Objektno programiranje
⟨ Klase, objekti ⟩

Tihomir Žilić, Mario Essert, Vladimir Milić

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

Zagreb, 2020./2021.

Sadržaj

1. Klase
   - Stvaranje klase naredbom `class`
   - Atributi klase
   - Metode klase
   - Instance klase - objekti
   - Atributi objekata
   - Više objekata i problemi atributa
   - Privatni atributi klase
   - Inicijalizacija objekta
   - Metaklasa `type`
   - Posebne metode

Klase (ili razredi)

Prožimanje podataka i funkcija unutar jednog razreda (klase).

Primjer klase: Školski razred,

- podaci - ime, prezime, godine, spol;
- funkcije - usmeni, pismeni ispit, zadaće, ocjene predmeta, druženje, aktivnosti.

Primjer klase: Abeceda,

- podaci - broj slova, tip abecede;
- funkcije - stvaranje riječi, prebrojavanje slova u toj riječi.

Primjer klase: Živa bića,

- podaci - visina, širina, dužina;
- funkcije - rast, razmnožavanje, hranjenje.

Primjer klase: Proizvodnja energije,

- podaci - tip elektrane, mjerena satna proizvodnja;
- funkcije - ukupna proizvodnje, akumulacija.

Stvaranje klase naredbom **class**

```
class Ime_klase:
    tijelo klase
```

ili

```
class Ime_klase(klase_roditelji):
    tijelo klase
```

Primjer stvaranje klase **MojaKlasa**:

```
>>> class MojaKlasa:
        a=1
```

- Ime klase prema dogovoru počinje velikim slovom, npr. **MojaKlasa**.
- Ime klase je bez zagrada, npr. **MojaKlasa:**, osim kod nasljeđivanja npr. **MojaKlasa(roditelji):**.

Atributi klase

- **Atributi klase** su **podaci**, **funkcije** i **metode** unutar klase.
- Metoda je funkcija u klasi koja ima prvi parametar `self` ili je dekorator.
- Za pristup atributu koristi se sintaksa s točkom, **`klasa.atribut`**.

```
class MojaKlasa:
    '''Ovo je moja klasa'''
    a = 9                  # atribut klase, podatak
    def kvad(x):           # atribut klase, funkcija
        return x**2
    def f(self,y):         # atribut klase, metoda
        return MojaKlasa.a + y + MojaKlasa.kvad(3)

>>> MojaKlasa.a
9
>>> MojaKlasa.kvad(5)
25
>>> MojaKlasa.f("a", 5)
23
```

Ugrađeni atributi klase

Klasama se pri stvaranju automatski pridjeljuju posebni atributi poput:

- **__name__**, ime klase,
- **__doc__**, dokumentacija (opis) klase,
- **__bases__**, n-terac objekata roditeljskih klasa u naredbi `class`,
- **__class__**, ime klase objekta, tj. isto kao **type(ime)**,
- **__dict__**, rječnik svih atributa klase.
- ...

Primjer:

```
>>> MojaKlasa.__name__
'MojaKlasa'
>>> MojaKlasa.__doc__
'Ovo je moja klasa'
>>> MojaKlasa.__bases__
(<class 'object'>,)
>>> MojaKlasa.__class__
<class 'type'>
>>> MojaKlasa.__dict__["a"]
9
```

Promjena, dodavanje i brisanje atributa klase

Promjena i dodavanje te brisanje atributa klase izvodi se na sljedeće načine:

- promjena atributa:

```
>>> MojaKlasa.a = 10 # promjena atributa a na 10 (a bio je 9)
>>> MojaKlasa.a      # ocitavanje vrijednosti atributa
10
```

- dodavanje atributa:

```
>>> MojaKlasa.c = "ja sam novi"  # dodavanje atributa c
>>> MojaKlasa.c
'ja sam novi'
```

- brisanje atributa naredbom `del`:

```
>>> del MojaKlasa.a   # brisanje atributa a
>>> MojaKlasa.a
AttributeError: type object 'MojaKlasa' has no attribute 'a'
```

Metode klase

Metode klase su funkcije definirane:

- **`self`** parametrom koji je vezan uz ime objekta izvedenog iz te klase:

```
def ime_funk(self,arg):
```

  Koristi se za poziv metode korištenjem: `objekt.ime_funkc(arg)`.

