Vežba 3

Na slici 1 se može videti primer procedurnog *Python* programa.

```python
def prikazi_proizvod(proizvod):
    print("\n".join([
        "",
        "{:>5}: {}".format("Šifra", proizvod["sifra"]),
        "{:>5}: {}".format("Naziv", proizvod["naziv"]),
        "{:>5}: {}".format("Cena", proizvod["cena"])
    ]))


def prikazi_proizvode(proizvodi):
    format_linije = "{:5} {:20} {:6}"

    print()
    print(format_linije.format("Šifra", "Naziv", "Cena"))
    print("{} {} {}".format("-"*5, "-"*20, "-"*6))
    for proizvod in proizvodi:
        print(format_linije.format(proizvod["sifra"], proizvod["naziv"], proizvod["cena"]))


def test():
    proizvod1 = {
        "sifra": "0001",
        "naziv": "Hleb 700g",
        "cena": 50.0,
    }
    proizvod2 = {
        "sifra": "0002",
        "naziv": "Mleko 1l",
        "cena": 80.0,
    }
    proizvod3 = {
        "sifra": "0003",
        "naziv": "Čokolada 100g",
        "cena": 120.0
    }
    prikazi_proizvod(proizvod1)
    prikazi_proizvod(proizvod2)
    prikazi_proizvod(proizvod3)

    print()
    proizvod1["cena"] = float(input("Unesite novu cenu za proizvod {}: ".format(proizvod1["naziv"])))
    print("Nova cena proizvoda {} je {}.".format(proizvod1["naziv"], proizvod1["cena"]))

    proizvodi = [
        proizvod1,
        proizvod2,
        proizvod3
    ]
    prikazi_proizvode(proizvodi)
```

Slika 1. Procedurni program

Na slici 2 se može videti primer objektnog *Python* programa sa identičnom funkcionalnošću.

```
class Proizvod:

    def __init__(self, sifra, naziv, cena):
        self.sifra = sifra
        self.naziv = naziv
        self.cena = cena

    def __str__(self):
        return "\n".join([
            "",
            "{:>5}: {}".format("Šifra", self.sifra),
            "{:>5}: {}".format("Naziv", self.naziv),
            "{:>5}: {}".format("Cena", self.cena)
        ])

    @classmethod
    def prikazi_proizvode(cls, proizvodi):
        format_linije = "{:5} {:20} {:6}"

        print()
        print(format_linije.format("Šifra", "Naziv", "Cena"))
        print("{} {} {}".format("-"*5, "-"*20, "-"*6))
        for proizvod in proizvodi:
            print(format_linije.format(proizvod.sifra, proizvod.naziv, proizvod.cena))


def test():
    proizvod1 = Proizvod("0001", "Hleb 700g", 50.0)
    proizvod2 = Proizvod("0002", "Mleko 1l", 80.0)
    proizvod3 = Proizvod("0003", "Čokolada 100g", 120.0)
    print(proizvod1)
    print(proizvod2)
    print(proizvod3)

    print()
    proizvod1.cena = float(input("Unesite novu cenu za proizvod {}: ".format(proizvod1.naziv)))
    print("Nova cena proizvoda {} je {}.".format(proizvod1.naziv, proizvod1.cena))

    proizvodi = [
        proizvod1,
        proizvod2,
        proizvod3
    ]
    Proizvod.prikazi_proizvode(proizvodi)
```

