Методе у класама

У Пајтон класи се може дефинисати три врсте метода: методе инстанци, методе класе и статичке методе.
У примеру су дефинисане све три врсте метода у истој класи:

```
class MojaKlasa:
    def metoda1(self):
        return 'pozvana je metoda instance', self

    @classmethod
    def metoda2(cls):
        return 'pozvana je metoda klase', cls

    @staticmethod
    def metoda3():
        return 'pozvana je staticka metoda'

a = MojaKlasa()
print(a.metoda1())
print(a.metoda2())
print(a.metoda3())
```

Даје:

('pozvana je metoda instance', <__main__.MojaKlasa object at 0x0000028B39F84D00>)
('pozvana je metoda klase', <class '__main__.MojaKlasa'>)
pozvana je staticka metoda

Прва метода (metoda1) је класична метода инстанце.
Метода узима један или више параметар (self у овом случају), који упућује на инстанцу класе када се метод позове.
Преко self параметра метод инстанце може приступити атрибутима и другим методама истог објекта.
Види се метода1 има приступ објекту инстанце преко self аргумента.
Приликом позива методе, замењује се self аргумент са објектом инстанце (у овом случају то је a).
Друга метода (metoda2) је метода класе и дефинише се коришћењем декоратера (`@classmethod`).
Ова метода узима cls параметар који по дифолту упућује на класу у којој је метода дефинисана (у овом случају то је класа MojaKlasa).
Метода класе не може променити стање објекта јер нема приступ аргументу self, али може променити стање класе.
Види се да метода има приступ само класи као посебном објекту.
Трећа метода (metoda3) је статичка метода и дефинише се коришћењем декоратера (`@staticmethod`).
Метода не узима нити self нити cls параметар али може узети неограничен број других параметара.
Статичка метода не може променити стање нити објекта нити класе и најчешће се користе за дефинисање досега (namespace) метода.
Види се да је и у овом случају позив методе ишао преко објекта инстанце али у позадини Пајтон ограничава статичку методу само на досег класе.

Вишеструко наслеђивање

У принципу, вишеструко наслеђивање је једноставан концепт: подкласа (дете класа) која наслеђује од више надкласа (родитељ класа) може приступити функционалностима свих својих надкласа.
Пример:

```
class B:
    def b(self):
        print('b')

class C:
    def c(self):
        print('c')

class D(B, C):
    def d(self):
        print('d')
```

```
d = D()
d.b()                #b
d.c()                #c
d.d()                #d
```

Види се да подкласа D наслеђује од надкласа B и C (све из њих па и методе надкласа) и када се позову одређене методе из надкласа нема никаквих проблема пошто су методе назване посебним именима.

```
class B:
    def x(self):
        print('x: B')

class C:
    def x(self):
        print('x: C')

class D(B, C):
    pass

d = D()
d.x()
print(D.mro())
```

Даје: x: B

[<class '__main__.D'>, <class '__main__.B'>, <class '__main__.C'>, <class 'object'>]

У примеру се две надкласе које имају методу истог имена (x) а подкласа наслеђује исту методу из обе надкласе.

Види се из излаза да је позивање методе изазвало појаву вредности из прве наведене надкласе у дефиницији подкласе (B, C).

Постоји концепт у програмирању којим се решавају овакви проблеми и који се назива редослед вишеструког извршавања (MRO – multiple resolution order).

MRO подкласе одлучује где ће Пајтон прво да потражи наведену методу и тиме се разрешава потенцијални конфликт.

Редослед претраге методе је следећи: подкласа, прва наведена надкласа па следеће по редоследу, класа object.

Проблем је у практичној употреби оваквог концепта.

Најједноставнија и најчешће коришћена форма вишеструког наслеђивања се назива mixin.

Она је надкласа која не би требало да постоји самостално, већ постоји да би се наслеђивала од стране других класа да би придодала некакву функционалност подкласама.

Нпр, хоћемо додати функционалност класи Kontakti која омогућава слање емејлова према self.email.

Пример:

```
class Kontakti:
    svi_kontakti = []

    def __init__(self, ime, email):
        self.ime = ime
        self.email = email
        self.svi_kontakti.append(self)

class SlanjeEmail:
    def salji_email(self, poruka):
        print("Saljem email prema " + self.email)
        #ovde se dodaje kod za rad sa emajlovima

class EmailOmogucenKontakt(Kontakti, SlanjeEmail):
    pass

e = EmailOmogucenKontakt("Miki Mikic", "mmikic@primer.net")
e.salji_email("Zdravo, ovo je testiranje emejla.")
```

Класа SlanjeEmail не ради ништа посебно али омогућава дефинисање нове класе која описује класе Kontakti и SlanjeEmail, коришћењем вишеструког наслеђивања.
Синтакса за вишеструко наслеђивање изгледа као листа параметара у дефиницији класе.
Уместо укључења једне основне класе унутар заграда, овде су укључене две (може и више), раздвојене зарезом.
Тестирање се изводи употребом mixin форме:

```
e = EmailOmogucenKontakt("Miki Mikic", "mmikic@primer.net")
e.salji_email("Zdravo, ovo je testiranje emejla.")
```