- kao **`@classmethod`** dekorator (iznad funkcije) tj. preko parametra koji je vezan uz ime te klase, npr. **`cls`** parametar:

```
@classmethod
def ime_funk(cls,arg):
```

  Koristi se za poziv metode korištenjem: `klasa.ime_funkc(arg)`

- kao **`@staticmethod`** dekorator (iznad funkcije):

```
@staticmethod
def ime_funk(...):
```

  Koristi se za poziv metode nevezane uz ime objekta ili klase.

Funkcionalnost klase korištenjem `self`

`self`:

- je ime prvog parametra metode (ako nije dekorator),
- je implicitni parametar, što znači da pri pozivu metode iz objekta on se ne vidi kao argument,
- pri stvaranju objekta iz klase povezuje se s imenom objekta, i tako specificira taj objekt,
- sve metode unutar klase koje ga sadrže imaju pristup njenim elementima i to je način kako se prenose identifikatori među metodama.

Instance klase - objekti, engl. *instances*

Nastaju iz klase i imaju sva obilježja te klase.

```
ime_objekta = Ime_klase()
```

- Pri stvaranju objekta iz klase potrebno je uz ime klase dodati i oble zagrade, kao `Ime_klase()`.
- Stvoreni objekt se znakom pridružbe (=) povezuje s identifikatorom `ime_objekta` i čije ime nosi.
- Pri stvaranju objekta parametar `self` unutar klase povezuje se s `ime_objekta`.

Primjer stvaranje objekta **`mojobjekt`** iz klase **`MojaKlasa`**:

```
mojobjekt = MojaKlasa()
```

Atributi objekata

Atributi objekta su **podaci** i **metode**. Funkcije unutar objekta mogu se pozvati samo ako su metode.

```
class MojaKlasa:
        '''Ovo je moja klasa'''
        a = 9                   # atribut klase, podatak
        def kvad(x):            # atribut klase, funkcija
            return x**2
        def f(self,y):          # atribut klase, metoda
            return MojaKlasa.a + y + MojaKlasa.kvad(3)

>>> mk = MojaKlasa() # stvaranje objekta mk iz klase MojaKlasa
>>> mk.a                # atribut objekta, podatak
9
>>> mk.f(6)             # atribut objekta, metoda
24
>>> mk.kvad(5)          # atribut objekta, funkcija
TypeError: kvad() takes 1 positional argument but 2 were given
```

Promjena vrijednosti atributa objekta

Atributi pojedinačnog objekta mogu se mijenjati.

```
class Prom:
        a = 9                   # atribut klase, integer podatak
        def f(self,y):          # atribut klase, metoda
            return y + Prom.a

>> mk = Prom()  # stvaranje objekta mk iz klase Prom
>>> print(mk.a, mk.f(3))     #ispis atributa objekta mk
9 12
>>> mk.a = "atribut podatak" #promjena atributa a
>>> def k(x): return 2*x      # nova funkcija k() izvan objekta
>>> mk.f = k # promjena sadržaja identifikatora mk.f s objekta f na k
>>> print(mk.a, mk.f(3))
atribut podatak 6
```

Više objekata i problemi atributa

Atribute klase sadrže svi objekti nastali iz iste klase. Atribute klase s **promjenjivim tipom** podatka, popt liste ili rječnika koristiti s oprezom.

```
class Prom:
        a = 9                   # atribut klase, integer podatak
        b = [1,2,3]             # atribut klase, lista podatak

>>> mk1 = Prom()  # stvaranje objekta mk1 iz klase Prom
>>> mk2 = Prom()  # stvaranje objekta mk2 iz klase Prom
>>> print(mk1.a, mk1.b)
9 [1, 2, 3]
>>> print(mk2.a, mk2.b)
9 [1, 2, 3]
>>> mk1.a = 10      # promjena integera u mk1
>>> print(mk1.a, mk2.a)
10 9
>>> mk1.b[1] = "s" # promjena u LISTI u mk1 utjece na mk1 i mk2
>>> print(mk1.b, mk2.b)
[1, 's', 3] [1, 's', 3]
```