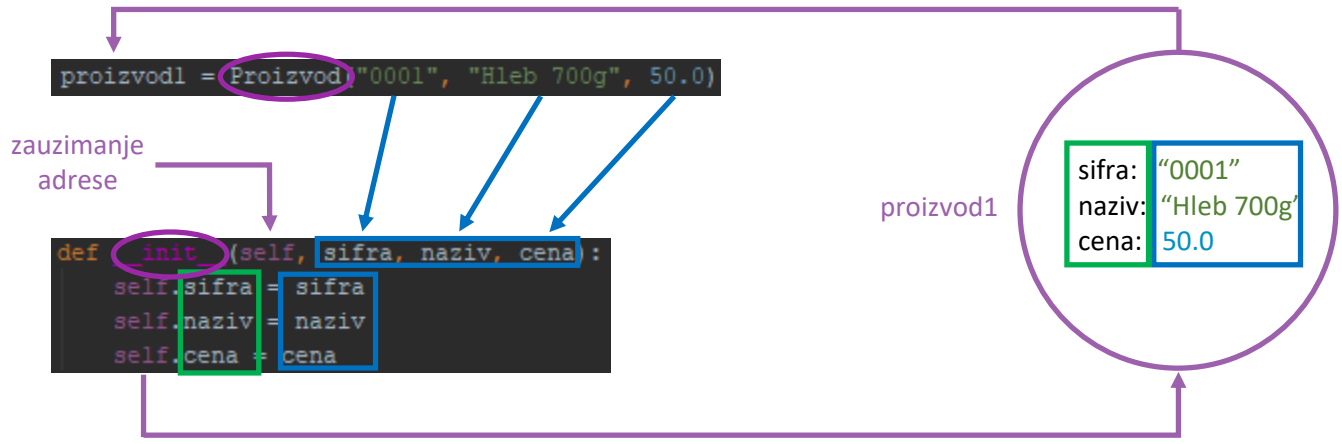
Slika 2. Objektni program

Klasa je mehanizam kojim se definiše struktura podataka zajedno sa funkcijama koje njome rukuju. Sve funkcije koje pripadaju klasi, moraju se uvući u njen blok (slika 3).

```
class Proizvod:

    def __init__(self, sifra, naziv, cena):
        self.sifra = sifra
        self.naziv = naziv
        self.cena = cena

    def __str__(self):
        return "\n".join([
            "",
            "{:>5}: {}".format("Šifra", self.sifra),
            "{:>5}: {}".format("Naziv", self.naziv),
            "{:>5}: {}".format("Cena", self.cena)
        ])

    @classmethod
    def prikazi_proizvode(cls, proizvodi):
        format_linije = "{:5} {:20} {:6}"

        print()
        print(format_linije.format("Šifra", "Naziv", "Cena"))
        print("{} {} {}".format("-"*5, "-"*20, "-"*6))
        for proizvod in proizvodi:
            print(format_linije.format(proizvod.sifra, proizvod.naziv, proizvod.cena))
```

Slika 3. Klasa

Konstruktor je funkcija posebne namene koja je zadužena za kreiranje objekta klase i inicijalizaciju njegovih atributa. Konstruktor ima rezervisan naziv ***__init__*** (slika 4). Kroz argument *self* pristupa se referenci (adresi) novokreiranog objekta i u nju se upisuju atributi. Konstruktor tipično za svaki atribut predviđa po jedan parametar. Konstruktor se poziva po nazivu klase.



Slika 4. Konstruktor

Prosleđivanjem reference na kreirani objekat *print* funkciji, prikazuje se njegova adresa (slika 5).

