

Za radoznale: Atribut klase, klasna i statička metoda

Ovo nije obavezno i neće biti na ispitu. Za one koji žele da vide kako se ista stvar radi naprednije. Svi ispisi su dobijeni pokretanjem.

1. Boja iz hex zapisa: još jedna klasna metoda koja pravi objekte

U programima za crtanje boje se često pišu u hex zapisu, kao #FFC857: dva znaka za crvenu, dva za zelenu i dva za plavu komponentu, u brojnom sistemu sa osnovom 16. Klasna metoda `iz_hex` pretvara takav zapis u tri broja i pravi objekat klase `Boja`, isto kao `Pahuljica.nasumicna` na času.

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

    @classmethod
    def iz_hex(cls, zapis):
        zapis = zapis.lstrip("#")           # skidamo # sa početka, ako postoji
        r = int(zapis[0:2], 16)             # dva znaka, osnova 16
        g = int(zapis[2:4], 16)
        b = int(zapis[4:6], 16)
        return cls(r, g, b)

    def __str__(self):
        return f"Boja({self.r}, {self.g}, {self.b})"

print(Boja.iz_hex("#FFC857"))             # žuta iz PZU1
print(Boja.iz_hex("C4A7FF"))              # lavanda iz PZU2
```

Ispis:

```
Boja(255, 200, 87)
Boja(196, 167, 255)
```

`int(tekst, 16)` čita broj zapisan u osnovi 16: FF je 255, C8 je 200. Ista klasa sada ima dva načina pravljenja objekta: `Boja(255, 200, 87)` i `Boja.iz_hex("#FFC857")`.

Zašto ovo nismo radili na času: za ovo treba znati brojne sisteme, a na času nam je bio važna sama klasna metoda, a ne zapis boje.

Fajl: `primeri/za_radoznale/03_r1_boja_iz_hexa.py`

2. Isto čudovište u funkciji: lista kao podrazumevana vrednost

Čudovište sa zajedničkom listom ne živi samo u klasama. Podrazumevana vrednost parametra pravi se samo jednom, kad Python pročita `def`, a ne pri svakom pozivu. Ako je ta vrednost lista, svi pozivi dele istu listu.

```
def dodaj_boju(boja, paleta=[]):
    paleta.append(boja)
    return paleta

print(dodaj_boju("plava"))
print(dodaj_boju("bela"))           # očekujemo samo ['bela']

def dodaj_boju_ispravno(boja, paleta=None):
    if paleta is None:               # nova lista pri svakom pozivu
        paleta = []
    paleta.append(boja)
    return paleta

print(dodaj_boju_ispravno("plava"))
print(dodaj_boju_ispravno("bela"))
```

Ispis:

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

Drugi poziv prve funkcije vraća i plavu boju iz prvog poziva. Rešenje je isto kao na času: listu pravimo tek u telu funkcije, kad je zaista potrebna. None je dogovoren znak za „nije prosleđeno“.

Zašto ovo nismo radili na času: podrazumevane vrednosti parametara smo samo pomenuli, a ova greška se pojavljuje tek u većim programima.

Fajl: `primeri/za_radoznale/03_r2_lista_kao_podrazumevana_vrednost.py`

3. dataclass: Python sam piše init

Kad klasa uglavnom samo čuva podatke, Python može sam da napiše **init**, lep ispis i poređenje objekata. Dovoljno je iznad klase napisati dekorator `@dataclass` i navesti attribute sa tipovima. Za listu koristimo `field(default_factory=list)`, pa svaki objekat dobija svoju novu listu.

```
from dataclasses import dataclass, field

@dataclass
class Pahuljica:
    x: int
    y: int
    velicina: int = 12
    oznake: list = field(default_factory=list)    # svakoj pahuljici nova lista

p = Pahuljica(10, 0)
print(p)
q = Pahuljica(10, 0)
q.oznake.append("zlatna")
print(p.oznake, q.oznake)
print(p == Pahuljica(10, 0))
```

Ispis:

```
Pahuljica(x=10, y=0, velicina=12, oznake=[])  
[] ['zlatna']  
True
```

Ispis `Pahuljica(x=10, y=0, ...)` dobili smo bez pisanja **str**, a poređenje sa `==` upoređuje attribute. Zanimljivo: ako napišemo oznake: `list = []`, `dataclass` javlja `ValueError` i ne dozvoljava naše čudovište.

Zašto ovo nismo radili na času: na času želimo da vidimo šta tačno radi **init**, a `dataclass` to sakriva.

Fajl: `primeri/za_radoznale/03_r3_dataclass.py`