

Za radoznale: Nasleđivanje

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. Kojim redom Python traži metodu: MRO

Kod višestrukog nasleđivanja Python pravi jedan spisak klasa kojim redom traži metode. Taj spisak se zove MRO (method resolution order, redosled traženja metoda) i čuva se u atributu **mro**. Zanimljivo je da `super()` ne ide uvek „roditelju“, nego sledećoj klasi u tom spisku. Ovde ZlatnaPahuljica nasleđuje Pahuljicu i Svetlo, a obe nasleđuju ObjekatIgre: crtež porodice liči na romb.

```
class ObjekatIgre:
    def opis(self):
        return "objekat"

class Pahuljica(ObjekatIgre):
    def opis(self):
        return "pahuljica → " + super().opis()

class Svetlo(ObjekatIgre):
    def opis(self):
        return "svetlo → " + super().opis()

class ZlatnaPahuljica(Pahuljica, Svetlo):
    def opis(self):
        return "zlatna → " + super().opis()

print(ZlatnaPahuljica().opis())
print([k.__name__ for k in ZlatnaPahuljica.__mro__])
```

Ispis:

```
zlatna → pahuljica → svetlo → objekat
['ZlatnaPahuljica', 'Pahuljica', 'Svetlo', 'ObjekatIgre', 'object']
```

`super()` u klasi Pahuljica ovde ne poziva ObjekatIgre, nego Svetlo, jer je Svetlo sledeće u spisku. Zato se svaka klasa pojavi tačno jednom, a ObjekatIgre na kraju. Klasa `object` na kraju spiska je roditelj svih klasa u Pythonu.

Zašto ovo nismo radili na času: na času nam je bilo dovoljno pravilo „važi klasa napisana prva u zagradi“, a romb od klasa u malim igrama retko treba.

Fajl: `primeri/za_radoznale/04_r1_mro.py`

2. Apstraktna klasa: roditelj od koga se ne pravi objekat

Klasa ObjekatIgre postoji samo da bi je druge nasleđivale; „samo objekat igre“ u igri ne postoji. Modul `abc` omogućava da to kažemo i Pythonu: klasa nasleđuje ABC, a metoda koju svaka izvedena klasa mora da

napiše dobija dekorator `@abstractmethod`. Tada Python ne dozvoljava pravljenje objekta roditeljske klase.

```
from abc import ABC, abstractmethod

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

    @abstractmethod
    def opis(self):
        ...

class Pahuljica(ObjekatIgre):
    def opis(self):
        return f"pahuljica u ({self.x}, {self.y})"

print(Pahuljica(10, 0).opis())
try:
    ObjekatIgre(5, 5)
except TypeError as greska:
    print("Ne može:", greska)
```

Ispis:

```
pahuljica u (10, 0)
Ne može: Can't instantiate abstract class ObjekatIgre without an implementation for
abstract method 'opis'
```

Pahuljica je napisala opis, pa se pravi bez problema. `ObjekatIgre(5, 5)` daje `TypeError`: Python podseća da je ova klasa samo nacrt za izvedene klase. Ako izvedena klasa zaboravi da napiše opis, ni od nje se ne može napraviti objekat.

Zašto ovo nismo radili na času: za ovo je potreban još jedan modul i još jedan dekorator, a na času smo želeli da vidimo samo nasleđivanje.

Fajl: `primeri/za_radoznale/04_r2_apstraktna_klasa.py`

3. Kompozicija: „ima“ umesto „jeste“

Nasleđivanje opisuje odnos „jeste“: pahuljica jeste objekat igre. Mnogo stvari u igri su odnos „ima“: igrač ima torbu, nije torba. Tada objekat jedne klase čuvamo kao atribut druge. To se zove kompozicija. Programeri igara je često biraju umesto velikih drveta klasa.

```

class Torba:
    def __init__(self):
        self.stvari = []

    def dodaj(self, stvar):
        self.stvari.append(stvar)

class Igrac:
    def __init__(self, ime):
        self.ime = ime
        self.torba = Torba()      # igrač IMA torbu

mia = Igrac("Mia")
mia.torba.dodaj("šal")
mia.torba.dodaj("ključ")
print(mia.ime, mia.torba.stvari)
print(isinstance(mia, Torba))

```

Ispis:

```

Mia ['šal', 'ključ']
False

```

Igrac ne nasleđuje Torba, pa `isinstance(mia, Torba)` je `False`. Ali igrač ima torbu u atributu i preko nje dodaje stvari. Ovo je i pravi način da svaki igrač ima svoju torbu, bez čudovišta iz teme 03.

Zašto ovo nismo radili na času: kompozicija je drugi način slaganja klasa; o tome kako se biraju između nasleđivanja i kompozicije piše poglavlje „Component“ sa slajda sa sajtovima.

Fajl: `primeri/za_radoznale/04_r3_kompozicija.py`