

Za radoznale: Enkapsulacija, polimorfizam i specijalne metode

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. repr: kako se objekat vidi u listi

Na času smo napisali **str**, pa `print(tacka)` lepo ispisuje (2, 3). Ali kad ispišemo listu tačaka, Python za svaki element ne poziva **str**, nego drugu specijalnu metodu, **repr**. Ako je nema, opet vidimo adrese u memoriji.

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

    def __str__(self):
        return f"({self.x}, {self.y})"

tacke = [Tacka(1, 2), Tacka(3, 4)]
print(tacke[0])
print(len(str(tacke)) > 20)      # u listi se vide adrese, pa je tekst dugačak

class Tacka2(Tacka):
    def __repr__(self):
        return f"Tacka2({self.x}, {self.y})"

print([Tacka2(1, 2), Tacka2(3, 4)])
```

Ispis:

```
(1, 2)
True
[Tacka2(1, 2), Tacka2(3, 4)]
```

Dogovor je da **str** daje lep opis za korisnika, a **repr** opis za programera, najbolje baš onakav kakav bi se napisao u kodu. Kad klasa ima samo **repr**, `print` koristi i njega.

Zašto ovo nismo radili na času: za ispis jednog objekta dovoljan je **str**, a dve metode za ispis bi na prvom susretu sa specijalnim metodama više zbunile nego pomogle.

Fajl: `primeri/za_radoznale/05_r1_repr.py`

2. rmul: kad je broj sa leve strane

Na času smo videli da `brzina * 2` radi, a `2 * brzina` daje `TypeError`. Kad Python izračunava `2 * brzina`, prvo pita broj 2 da li zna da se pomnoži tačkom. Ne zna, pa onda pita tačku da li ima metodu **rmul** („množenje sa desne strane“).

```

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

    def __str__(self):
        return f"({self.x}, {self.y})"

    def __mul__(self, broj):
        return Tacka(self.x * broj, self.y * broj)

    def __rmul__(self, broj):
        return self * broj           # isto kao tacka * broj

t = Tacka(3, 4)
print(t * 2)
print(2 * t)

```

Ispis:

```

(6, 8)
(6, 8)

```

rmul samo poziva **mul**, jer je kod tačke i broja redosled nebitan. Isto postoje **radd**, **rsub** i ostali. Vector2 ih već ima, pa `2 * brzina` radi i sa njim.

Zašto ovo nismo radili na času: za igru je dovoljno da uvek pišemo vektor sa leve strane (`brzina * dt`), a pravilo „prvo pita levi, pa desni objekat“ je detalj koji nije potreban za ispit.

Fajl: `primeri/za_radoznale/05_r2_rmul.py`

3. @dataclass: Python sam piše init, repr i eq

Klase koje samo čuvaju podatke (tačka, boja, rezultat igre) uvek imaju isti **init** koji prepisuje parametre u atribut. Modul `dataclasses` iz standardne biblioteke to piše umesto nas, zajedno sa **repr** i **eq**.

```

from dataclasses import dataclass

@dataclass
class Rezultat:
    ime: str
    poeni: int = 0

a = Rezultat("Ana", 120)
b = Rezultat("Ana", 120)
print(a)
print(a == b)
print(Rezultat("Mia"))

```

Ispis:

```

Rezultat(ime='Ana', poeni=120)
True
Rezultat(ime='Mia', poeni=0)

```

Iza imena atributa piše njegov tip (str, int), a posle znaka jednakosti podrazumevana vrednost. Ispis i poređenje rade bez ijedne metode koju smo sami napisali.

Zašto ovo nismo radili na času: da bismo razumeli šta dataclass radi, prvo je trebalo da sami napišemo **init**, **str** i **eq**, a zapis sa tipovima (ime: str) nismo radili.

Fajl: `primeri/za_radoznaLe/05_r3_dataclass.py`