

# Atribut klase, klasna i statička metoda

Šta pripada svakom objektu, a šta celoj klasi

**Danas pravimo: fabriku pahuljica koja zna koliko je pahuljica napravila**



Jedan broj za sve pahuljice: atribut klase

# Tri pitanja za zagrevanje

## 1. prošla tema: self

```
class Brojac:  
    def __init__(self):  
        self.n = 0  
  
    def dodaj(self):  
        self.n += 1
```

```
b = Brojac()  
b.dodaj()  
b.dodaj()  
print(b.n)
```

2

## 2. tema 01: b = a

```
a = [1, 2]  
b = a  
b.append(3)  
print(a, a is b)
```

[1, 2, 3] True

## 3. mešano: lista objekata

```
class Zvezda:  
    pass  
  
nebo = []  
for i in range(3):  
    z = Zvezda()  
    z.sjaj = i * 2  
    nebo.append(z)  
print(nebo[2].sjaj)
```

4

# Šest novih reči za danas

**atribut instance**

pripada jednom objektu: `self.x`, `self.y`

**atribut klase**

piše se u telu klase i jedan je za sve objekte

**zaklanjanje**

objekat dobije svoj atribut istog imena i više ne vidi klasin

**klasna metoda**

@classmethod: dobija klasu (`cls`), a ne objekat

**statička metoda**

@staticmethod: ne dobija ni objekat ni klasu

**kopija**

nov objekat; plitka (`copy`) ili duboka (`deepcopy`)

# Objekat čuva svoje, a klasa ono što je zajedničko

## klasa Pahuljica

```
boja = "bela"  
ukupno = 2
```

atributi klase: jedan primerak, vide ga svi objekti



### objekat p1

```
x = 10, y = 0
```

atributi instance

p1.boja: nema u p1,  
pa Python gleda u  
klasu



### objekat p2

```
x = 50, y = 20
```

atributi instance

*Python atribut prvo traži u objektu, a ako ga tamo nema, u klasi.*

# Atribut klase

Jedna vrednost za sve objekte

# Atribut klase pišemo u telu klase, bez self

```
class Pahuljica:  
    boja = "bela"          # atribut klase  
  
    def __init__(self, x, y):  
        self.x = x         # atributi instance  
        self.y = y
```

```
p1 = Pahuljica(10, 0)  
p2 = Pahuljica(50, 20)  
print(p1.x, p2.x)  
print(p1.boja, p2.boja)  
print(Pahuljica.boja)
```

ISPIS

```
10 50  
bela bela  
bela
```

- boja: jedan primerak, u klasi.
- x i y: svaka pahuljica ima svoje.
- Atribut klase čitamo i preko objekta (p1.boja) i preko klase (Pahuljica.boja).

# Dodela preko objekta pravi novi atribut

```
p1 = Pahuljica(10, 0)
p2 = Pahuljica(50, 20)
Pahuljica.boja = "plava"      # menjamo klasu
print(p1.boja, p2.boja)
p1.boja = "zlatna"            # novi atribut u p1
print(p1.boja, p2.boja)
print(vars(p1))
print(vars(p2))
```

*vars(objekat) pokazuje samo sopstvene attribute objekta.*

ISPIS

```
plava plava
zlatna plava
{'x': 10, 'y': 0, 'boja': 'zlatna'}
{'x': 50, 'y': 20}
```

**Pahuljica.boja = ...**

menja klasu: vide je svi koji nemaju svoju

**p1.boja = ...**

pravi atribut u p1; klasin je zaklonjen

# Šta ispisuje ovaj program?

```
class Lik:  
    zivoti = 3  
  
a = Lik()  
b = Lik()  
a.zivoti = 1  
Lik.zivoti = 5  
print(a.zivoti, b.zivoti)
```

Pažljivo: atribut menjamo jednom preko objekta, a jednom preko klase.

**A**

5 5

**B**

1 3

**C**

1 5

tačno

**D**

3 5

# Brojač objekata je atribut klase

```
class Pahuljica:
    ukupno = 0 # koliko je napravljeno

    def __init__(self, x, y):
        self.x = x
        self.y = y
        Pahuljica.ukupno += 1

print(Pahuljica.ukupno)
for i in range(5):
    Pahuljica(i * 10, 0)
print(Pahuljica.ukupno)
```

ISPIS

0  
5

- Pišemo `Pahuljica.ukupno`, a ne `self.ukupno`.
- Svaki poziv `Pahuljica(...)` pokreće `__init__`, pa brojač raste.