poziv

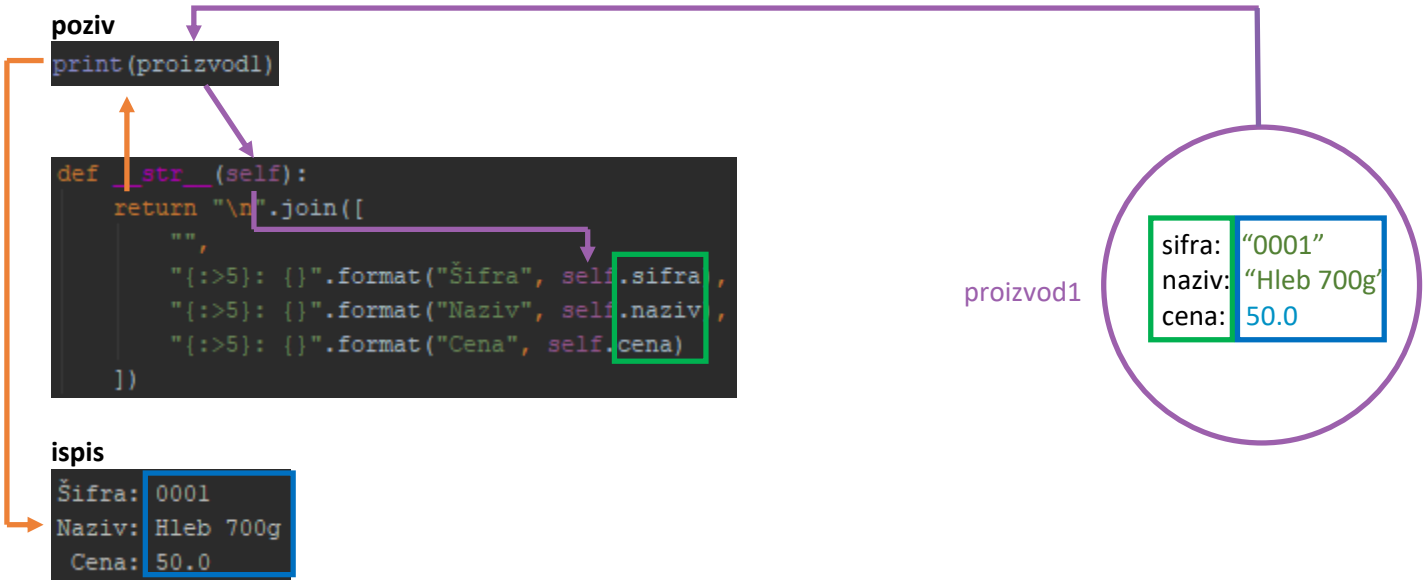
```
print(proizvod1)
```

ispis

```
<__main__.Proizvod object at 0x000000000362DE10>
```

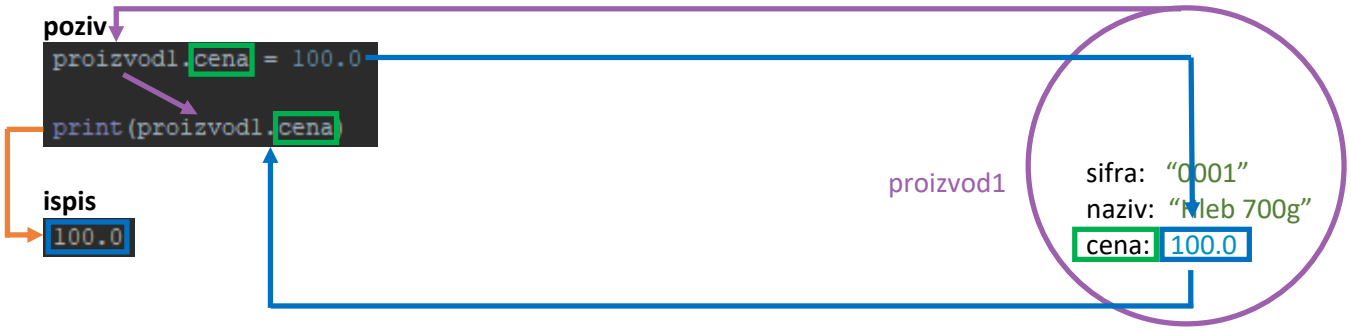
Slika 5. Adresa objekta

__str__ je funkcija posebne namene koja premošćava podrazumevani ispis adrese objekta. Biće ispisana upravo ona vrednost koju ova funkcija vrati (slika 6).



Slika 6. *__str__* funkcija

Atributi objekta se menjaju i čitaju putem reference (slika 7).



Slika 7. Pristup atributima

Ako se nazivi atributa u konstruktoru (a posledično i u ostalim funkcijama) navedu sa predvodećim znacima __, tada su atributi **privatni**, tj. zatvoreni su za pristup iz funkcija koje ne pripadaju klasi. Funkcije klase i dalje mogu da pristupe atributima po nazivu (koji sada uključuje predvodeće znake __) (slika 8).

```python
class Proizvod:

    def __init__(self, sifra, naziv, cena):
        self.__sifra = sifra
        self.__naziv = naziv
        self.__cena = cena

    def __str__(self):
        return "\n".join([
            "",
            "{:>5}: {}".format("Šifra", self.__sifra),
            "{:>5}: {}".format("Naziv", self.__naziv),
            "{:>5}: {}".format("Cena", self.__cena)
        ])

    @classmethod
    def prikazi_proizvode(cls, proizvodi):
        format_linije = "{:5} {:20} {:6}"

        print()
        print(format_linije.format("Šifra", "Naziv", "Cena"))
        print("{} {} {}".format("-"*5, "-"*20, "-"*6))
        for proizvod in proizvodi:
            print(format_linije.format(proizvod.__sifra, proizvod.__naziv, proizvod.__cena))
```

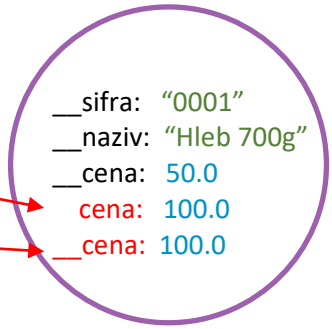
Slika 8. Privatni atributi

Nijedan od 2 zapisa sa slike 9 ne bi radio na način na koji se to očekuje, već bi nastali novi atributi.

