1 Kreirati klasu Prva koja nema ništa u konstruktoru ali unutar ini metode kreiramo varijablu self.first koja je zapravo objekt klase Druga.

1.1 Kreirati metoda_prva unutar klase Prva koja ispisuje tekst: metoda klase prva

1.2 Klasa druga nema argumenata unutar konstruktora ali unutar init metode se kreira varijabla_objekta_druga="nekakv string druga te metoda_druga(self) koja ispisuje tekst:unutar druge metode

```
class Prva:
    def __init__(self):
        self.first = Druga()

    def metoda_prva(self):
        print("Unutar prve metode klase prva")

class Druga:
    def __init__(self):
        self.varijabla_objekta_duga = "Nekakav string druge klase argument varijabla druga"

    def metoda_druga(self):
        print("unutar druge metode")

obj1 = Prva()

print(obj1)

obj2 = obj1.first

print(obj2)

obj2.metoda_druga()

print(obj2.varijabla_objekta_duga)
```

Kreirati klase A I B. U init. U konstruktoru se ne nalazi ništa

2 # Kreirati klasu igrač, init metoda: naziv, unutar konstruktora postavljamo orude na None

2.1 # kreiramo metodu dodaj stit koja dodaje objekt stita atributu orude

2.2 # __str_metodom ispisujemo naziv igrača te broj obrambenih bodova igrača

2.3 # kreiramo klasu šti koja se sastoji od dva argumenta, naziv i broj obrambenih bodova

2.4 # broj_obrambenih_bodova je postavljen na početnu vrijednost 100

```
class Igrac:
    def __init__(self, naziv):
        self.naziv = naziv
        self.orude = None

    def dodaj_stit(self, st):
        self.orude = st

    def __str__(self):
        return (f'igrač {self.naziv} posjeduje štit s obranom: {self.orude.broj_obrambenih_bodova}')
class Stit:
    def __init__(self, naziv, broj_obrambenih_bodova=100):
        self.naziv = naziv
        self.broj_obrambenih_bodova = broj_obrambenih_bodova

st1 = Stit("drveni")
st2 = Stit("Metalni", 200)
ig1 = Igrac("perica")
ig1.dodaj_stit(st1)
print(ig1)
ig1.dodaj_stit(st2)
print(ig1)
```

3 Kreirati klasu nogometas. U konstruktoru se unosi ime a u init metodi se još definira trenutni_klub=None te placa=0.

3.1 Klasa nogometas ima metodu postavi_klub(self,nazv_kluba) gdje se definira trenutni klub igrača

3.2 Klasa **klub** u konstruktoru zatjeva naziv_kluba te rječnik popis_igraca u init metodi

3.3 Sadrži metodu dodaj_igraca koja dodaje proizvoljan broj igraca i njihove plaće, tu se i definira igračev trenutni_klub

3.4 Sadrži metodu ipsis koja ispisuje ime igrača te njegov klub

```
class Nogometas:
   def __init__(self, ime):
      self.ime = ime
      self.trenutni_klub = None
      self.placa = 0

   def postavi_klub(self, naziv_kluba):
      self.trenutni_klub = naziv_kluba

class Klub:
   def __init__(self, naziv_kluba):
      self.naziv_kluba = naziv_kluba
      self.popis_igraca = {}

   def dodaj_igraca(self, **kwargs):
      for k, v in kwargs.items():
       # eval() funkcija procjenjuje izraz, ako je valjan biti će izvršen odnosno neće se tretirati kao
string
         k = eval(k)
         k.postavi_klub(self.naziv_kluba)
         self.popis_igraca[k] = v

   def ispis(self):
      for k, v in self.popis_igraca.items():
         print(k.ime, k.trenutni_klub)

ig1 = Nogometas("kranjčar")

ig2 = Nogometas("deverić")

ig3 = Nogometas("zajec")
```

```
k1 = Klub("NK Milicioner")

k2 = Klub("BMC Botswana Meat Comission")

k2.dodaj_igraca(ig1=1000, ig2=2000, ig3=1500)

k2.ispis()
```

4 (vježba)Kreirati klase RentaCar te Klijent.