# Čudovište: zajednička lista

```
class Igrac:  
    torba = []      # u klasi!  
  
    def __init__(self, ime):  
        self.ime = ime  
  
    def uzmi(self, stvar):  
        self.torba.append(stvar)  
  
mia = Igrac("Mia")  
ana = Igrac("Ana")  
mia.uzmi("šal")  
print(ana.torba)
```

## Kako izgleda

Mia uzme šal, a `print(ana.torba)` ispiše ['šal']. Nema nikakve poruke o grešci.

## Zašto se pojavljuje

Lista u telu klase postoji samo jednom. `self.torba.append(...)` ne pravi novu listu, nego menja zajedničku.

## Kako se pobeđuje

Listu pravimo u `__init__`: `self.torba = []`. Tako svaki igrač dobije svoju torbu.

# Klasna i statička metoda

Metode kojima ne treba jedan određeni objekat

# Metoda može biti obična, klasna ili statička

```
class Primer:  
    def obican(self):          # self je objekat  
        ...  
  
    @classmethod  
    def klasni(cls):          # cls je klasa  
        ...  
  
    @staticmethod  
    def staticki(x):          # ništa automatski  
        ...
```

*Red sa @ iznad def je dekorator: oznaka koja menja ponašanje metode.*

## obična metoda

dobija objekat (self)

`p.crtaj(ekran)`

## klasna metoda

dobija klasu (cls)

`Pahuljica.nasumicna(640, 360)`

## statička metoda

ne dobija ništa automatski

`Boja.je_ispravna(30, 40, 90)`

# Klasna metoda dobija klasu kao cls

```
class Pahuljica:  
    ukupno = 0  
  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
        Pahuljica.ukupno += 1
```

```
@classmethod  
def izvestaj(cls):  
    return f"Napravljeno: {cls.ukupno}"
```

```
Pahuljica(10, 0)  
Pahuljica(50, 20)  
print(Pahuljica.izvestaj())
```

ISPIS

Napravljeno: 2

- Pozivamo je preko klase: Pahuljica.izvestaj().
- cls je ovde isto što i Pahuljica.
- Ne treba nam nijedna pahuljica da bismo je pozvali.

# Klasna metoda može da pravi nove objekte

```
import random

class Pahuljica:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @classmethod
    def nasumicna(cls, sirina, visina):
        x = random.randint(0, sirina)
        y = random.randint(-visina, 0)
        return cls(x, y)

random.seed(3) # da svi dobijemo iste brojeve
p = Pahuljica.nasumicna(640, 360)
print(p.x, p.y)
```

ISPIS

243 -57

- cls(x, y) poziva \_\_init\_\_, kao Pahuljica(x, y).
- Ovo je drugi način da napravimo pahuljicu: sama bira mesto.

# Šta ispisuje ovaj program?

```
class Paleta:  
    boje = ["bela"]  
  
    @classmethod  
    def dodaj(cls, boja):  
        cls.boje.append(boja)
```

```
Paleta.dodaj("plava")  
p = Paleta()  
p.dodaj("siva")  
print(Paleta.boje)
```

Drugi poziv ide preko objekta p.

**A**

```
['bela',  
'plava']
```

**B**

tačno

```
['bela',  
'plava',  
'siva']
```

**C**

```
['siva']
```

**D**

```
TypeError
```

# Statička metoda proverava pre nego što pravimo

```
class Boja:
    def __init__(self, r, g, b):
        self.rgb = (r, g, b)

    @staticmethod
    def je_ispravna(r, g, b):
        for k in (r, g, b):
            if k < 0 or k > 255:
                return False
        return True

print(Boja.je_ispravna(255, 200, 0))
print(Boja.je_ispravna(300, 0, 0))
if Boja.je_ispravna(30, 40, 90):
    nebo = Boja(30, 40, 90)
```

ISPIS

True  
False

- Nema self ni cls: radi samo sa onim što joj damo.
- U klasi je jer logički pripada boji.
- Poziv: Boja.je\_ispravna(r, g, b).