poziv

```python
proizvod1.cena = 100.0

print(proizvod1.cena)
```

```python
proizvod1.__cena = 100.0

print(proizvod1.__cena)
```

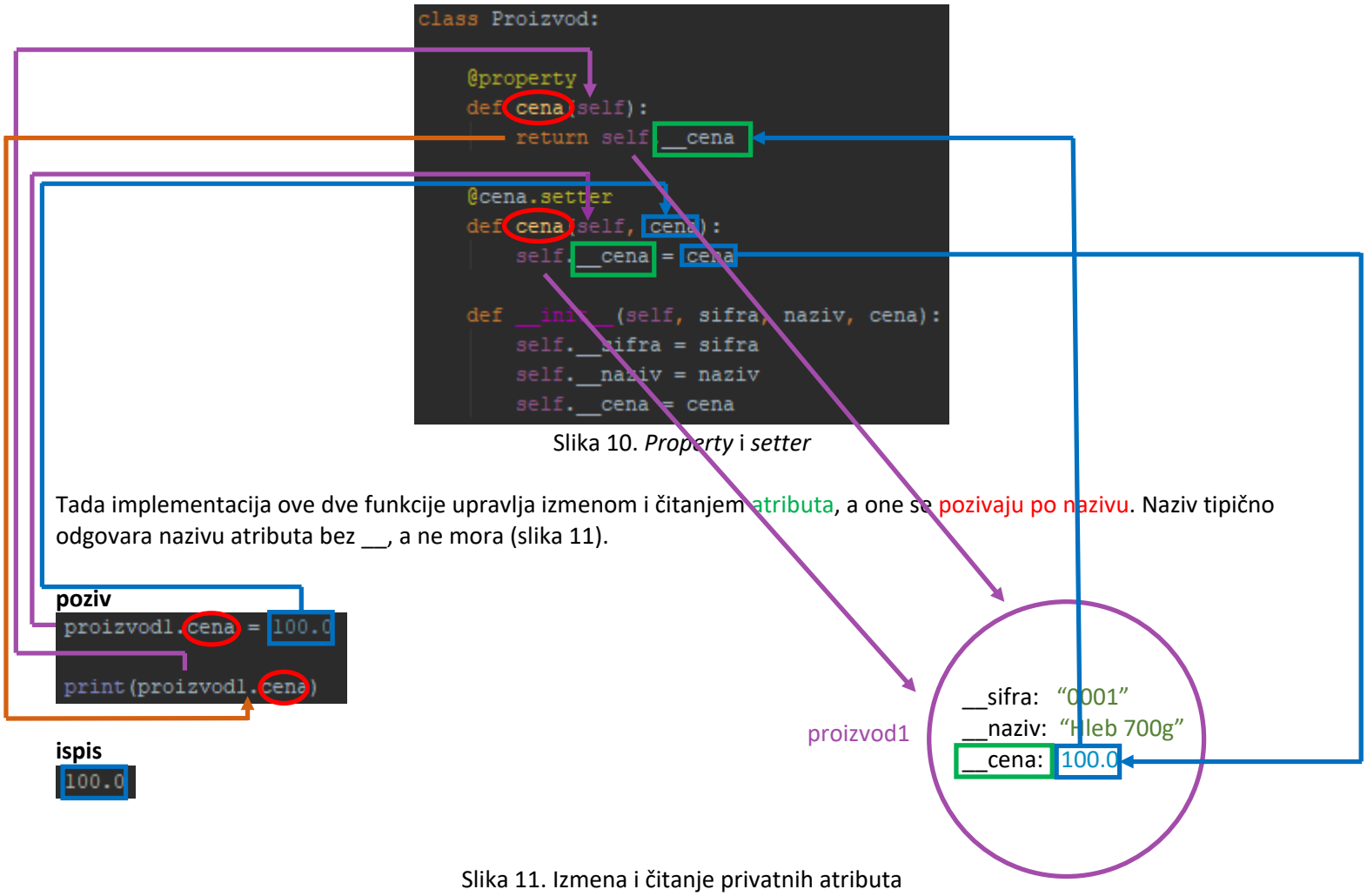


ispis

```
100.0
```

Slika 9. Pokušaj pristupa privatnim atributima

Dodavanjem ***property* (*getter*)** i ***setter*** funkcija u klasu, atributi se otvaraju za izmenu čitanje iz funkcija koje ne pripadaju klasi (slika 10).