Primjer:

```
class MojaKlasa:
    a = 9
    def f(self,y):
        return MojaKlasa.mult(5) + y*self.mult(5) + MojaKlasa.kvad(3)
    @staticmethod
    def kvad(x):
        return x**2
    @classmethod
    def mult(cls,g):
        return cls.kvad(g) + g + cls.a

>>> m = MojaKlasa()
>>> print("Iz objekta:", m.f(3))
Iz objekta: 165
>>> print("Iz klase:", MojaKlasa.mult(3))
Iz klase: 21
>>> print("Iz objekta:", m.mult(3))
Iz objekta: 21
>>> print("Iz klase:", MojaKlasa.kvad(3))
Iz klase: 9
>>> print("Iz objekta:", m.kvad(3))
Iz objekta: 9
```

Primjer:

```
class Prijenos:
    def A1(self):
        self.K = "Dobar "
        return "prijatelju"
    def A2(self, x):
        print(self.K + x + self.A1())

>>> p1 = Prijenos()
>>> p1.A1()          # spremanje varijable K u self (p1) objekt
'prijatelju'
>>> p1.A2("dan ") # potrebno je prije pozvati p1.A1()
Dobar dan prijatelju
```

Privatni atributi klase

Identifikator (npr. `ident`) u klasi:

- koji počinje s jednom podcrtom, kao npr. `_ident`, govori o tome da je to privatni atribut klase i da ga korisnik ne mijenja (iako može na isti način kao i klasične metode bez podcrta);
- koji počinje s dvije podcrte, kao npr. `__ident`, Python compiler implicitno mijenja u `_classname__ident`, gdje je `classname` ime klase. Ovo omogućuje da takvi privatni atributi ostanu nepromijenjeni pri nasljeđivanju klasa;
- prethodno dva navedena mogu također završavati bez podcrte ili najviše s jednom, npr. `_ident_` ili `__ident_`;
- ako počinje i završava s dvije podcrte, to je Python-ova ugrađena metoda! Može ju se pod tim imenom predefinirati pri konstruiranju klase;
- svi privatni atributi objekta (klase) su vidljivi i može im se pristupiti "izvana"!

Primjer privatni atributi klase

```
class Moje:
    def __privatna(self):
        x = ":-)"
        return x
    def javna(self):
        print("Privatna poruka glasi: ", self.__privatna())

>>> m = Moje()
>>> m.__privatna()
AttributeError: 'Moje' object has no attribute '__privatna'
>>> m._Moje__privatna()
':-)'
>>> m.javna()
Privatna poruka glasi:  :-)
```

Inicijalizacija objekta

Stvoreni objekt **self** sadrži samo definicije metoda, pa je potrebna inicijalizacija. Posebna metoda **__init__** inicijalizira objekt automatski odmah nakon njegovog stvaranja.
Napomena: **__init__** je metoda koja treba vraćati `None`, pa se naredba `return None` može izostaviti.

```
class Prijenos:
    def __init__(self):
        self.K = "Dobar "
        self.A1 = " prijatelju"
    def A2(self, x):
        print(self.K + x + self.A1)

>>> p1 = Prijenos()
>>> p1.A2("dan")
Dobar dan prijatelju
```