# Taksi služba: brojači su atributi klase

```
class Taksi:
    broj_vozila = 0      # zajednički brojači
    broj_putnika = 0     # za celu službu

    def __init__(self, vozac, km, racun, putnika):
        self.vozac = vozac
        self.km = km
        self.racun = racun
        Taksi.broj_vozila += 1
        Taksi.broj_putnika += putnika

    def cena_po_km(self):
        return self.racun / self.km
```

## broj\_vozila

koliko je taksi vozila evidentirano u službi

## broj\_putnika

koliko je putnika prevezeno, ukupno

## cena\_po\_km

obična metoda: računa za jedno vozilo

*Zadatak iz starih beleški, sa srpskim imenima i ispravljenim komentarima.*

# Taksi služba: klasna i statička metoda

```
@classmethod
def prosek_putnika(cls):
    if cls.broj_vozila == 0:
        return 0
    return cls.broj_putnika / cls.broj_vozila
@staticmethod
def ispravan_put(km):
    return km > 0
```

```
print(Taksi.prosek_putnika())
if Taksi.ispravan_put(80):
    t1 = Taksi("Jovan", 80, 2200, 3)
    t2 = Taksi("Petar", 60, 1500, 1)
    t3 = Taksi("Miloš", 20, 680, 2)
    print(Taksi.broj_vozila)
    print(Taksi.prosek_putnika())
    print(round(t1.cena_po_km(), 2))
    print(Taksi.ispravan_put(-5))
```

## ISPIS

```
0
3
2.0
27.5
False
```

Prvi red: još nema vozila, pa je prosek 0, a ne deljenje nulom.

# Koju metodu da izaberem?

Treba li metodi jedan određeni objekat i njegovi atributi?

da

**obična metoda**

```
def m(self, ...)
```

ne

Treba li joj klasa: atributi klase ili pravljenje objekta?

da

**klasna metoda**

```
@classmethod · cls
```

ne

**statička metoda**

```
@staticmethod · obična funkcija u klasi
```

*Kad niste sigurni: počnite od obične metode.*

# Šta se desi kad pokrenemo ovaj program?

```
class Pahuľjica:  
    ukupno = 0  
  
    @staticmethod  
    def koliko():  
        return ukupno  
  
print(Pahuľjica.koliko())
```

Statička metoda pokušava da pročita atribut klase.

**A**

ispiše 0

**B**

tačno

NameError

**C**

ispiše None

**D**

TypeError

# Kopije objekata

Kad nam treba nezavisan primerak

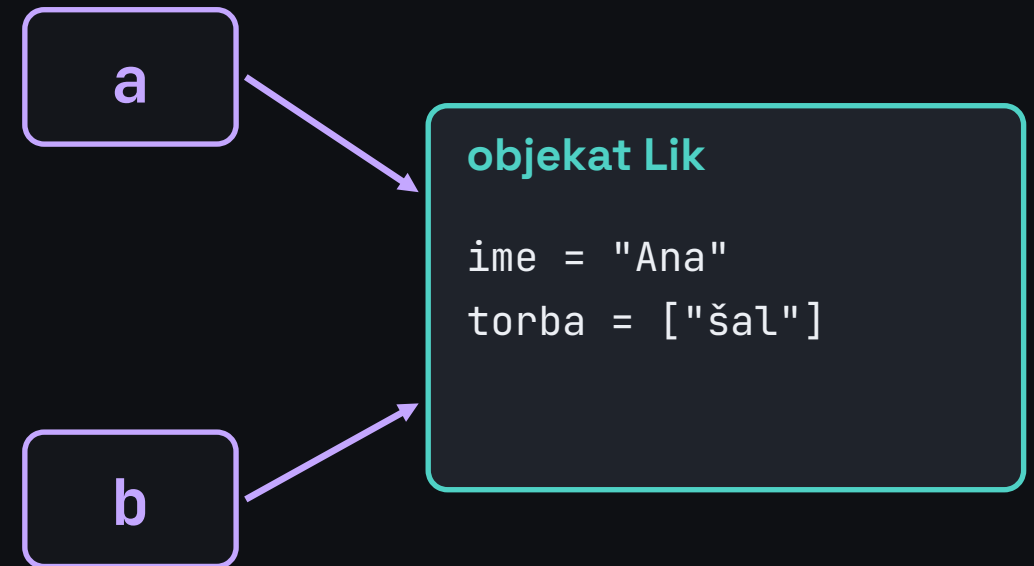
# b = a ne pravi kopiju, nego drugu etiketu

```
class Lik:
    def __init__(self, ime):
        self.ime = ime
        self.torba = ["šal"]

a = Lik("Mia")
b = a                # isti objekat
b.ime = "Ana"
print(a.ime, a is b)
```