Slika 10. *Property* i *setter*

Tada implementacija ove dve funkcije upravlja izmenom i čitanjem atributa, a one se pozivaju po nazivu. Naziv tipično odgovara nazivu atributa bez __, a ne mora (slika 11).

Slika 11. Izmena i čitanje privatnih atributa

Funkcije koje obavljaju operaciju nad jednim konkretnim objektom zovu se **metode**. Njihov prvi parametar *self* je uvek referenca na objekat. One se u opštem slučaju pozivaju korišćenjem reference na objekat. Izuzetak su konstruktor i *__str__* metode, koje se pozivaju na drugačiji način. Funkcije klase koje nisu metode se jednostavno zovu "**funkcije klase**" i anotirane su sa *@classmethod* anotacijom. Njihov prvi parametar *cls* je uvek referenca na klasu. Njihovo pozivanje ne zahteva postojanje objekta klase i pozivaju se koišćenjem naziva klase (slika 12). I funkcije klase i metode mogu direktno da pristupaju atributima klase.

metode

definicija

```
class Proizvod:

    def cena_sa_pdv(self, stopa):
        return self.__cena * (1 + stopa)
```

poziv

```
cena_pdv = proizvod1.cena_sa_pdv(0.18)
print(cena_pdv)
```

funkcije klase

definicija

```
class Proizvod:

    @classmethod
    def prikazi_proizvode(cls, proizvodi):
        format_linije = "{:5} {:20} {:6}"

        print()
        print(format_linije.format("Šifra", "Naziv", "Cena"))
        print("{} {} {}".format("-"*5, "-"*20, "-"*6))
        for proizvod in proizvodi:
            print(format_linije.format(proizvod.__sifra, proizvod.__naziv, proizvod.__cena))
```

poziv

```
Proizvod.prikazi_proizvode(proizvodi)
```

ispis

```
59.0
```

ispis

```
Šifra Naziv                Cena
----- -------------------- ------
0001  Hleb 700g             50.0
0002  Mleko 1l              80.0
0003  Čokolada 100g        120.0
```

Pogledati stranu 7!

Zadatak 1

Napisati modul *osoba_objektno.py* koji ima istu funkcionalnost kao modul *osoba_strukturalno.py*, pri čemu osobe treba da su predstavljene objektima klase *Osoba* umesto rečnicima.
Klasa osoba treba da poseduje sledeće:

- Konstruktor: ***__init__(self, jmbg, ime, prezime, god_rodjenja)***. Atribute u konstruktoru implementirati kao privatne.
- ***Properties*** i ***setter*-e** za svaki od 4 atributa: __jmbg, __ime, __prezime, __god_rodjenja.
- Metodu ***starost(self)***, koja implementira istu funkcionalnost kao funkcija *izracunaj_starost(osoba)*.
- Metodu ***__str__(self)***, koja implementira istu funkcionalnost kao funkcija *prikazi_osobu(osoba)*.
- Funkciju klase ***prikazi_osobe(cls, osobe)***, koja implementira istu funkcionalnost kao funkcija *prikazi_osobe(osobe)*.

Test funkcija koja odgovara ovakvom programu je na slici 1.

```
def test():
    osoba1 = Osoba("1111111111111", "Aaa", "Aaa", 2001)
    osoba2 = Osoba("2222222222222", "Bbb", "Bbb", 2002)
    osoba3 = Osoba("3333333333333", "Ccc", "Ccc", 2003)

    print(osoba1)
    print(osoba2)
    print(osoba3)

    print()
    osoba1.jmbg = input("Unesite novi JMBG za osobu 1: ")
    osoba1.ime = input("Unesite novo ime za osobu 1: ")
    osoba1.prezime = input("Unesite novo prezime za osobu 1: ")
    osoba1.god_rodjenja = int(input("Unesite novu godinu rođenja za osobu 1: "))
    print()
    print("Novi JMBG osobe 1 je:", osoba1.jmbg)
    print("Novo ime osobe 1 je:", osoba1.ime)
    print("Novo prezime osobe 1 je:", osoba1.prezime)
    print("Nova godina rođenja osobe 1 je:", osoba1.god_rodjenja)

    osobe = [
        osoba1,
        osoba2,
        osoba3
    ]
    Osoba.prikazi_osobe(osobe)
```

Slika 1. Test funkcija u modula *osoba_objektno.py*