema osnovnog slučaja?

4. OSVRT NA RJEŠENJE I METODU RJEŠAVANJA

- Algoritam kojim je riješen problem naziva se **rekurzivni algoritam** ili **rekurzija**.
- Osnovna je ideja rekurzije riješiti algoritam funkcijom koja poziva samu sebe. U ovom slučaju to je funkcija *Ples*.
- Unutar algoritma mora postojati naredba kojom se prekida izvršavanje algoritma.
  U protivnom, budući da funkcija poziva samu sebe, algoritam bi se izvodio beskonačno.
  Ovaj algoritam završava kada ulazna vrijednost funkcije *Ples* postane *0*, odnosno kad su izabrani svi plesači.

ZADATAK

Riješi problem.

- Za učitani broj *n* napiši rekurzivni algoritam za izračunavanje umnoška prirodnih brojeva do *n*.

3. IZVRŠAVANJE OSMIŠLJENOG PLANA

Algoritam:

umnožak (n)

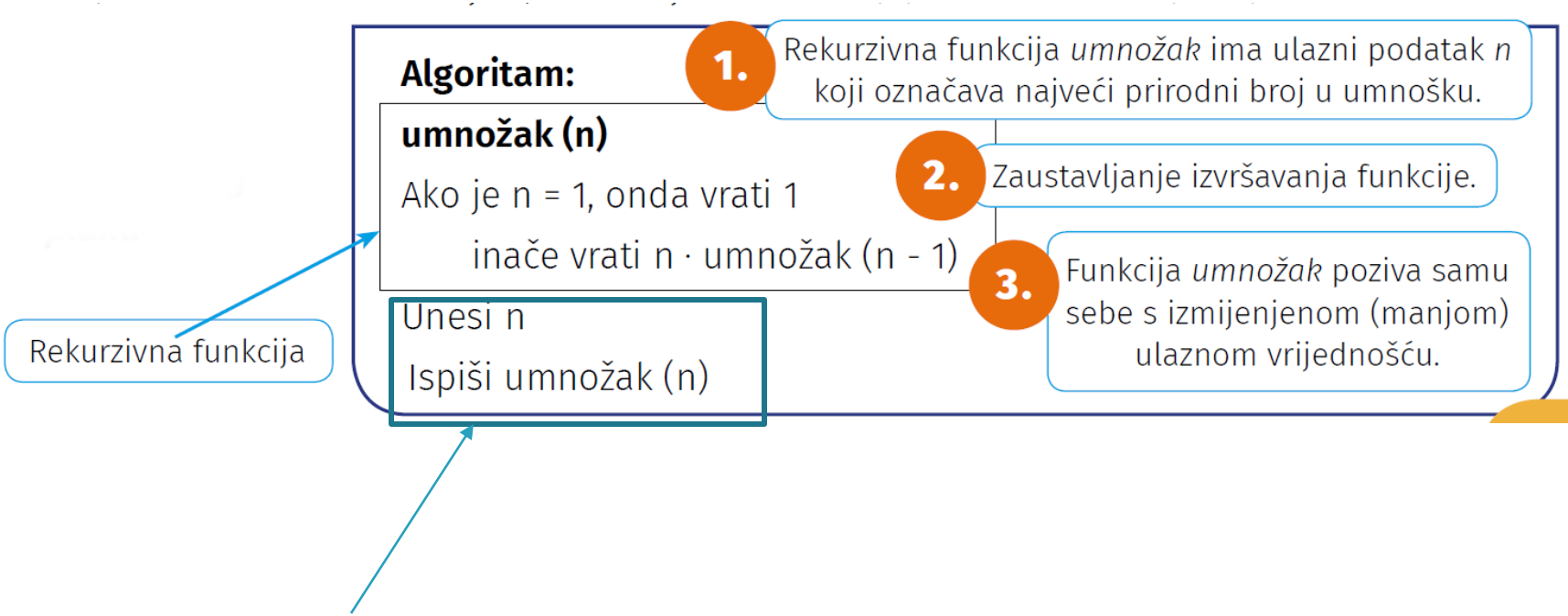
Ako je n = 1, onda vrati 1

inače vrati n · umnožak (n - 1)

Unesi n

Ispiši umnožak (n)

3. IZVRŠAVANJE OSMIŠLJENOG PLANA



Glavni program u kojem se poziva rekurzivna funkcija

ZAŠTO RJEŠAVAMO ZADATKE REKURZIJOM?
KOJI ALGORITMI TROŠE VIŠE MEMORIJE?
KOJI SE SPORIJE IZVRŠAVAJU?

ZAŠTO REKURZIJA

- Primjenom rekurzija dobijemo kraće i jednostavnije algoritme, a neke probleme bilo bi vrlo teško riješiti na neki drugi način.
- Pri izvršavanju programa rekurzivni programi su zahtjevniji i troše više memorije od iterativnih programa.
  Sporiji su u izvršavanju zato što moraju stalno spremati podatke na stog i čitati podatke s nj
- Rekurzije imaju veliku primjenu u programiranju jer se uz relativno jednostavne algoritme mogu stvarati složene strukture.