ISPIS

Ana True



**a is b → True: jedan objekat, dva imena**

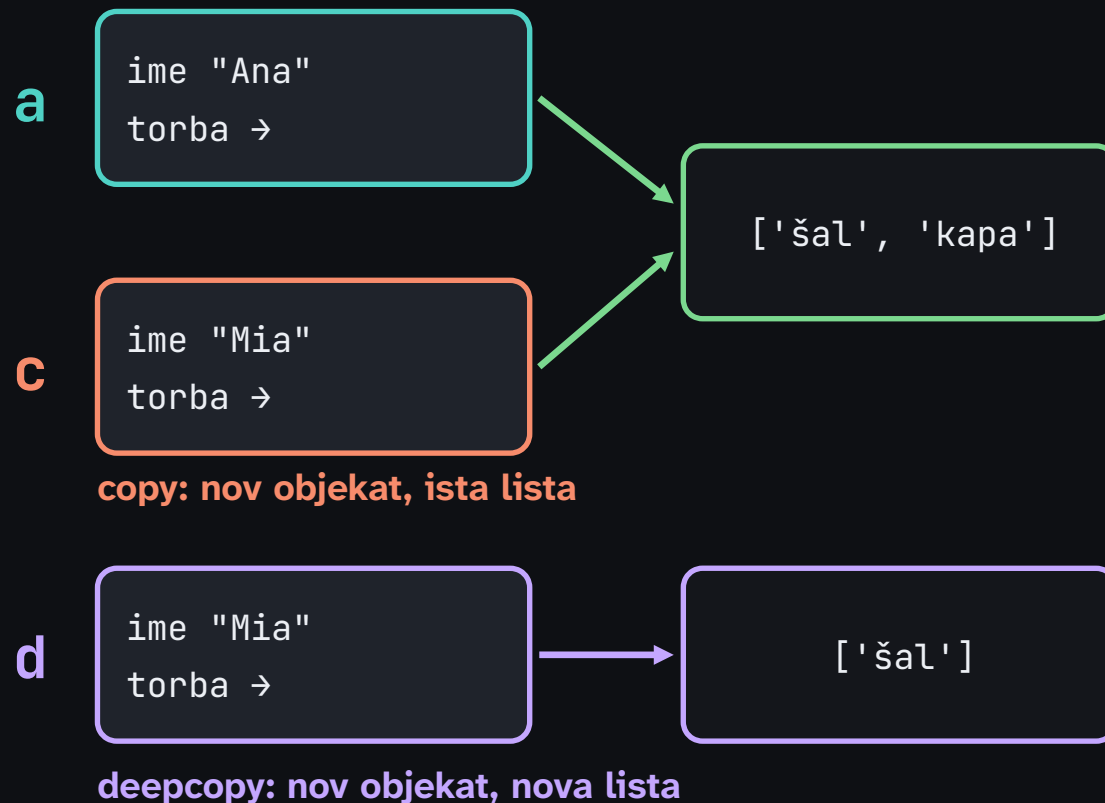
# copy deli unutrašnju listu, deepcopy ne deli

```
import copy

a = Lik("Mia") # torba: ["šal"]
c = copy.copy(a)
d = copy.deepcopy(a)
a.ime = "Ana"
a.torba.append("kapa")
print(c.ime, c.torba)
print(d.ime, d.torba)
print(c is a, c.torba is a.torba)
```

ISPIS

```
Mia ['šal', 'kapa']
Mia ['šal']
False True
```



# Šta ispisuje ovaj program?

```
import copy

paleta = [[255, 0, 0], [0, 0, 255]]
nova = copy.copy(paleta)
nova[0][0] = 0
nova.append([0, 255, 0])
print(len(paleta), paleta[0])
```

Paleta ima dve boje, a svaka boja je lista [r, g, b].

**A**

2 [255, 0, 0]

**B**

3 [0, 0, 0]

**C**

tačno

2 [0, 0, 0]