4.1 Klasa RentaCar sadrži kao argument naziv **naziv_poslovnic**e te listu **dostupni_automobili** te metode za **prikaz_raspolozivih_rentacar** koja ispisuje sve automobile koje su trenutno u poslovnici, **posudivanje_rentacar** (brisanje auta s liste) i **povratak_rentacar** (dodavanje auta u listu).

4.2 Klasa RentaCar sadrži metodu posudivanje_rentacar koja zahtjeva unos automobila koji se posuđuje te naziv objekta koji posuđuje automobil

4.3 Klasa rentcar sadrži i metodu povratak_rentacar koja zahtjeva unos automobila koji se vraća

4.4 Te klasu transfer koja omogućuje prijenos automobila iz jedne poslovnice u drugu_Poziva se sa poslovnicom koja prima automobil

4.5 Klasa **Klijent** sadrži samo argument ime koji se unosi prilikom kreiranja dokumenta, a praznu istu **popis_posudenih_utomobila** u koju se smještaju svi auti koje je određeni klijent posudio unutar init metode ali se ne ispunjava s init metodom ta lista**.**

4.6 Ispisati sve do sada poduđene automobile pojedinog klijenta

4.7 Ispisati sve posuđene automobile te koliko puta se koji automobil posudio na razini svih poslovnica

```
class RentaCar:

    def __init__(self, naziv_poslovnice, dostupni_automobili):
        self.naziv_poslovnice = naziv_poslovnice
        self.dostupni_automobili = dostupni_automobili
        self.popis_posudbi = {}

    def posudivanje_rentacar(self, auto, klijent):
        self.dostupni_automobili.remove(auto)
        klijent.popis_posudenih_automobila.append(auto)
```

```
    def povratak_rentacar(self, auto):
        self.dostupni_automobili.append(auto)

    def transfer(self, other, auto):
        self.dostupni_automobili.append(auto)
        other.dostupni_automobili.remove(auto)


class Klijent:
    def __init__(self, ime):
        self.ime = ime
        self.popis_posudenih_automobila = []

poslovnica1 = RentaCar(
    "iza ugla", ["audi1", "audi1", "audi3", "audi4", "audi5", "audi5", "audi3"])
poslovnica2 = RentaCar("iza ugla", [
                "audi1", "audi1", "audi3", "audi4", "audi5", "audi5", "audi2", "audi"])
k1 = Klijent("ivo")
k2 = Klijent("ana")
k3 = Klijent("lojtra")
poslovnica1.posudivanje_rentacar("audi1", k2)
poslovnica1.posudivanje_rentacar("audi3", k2)
poslovnica1.posudivanje_rentacar("audi3", k1)
poslovnica2.posudivanje_rentacar("audi2", k1)
print(k2.popis_posudenih_automobila)
print(poslovnica1.dostupni_automobili)
poslovnica1.transfer(poslovnica2, "audi1")
print(poslovnica1.dostupni_automobili)
```

5 kreirati klasu Hotel

5.1 # u konstruktor se unosi naziv_hotela

5.2 # unutar init metode se nalazi rječnik s key=objekt gost te broj noćenja koliko je gost ostvario

5.3 # kreiramo metodu rezervacija, unosi se objekt gost te dani koliko će noćiti gost

5.4 # kreiramo klasu **gost** u koju se unosi naziv te unutar init metode lista posjeceni_hoteli

5.5 # ispisati sve hotele koje je pojedini gost posjetio

5.6 # kreirati metodu objekta pojedinog hotela koja ispisuje imena gostiju s brojem posjeta tom i tom hotelu