Različita inicijalizacija objekta

Objekti se pri stvaranju mogu inicijalizirati različitim početnim argumentima. Za to je potrebno u metodu **__init__** postaviti dodatne parametre nakon parametra **self**.
Objekt se stvara i inicijalizira pozivom klase s argumentima.

```
class Prijenos:
    def __init__(self,y):
        self.K = y
    def A2(self, x):
        print(self.K + x)

>>> p1 = Prijenos("Dobar ")  # poziv klase s "Dobar "
>>> p1.A2("dan")
Dobar dan

>>> p2 = Prijenos("Super ")  # poziv klase sa "Super "
>>> p2.A2("dan")
Super dan
```

Promjenjivi tipovi podataka kao atributi ili u metodi

Atributi klase ostaju isti u svim objektima iz te klase, a ako se u bilo kojem promijeni, mijenja se svugdje. Podaci ako su unutar metode onda pripadaju samo tom objektu.

```
class Prijenos:
    z = ["a"]
    def __init__(self):
        self.g = [1]

>>> p1 = Prijenos()    # objekt p1
>>> p2 = Prijenos()    # objekt p2
>>> p1.z[0] = "b"      # promjena atributa z (lista)
>>> p1.g[0] = 2        # promjena podatka g (lista) u metodi
>>> print("Objekt p1:", p1.z, p1.g) # poziv z i g iz p1
Objekt p1: ['b'] [2]
>>> print("Objekt p2:", p2.z, p2.g) # poziv z i g iz p2
Objekt p2: ['b'] [1]
```

Istoimeni podaci kao atributi i u metodi

Podaci atributi ostaju isti u svim objektima stvorenim iz iste klase. Ti objekti sadrže i podatke atribute kao i podatke iz metoda, a ako ima istoimenih, onda će odabrati podatke iz metode.

Primjer:

```
class Prijenos:
    z = "z-podatak atribut"
    c = "c-podatak atribut"
    def __init__(self):
        self.z = "z-podatak metode"

>>> pokazi = Prijenos()  # stvaranje objekta 'pokazi' iz klase Prijeno
>>> pokazi.z      # 'z-podatak metode'
>>> pokazi.c      # 'c-podatak atribut'
>>> Prijenos.z    # 'z-podatak atribut'
>>> Prijenos.c    # 'c-podatak atribut
```

Metaklasa `type`

Ideja: *podaci* i *funkcije* zajedno u klasi (razredu).

- **METAKLASA** (defaultna ima naziv **`type`**)
- Metaklasa je stvaratelj klasa (tzv. tipova podataka)
  - **KLASA** je objekt nastao iz metaklase
  - `ime klase` je ime tipa podatka, npr. `int`, `float`, `string`, `function`, `list`, `tuple`,...
    - **OBJEKTI** nekog tipa nastaju iz klase tog tipa,
    - npr. iz klase `float` nastaju objekti 4.1,88.8,...
    - s npr. pripadajućim metodama `>>> 88.8.__add__(34)` isto što i `>>> 88.8+34`, ...

`type(objekt)` vraća ime klase `objekta`:

```
>>> type(88.8),type([1,2,3])          #objekti klase
(<class 'float'>,  <class 'list'>) #klase
>>> type(float),type(list)            #objekti metaklase
(<class 'type'>, <class 'type'>)    #metaklasa
>>> type(type)
<class 'type'>                      #metaklasa
```

Metaklasa **type**, ovisno kako se pozove:

1. stvara novi objekt tj. klasu (tzv. novi tip podatka),
2. vraća ime klase (tzv. tip podatka) objekta.

```
>>> help(type)
Help on class type in module builtins:

class type(object)
|   type(object_or_name, bases, dict)
|   type(object) -> the object's type
|   type(name, bases, dict) -> a new type
...
```

Pozivom metaklase **type** stvara se novi objekt tj. klasa (tzv. novi tip podatka).

```
Ime_klase = type("Ime_klase", bases, dict)
```

Primjer stvaranje klase **MojaKlasa** iz metaklase **type**:

```
>>> MojaKlasa=type("MojaKlasa",(),{"a":1})
```

Posebne metode klase

Posebne metode nad objektima (npr. `q` i `p`) koriste sintaksu: `q.__metoda__(p)`, tj. `p.__rmetoda__(q)`.