**D**

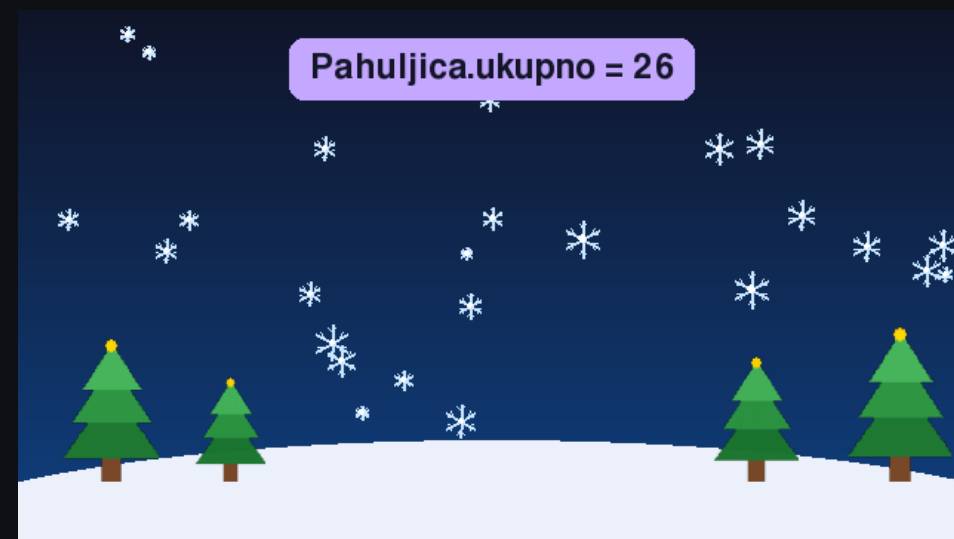
3 [255, 0, 0]

# Zimska šetnja: fabrika pahuljica

**Cilj: klasa Pahuljica koju ćemo koristiti u igri. Zna koliko je pahuljica napravljeno, ume sama da napravi slučajnu pahuljicu i ne dozvoljava prevelike.**

- 1 Atributi klase: ukupno i granice veličine
- 2 `__init__` broji i proverava veličinu
- 3 Statička metoda `ispravna(velicina)`
- 4 Klasna metoda `nasumicna(sirina, visina)`
- 5 Klasna metoda `izvestaj()` i `__str__`

GDE ĆEMO JE KORISTITI



Danas radimo u konzoli; ista klasa crta sneg kad stignemo do prozora i petlje igre.

# Fabrika, prvi deo: brojač i provera

```
class Pahuljica:
    ukupno = 0          # brojač napravljenih
    NAJMANJA = 8        # granice veličine u px
    NAJVECA = 24

    def __init__(self, x, y, velicina):
        if not Pahuljica.ispravna(velicina):
            velicina = Pahuljica.NAJMANJA
        self.x = x
        self.y = y
        self.velicina = velicina
        Pahuljica.ukupno += 1

    @staticmethod
    def ispravna(velicina):
        return Pahuljica.NAJMANJA <= velicina <= Pahuljica.NAJVECA
```

## NAJMANJA

velika slova: vrednost koju ne menjamo

## ispravna

statička metoda, čita granice preko imena klase

# Fabrika, drugi deo: klasne metode i `__str__`

```
@classmethod
def nasumicna(cls, sirina, visina):
    x = random.randint(0, sirina)
    y = random.randint(-visina, 0)
    v = random.randint(cls.NAJMANJA, cls.NAJVECA)
    return cls(x, y, v)
```

```
@classmethod
def izvestaj(cls):
    return f"Napravljeno pahuljica: {cls.ukupno}"
```

```
def __str__(self):
    return f"{self.velicina} px u ({self.x}, {self.y})"
```

`cls.NAJVECA`

klasna metoda čita atribut klase preko cls

`cls(x, y, v)`

pravi pahuljicu: poziva `__init__`

`__str__`

kako pahuljica izgleda u print

# Fabrika radi: pet pahuljica i izveštaj

```
random.seed(5)
sneg = []
for i in range(4):
    sneg.append(Pahuljica.nasumicna(640, 360))
sneg.append(Pahuljica(100, 0, 99)) # prevelika
for p in sneg:
    print(p)
print(Pahuljica.izvestaj())
```

## ISPIS

```
19 px u (637, -230)
22 px u (542, -346)
9 px u (255, -28)
19 px u (160, -303)
8 px u (100, 0)
Napravljeno pahuljica: 5
```

Prevelika pahuljica (99 px) dobija najmanju dozvoljenu veličinu, 8 px.

# Pronađi grešku: brojač jelki ostaje 0

```
class Jelka:
    ukupno = 0

    def __init__(self, x):
        self.x = x
        self.ukupno += 1

for x in (40, 130, 480):
    j = Jelka(x)
    print(Jelka.ukupno, j.ukupno)
```

ISPIS: Jelka.ukupno, j.ukupno

0 1

**self.ukupno += 1**

čita 0 iz klase, upisuje 1 u objekat

**Jelka.ukupno**

zato ostaje 0

**ispravka**

Jelka.ukupno += 1

Program se ne ruši i ne javlja grešku, a napravljene su tri jelke.