5.7 # ispisati rječnik s imenima hotela s ukupnim brojem noćenja

```
class Hotel:
    svi_hoteli = {}

    def __init__(self, naziv_hotela):
        self.naziv_hotela = naziv_hotela
        self.broj_nocenja = {}

    def rezervacija(self, gost, dani):
        self.broj_nocenja[gost] = dani
        gost.posjeceni_hoteli.append(self.naziv_hotela)

        if gost not in self.broj_nocenja.keys():
            self.broj_nocenja[gost] = dani
        else:
            self.broj_nocenja[gost] += dani
        if self.naziv_hotela not in Hotel.svi_hoteli.keys():
            Hotel.svi_hoteli[self.naziv_hotela] = dani
        else:
            Hotel.svi_hoteli[self.naziv_hotela] += dani

    def ispis(self):
        for k, v in self.broj_nocenja.items():
            print(k.naziv, "-", v)

class Gost:
    def __init__(self, naziv):
        self.naziv = naziv
```

```
        self.posjeceni_hoteli = []


h1 = Hotel("vedro nebo")

h2 = Hotel("u čamcu")

h3 = Hotel("na klupi")

g1 = Gost("ivo")

g2 = Gost("ana")

g3 = Gost("pero")

h1.rezervacija(g1, 20)

h2.rezervacija(g1, 10)

h2.rezervacija(g1, 15)

h2.rezervacija(g2, 15)

# posjećeni hoteli ive

print(g1.posjeceni_hoteli)

# ostvareni broj noćenja u čamcu

print(h2.broj_nocenja)

# ispis svih hotela s noćenjima

print(Hotel.svi_hoteli)

h2.ispis()
```

6 (vježba)Kreirati klase Knjiznica te Student.

6.1 Klasa Knjiznica sadrži kao argument naziv knjiznice te listu dostupne_knjige, te metode za **prikaz_knjiga_knjiznica** koja ispisuje sve knjige koje su trenutno u knjižnici, **posudivanje_knjiznica** (brisanje knjige s liste) i **povratak_knjiznica** (dodavanje knjige u listu).

```
ok = Knjiznica("travno", ['koko u parizu', 'konan barbarian', 'hulk'])
```

6.2 Klasa **Student** sadrži samo argument ime koji se unosi prilikom kreiranja dokumenta, a praznu istu **posudene_knjige_student** u koju se smještaju sve knjige koje je određeni student posudio unutar init metode ali se ne ispunjava s init metodom ta lista**.**

6.3 Klasa student sadrži samo metode **potraznja_knjige_student** koja nema varijablu nego se unutar nje input:unesite knjigu te se poziva metoda klase knjiznica **posudivanje_knjiznica** i **povratak_knjige_student** preko koje se vraća knjiga, odnosno korisnik upisuje koju knjigu student želi vratiti, poziva se metoda klase Knjiznica **povratak_knjiznica**