Operator	Metoda	Operator	Metoda
+	`__add__`	+=	`__iadd__`
*	`__mul__`	*=	`__imul__`
−	`__sub__`	-=	`__isub__`
/	`__truediv__`	/=	`__itruediv__`
//	`__floordiv__`	//=	`__ifloordiv__`
**	`__pow__`		
==	`__eq__`	!=	`__ne__`
<=	`__le__`	<	`__lt__`
>=	`__ge__`	>	`__gt__`
()	`__call__`	[ ]	`__getitem__`

Primjer:

```
>>> 10.5.__mul__(7)  # isto kao 10.5*7
73.5
```

Proizvoljno definiranje posebnih metoda.

Sve predefinirane posebne metode klase mogu se i po volji definirati.

Primjer klase za množenje objekata, **__mul__** i **__rmul__**:

```
>>> 'Ho'*(10+3j)
TypeError: can not multiply sequence by non-int of type 'complex'

""" Mnozenje kompleksnog broja i stringa definiranog kao
umnozak stringa i imaginarnog dijela kompleksnog broja """

>>> class UmnozakTipova:
        def __init__(self,tip):
            self.tip=tip
        def __mul__(self,other):     #  Tstring*(10+3j)
            return int(other.imag)*self.tip  # 'Ho'*3
        def __rmul__(self,other):     #  (10+3j)*Tstring
            return int(other.imag)*self.tip   # 'Ho'*3

>>> Tstring=UmnozakTipova("Ho")
>>> print( "Umnozak kompleksnog broja i stringa:", Tstring*(10+3j))
Umnozak kompleksnog broja i stringa: 'HoHoHo'
```

Metoda `__call__`, poziv objekta kao funkcije

Primjer funkcije:

```
>>> def funk():
        print("funk funkcija")
>>> funk()    # poziv funkcije funk
funk funkcija
```

Primjer klase:

```
>>> class KK:
        def funk(self):
            print("funk metoda")

>>> pk=KK()
>>> pk.funk()    # poziv metode funk ()
funk metoda

>>> class KK:
        def __call__(self):
            print("call metoda")

>>> funk=KK()   # objekt s imenom funk
>>> funk()      # poziv __call__ metode
call metoda
```

Metoda __repr__, reprezentacija objekta

Primjer klase bez __repr__:

```
>>> class Koktel:
        def __init__(self, sastojci):
            self.sastojci = sastojci

>>> tvoj_koktel=Koktel(['rum','sok od višnje'])
>>> tvoj_koktel      # poziv __repr__ metode
<__main__.Koktel at 0x7f6270058550>
```

Primjer klase s __repr__:

```
>>> class Koktel:
        def __init__(self, sastojci):
            self.sastojci = sastojci
        def __repr__(self):
            return f'Sok za veselje, sastav: {self.sastojci}'

>>> tvoj_koktel=Koktel(['rum','sok od višnje'])
>>> tvoj_koktel      # poziv __repr__ metode
Sok za veselje, sastav: ['rum', 'sok od višnje']
```

Metoda __str__, string objekta, print(), format()

Primjer klase s __repr__ i __str__:

```
>>> class Elektromotor:
        def __init__(self, tip, snaga):
            self.tip = tip
            self.snaga = snaga
        def __repr__(self):
            return '__repr__ za Elektromotor'
        def __str__(self):
            return '__str__ za Elektromotor'

>>> moj_EM=Elektromotor('DC','1kW')
>>> moj_EM.snaga, moj_EM.tip  #provjera
('1kW', 'DC')
>>> print(moj_EM)
__str__ za Elektromotor
>>> moj_EM
__repr__ za Elektromotor
>>> '{}'.format(moj_EM) #format izvrsava __str__ iz objekta moj_EM
'__str__ za Elektromotor'
```

Napomena: ako klasa nema definiran __str__ unutar klase, onda **print(),format()** umjesto __str__ pozivaju __repr__