# Izaberite nivo i počnite

## LEVEL 1

### Jelke u šumi

Klasa Jelka sa atributima klase boja i ukupno. Napravite četiri jelke, pa jednom naredbom svima promenite boju.

*Pomoć: Jelka.ukupno += 1 u \_\_init\_\_.*

## LEVEL 2

### Paleta sa proverom

Klasa Boja sa statičkom metodom je\_ispravna i klasnom metodom siva(nivo), koja pravi sivu boju.

*Pomoć: siva vraća cls(nivo, nivo, nivo).*

## LEVEL 3

### Galerija

Klasa Slika pamti sve slike u jednoj listi u klasi i ume da nađe najstariju. Kopija za izložbu ima svoje oznake.

*Pomoć: ovde je zajednička lista namerna; kopija je deepcopy.*

# Sledeći put: klase koje nasleđuju klase

```
class ObjekatIgre:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

```
class Pahuljica(ObjekatIgre):  
    ukupno = 0
```

U temi 04 pahuljica, igrač i prepreka dobijaju zajedničku roditeljsku klasu ObjekatIgre: položaj pišemo jednom, a svaka klasa dodaje svoje. Brojač ukupno ostaje tu gde je danas, u klasi Pahuljica.

# Rezime

## atribut klase

```
class Pahuljica:  
    ukupno = 0
```

Jedan za sve; menjamo ga preko klase.

## klasna metoda

```
@classmethod  
def nasumicna(cls, ...):
```

Dobija klasu; pravi objekte sa cls(...).

## statička metoda

```
@staticmethod  
def je_ispravna(r, g, b):
```

Ne dobija ništa; obična funkcija u klasi.

## kopije

b = a     ·     copy     ·     deepcopy

Ista etiketa, plitka i duboka kopija.

# Pitanje pre odlaska

```
class Sneg:  
    pahuljica = 0  
  
a = Sneg()  
a.pahuljica += 1  
print(Sneg.pahuljica, a.pahuljica)
```

Šta ispisuje ovaj program?

Pošaljite dva broja pre nego što izađete.



# Literatura

- Zvanični Python tutorijal, poglavlje 9 „Classes“, odeljak 9.3.5 o atributima klase i instance: [docs.python.org/3/tutorial](https://docs.python.org/3/tutorial)
- Lutz, M. Learning Python, 5. izdanje, O'Reilly, 2013 (delovi o klasama)
- Wentworth, P., Elkner, J., Downey, A. B., Meyers, C. How to Think Like a Computer Scientist: Learning with Python 3 (besplatna knjiga na internetu)
- Bunn, T. Learn Python Visually: Creative Coding with Processing.py, No Starch Press, 2021

# Dodatna literatura i zanimljivi sajtovi

## Python tutorijal: atributi klase

[docs.python.org/3/tutorial/classes.html](https://docs.python.org/3/tutorial/classes.html)

Odeljak 9.3.5 zvaničnog tutorijala, sa primerom zajedničke liste.

## Python: modul copy

[docs.python.org/3/library/copy.html](https://docs.python.org/3/library/copy.html)

Zvaničan opis plitke i duboke kopije.

## Real Python: tri vrste metoda

[realpython.com · instance, class, static](https://realpython.com/instance-class-static/)

Obična, klasna i statička metoda; klasne metode prave različite pice.

## Python Tutor

[pythontutor.com](https://pythontutor.com)

Crta promenljive i objekte: vidi se šta je ista etiketa, a šta kopija.

# Hvala!

Pitanja su uvek dobrodošla: na času, u četvrtu ili međim.

Slajdovi, primeri i kartica za podsećanje su na sajtu predmeta.

## PREDAVANJA

**dr Staša Vujičić Stanković**

[stasa.vujicic.stankovic@math.rs](mailto:stasa.vujicic.stankovic@math.rs)

[www.matf.bg.ac.rs/~stasa.vujicic.stankovic](http://www.matf.bg.ac.rs/~stasa.vujicic.stankovic)

## VEŽBE

**Milan Kocić**

[milan.kocic@matf.bg.ac.rs](mailto:milan.kocic@matf.bg.ac.rs)

[poincare.matf.bg.ac.rs/~milan.kocic](http://poincare.matf.bg.ac.rs/~milan.kocic)