6.4 Kreirati metodu **ispis_student** koja ispisuje sve knjige koje je određeni student posudio

6.5 Unutar while petlje ponuditi mogućnosti korisniku kao na slici:

```
print("""----------------------------Prikaz knjiga------------------------------
    1.prikaz svih raspoloživih knjiga
    2.potraživanje knjige
    3.povratak knjige
    4.popis knjiga koje ste pročitali
    5.izlaz
    """)
```

```
class Knjiznica:
    def __init__(self, naziv, dostupne_knjige):
        self.dostupne_knjige = dostupne_knjige
        self.naziv = naziv

    def prikaz_knjiga_knjiznica(self):
        for element in self.dostupne_knjige:
            print(element)

    def posudivanje_knjiznica(self, knjiga):
        if knjiga in self.dostupne_knjige:
            self.dostupne_knjige.remove(knjiga)
        else:
            print(f"nema te knjige u knjižnici {self.naziv}")

    def povratak_knjiznica(self, knjiga):
        self.dostupne_knjige.append(knjiga)
```

```
ok = Knjiznica("travno", ['koko u parizu', 'konan barbarian', 'hulk'])


class Student:
    def __init__(self, ime):
        self.ime = ime
        self.posudene_knjige_student = []

    def potraznja_knjige_student(self):
        self.knjiga = input("unesite naziv knjige koju želite posuditi")
        self.posudene_knjige_student.append(self.knjiga)
        return self.knjiga

    def povratak_knjige_student(self):
        self.knjiga = input("unesite naziv knjige koju želite vratiti")
        # kako se ova funkcija nalazi unutar druge metode (povratak_knjiznica) moramo vraćati
vrijednost
        return self.knjiga

    def ispis_student(self):
        for element in self.posudene_knjige_student:
            print(element)

def main():
    knjiznica = Knjiznica(['koko u parizu', 'konan', 'hulk'])
    student1 = Student("ivo")
    done = False
    while done == False:
        print("""----------------------------Prikaz knjiga-----------------------------
        1.prikaz svih raspoloživih knjiga
        2.potraživanje knjige
        3.povratak knjige
        4.popis knjiga koje ste pročitali
        5.izlaz
        """)
        izbor = input("unesite broj s liste")
        if izbor == "1":
            knjiznica.prikazi()
        elif izbor == "2":
            knjiznica.posudba(student1.potraznja_knjige())
```

```
        elif izbor == "3":
            knjiznica.povrat(student1.povratak_knjige())
        elif izbor == "4":
            print(student1.posudeno)
        elif izbor == "5":
            done == True
main()
```

7 Kreirati klasu Card - sadrži argumente boja i broj u konstruktoru te metodu **prikazi_kartu** koja ispisuje boju i broj.

7.1 Klasu Spil koja sadrži listu karata **cards** i metodu **build** kao argument u konstruktoru ali se ne unose prilikom kreiranja objekta, samo se inicijaliziraju unutar init metode (na taj način se metoda odmah pokreće prilikom stvaranja objekta deck klase Deck.

7.2 Metoda build kreira špil sa bojom i brojem. Metoda prikazi prikazuje sve karte u špilu.

["srce","kocka","tref","pik"]:

7.3 Methoda mjesanje miješa karte iz prethodno kreiranog špila - cards.

7.4 Metoda prikaži prikazuje sve karte u špilu

7.5 Metoda izvlacenjeKarte izvlači (briše) zadnju kartu iz špila.

7.6 Klasa Igrac sadrži argument ime (definira se u konstruktoru) te liste **lista_karata** koja se nalazi unutar konstruktora. Također sadrži metode **izvlacenje_1_karte** (koristeći metodu izvlacenjeKarte iz klase Spil) koja dodaje kartu u **listu_karata,**

7.7 te metodu **prikazKarte** koja prikazuje što je igrač izvukao.

7.8 Metoda **odbacivanje** simulira igračevo odbacivanje zadnje karte koju ima te dva argumenta, ime i lista_Karata.

```
import random
#u klasu odnosno objekt karta -spremamo informaciju o kiarti
class Card():
    def __init__(self,boja,broj):
        self.boja=boja
        self.broj=broj
    def prikazi_kartu(self):
        print("{} od {}". format(self.broj,self.boja))
#karta=Card("crvena",10)

#karta.prikazi_kartu()
```

```
class Spil ():
    def __init__(self):
        self.cards=[]
        #ovako se metoda build() pokreće odmah prilikom stvaranja objekta (kreira se špil karata)
        self.build()
    def build(self):
        for boja in ["srce","kocka","tref","pik"]:
            for broj in range(1,14):
                self.cards.append(Card(boja,broj))
    def prikazi(self):
        for karta in self.cards:
            karta.prikazi_kartu()
    def mijesanje(self):
        #miješamo špil
        random.shuffle(self.cards)
    def izvlacenjeKarte(self):
        return self.cards.pop()
class Igrac():
    def __init__(self,ime):
        self.ime=ime
        self.lista_karata=[]
    def izvlacenje_1_karte(self,spil):
            self.lista_karata.append(spil.izvlacenjeKarte())
            return self
    def prikaz_liste_karata(self):
            for karta in self.lista_karata:
                karta.prikazi_kartu()
    def odbacivanje(self):
            return self.lista_karata.pop()
spil=Spil()

#spil.prikazi()

#spil.mijesanje()

#spil.prikazi()

perica=Igrac("perica")

print("-------")

perica.izvlacenje_1_karte(spil).izvlacenje_1_karte(spil)
```

```
perica.prikaz_liste_karata()
```