Даје:

```
[<__main__.EmailOmogucenKontakt object at 0x000001F65FCA8358>]
Saljem email prema mmikic@primer.net
```

Израда лабораторијских вежби:

Задатак 049: Класа Контакти има атрибут класе сви_контакти и иницијализатор са атрибутима име и емејл и понашање да се надовезује у сви_контакти сваки унос вредности атрибута објеката. Сви_контакти се проширује са новом класом КонтактЛиста која уводи методу инстанце претрага са атрибутом име. Ова метода враћа све контакте који имају тражену вредност атрибута име.

```
class KontaktLista(list):
    def pretraga(self, ime):
        '''Vraca sve kontakte koji imaju trazenu vrednost u imenu.'''
        trazeni_kontakti = []
        for kontakt in self:
            if ime in kontakt.ime:
                trazeni_kontakti.append(kontakt)
        return trazeni_kontakti

class Kontakti:
    svi_kontakti = KontaktLista()
    def __init__(self, ime, emejl):
        self.ime = ime
        self.emejl = emejl
        self.svi_kontakti.append(self)

k1 = Kontakti("Miki A", "mikia@primer.net")
k2 = Kontakti("Miki B", "mikib@primer.net")
k3 = Kontakti("Ana C", "anac@primer.net")
print([a.ime for a in Kontakti.svi_kontakti.pretraga('Miki')])
```

Задатак 050: Класа Пас има атрибут класе врста, иницијализатор са атрибутима инстанце име и године, и две методе инстанце: опис (исписује поруку о старости пса), лајање (исписује поруку). Подкласе Теријер и Булдог имају своје методе трчи са атрибутом брзина која даје поруку. Коришћењем функције isinstance() проверити који објекти су инстанце које класе.

```
class Pas:
    vrsta = 'sisar'
    def __init__(self, ime, godine):
        self.ime = ime
        self.godine = godine

    def opis(self):
        return "{} je {} godina star.".format(self.ime, self.godine)

    def lajanje(self, zvuci):
        return "{} kaze {}".format(self.name, zvuci)

class Terijer(Pas):
    def trci(self, brzina):
        return "{} trci {}.".format(self.ime, brzina)
```

```
class Buldog(Pas):
    def trci(self, brzina):
        return "{} trci {}.".format(self.ime, brzina)

oliver = Buldog("Oliver", 12)
print(oliver.opis())
print(oliver.trci("polako"))
print(isinstance(oliver, Pas))
julija = Pas("Julija", 100)
print(isinstance(julija, Pas))
dzonivoker = Terijer("Dzoni Voker", 4)
print(isinstance(dzonivoker, Buldog))
print(isinstance(julija, oliver))
```

Daje:
Oliver je 12 godina star.
Oliver trci polako.
True
True
False
Traceback (most recent call last):
  File "C:\Users\nera\source\repos\PythonApplication6\PythonApplication6.py", line 29, in <module>
    print(isinstance(julija, oliver))
TypeError: isinstance() arg 2 must be a type or tuple of types

Задатак 051: Коришћењем класа из задатка 050 испитати могућност промене (override) атрибута родитељске класе у класи деце. Променити атрибут класе у једној од класа деце.

```
class Pas:
    vrsta = 'sisar'
    def __init__(self, ime, godine):
        self.ime = ime
        self.godine = godine
        print("{} je {} i ima {} godina.".format(self.ime, self.vrsta, self.godine))

class Terijer(Pas):
    pass

class Buldog(Pas):
    vrsta = 'dinosaurus'

julija = Terijer("Julija", 13)
oliver = Buldog("Oliver", 10000000)
```

Daje:
Julija je sisar i ima 13 godina.
Oliver je dinosaurus i ima 10000000 godina.

Задаци за самосталан рад:

46. Креирати класу Љубимци која има атрибут класе пси (листа) и иницијализатор који има атрибут листу пси. У тој листи ће се наћи инстанце класе Пас; ова класа је потпуно одвојена од класе Пас. Другим речима, класа Пас не наслеђује из класе Љубимци. Доделити три инстанце паса инстанци класе Љубимци. Излаз треба да изгледа овако:
Imam 3 psa.
Miki ima 6 godina.
Oliver ima 7 godina.
Julija ima 9 godina.
Svi su vrste sisar.

47. Коришћењем истог кода додати инстанцу атрибута је_гладан = True у класу Пас. Затим додати методу једи која мења вредност је_гладан на False када се позове. Осмислити најбољи начин за храњење сваког пса и дати поруку да су пси гладни (ако су сви гладни) или поруку да пси нису гладни (ако сви нису гладни). Излаз треба да изгледа овако:
Imam 3 psa.
Miki ima 6 godina.
Oliver ima 7 godina.
Julija ima 9 godina.
Svi su vrste sisar.
Moji psi nisu gladni.