8 Kreirati klasu Menu koja sadrži argumente name, items(items je rječnik s jelima i cijenom), start_time i end_time (vrijeme unosimo kao običan cijeli broj npr 1500). Start_time i end_time su vremena kada je meni u to cijeni

```
early_bird_items = {

 'salumeria plate': 8.00, 'salad and breadsticks (serves 2, no refills)': 14.00, 'pizza with quattro formaggi': 9.00, 'duck ragu': 17.50, 'mushroom ravioli (vegan)': 13.50, 'coffee': 1.50, 'espresso': 3.00}

early_bird_menu = Menu('Early Bird', early_bird_items, 1500, 1800)

 # Dinner Menu

dinner_items = {

 'crostini with eggplant caponata': 13.00, 'ceaser salad': 16.00, 'pizza with quattro formaggi': 11.00, 'duck ragu': 19.50, 'mushroom ravioli (vegan)': 13.50, 'coffee': 2.00, 'espresso': 3.00}

dinner_menu = Menu("Dinner", dinner_items, 1700, 2300)
```

8.1 Također treba se ispisati uz pomoć repr (specijalna funkcija za jednostavni ispis objekta) funkcije ime menija, početak i kraj .

```
print(brunch_menu)
```

Rješenje: Brunch menu available from 1100 to 1600

8.2 Trebamo kreirati metodu calculate_bill koja zbraja cijene svih jela iz menija i vraća iznos računa.

```
print(brunch_menu.calculate_bill())
```

8.3 Kreirati klasu **Restoran** koja prihvaća dva argumenta. Naziv i lista_menija od objekata klase Menu

```
menus = [early_bird_menu, dinner_menu]


r1 = Restoran("Kod Ljube", menus)

mesni = {"pajcek": 20.00, "kokoš": 15}

only_meat = Menu("mesni meni", mesni, 1800, 1900)

r2 = Restoran("Šećerna trska", [dinner_menu, only_meat])
```

8.4 kreirati klasu Fransiza koja prihvaća kao argumente naziv franišize i listu koja se sastoji od objekata klase Restoran.

8.5 Kreirati metodu ispis koja ispisuje za uneseno vrijeme koji meniji su dostupni u kojem restoranu

```
class Menu:
    def __init__(self, name, items, start_time, end_time):
        self.items = items
        self.start_time = start_time
        self.end_time = end_time
        self.name = name

    def __repr__(self):
        return f'meni:{self.name}. Od {self.start_time} do {self.end_time}'

    def calculate_bill(self):
        zb = 0
        for v in self.items.values():
            zb += v
        return zb

class Restoran:
    def __init__(self, naziv, lista_menija):
        self.naziv = naziv
        self.lista_menija = lista_menija

class Fransiza:
    def __init__(self, naziv_fransize, lista_restorana):
        self.naziv_fransize = naziv_fransize
        self.lista_restorana = lista_restorana

    def ispis(self, vrijeme):
        for restoran in self.lista_restorana:
            for menu in restoran.lista_menija:
                if vrijeme >= menu.start_time and vrijeme <= menu.end_time:
                    print(menu)
early_bird_items = {
    'salumeria plate': 8.00, 'salad and breadsticks (serves 2, no refills)': 14.00, 'pizza with quattro formaggi': 9.00, 'duck ragu': 17.50, 'mushroom ravioli (vegan)': 13.50, 'coffee': 1.50, 'espresso': 3.00}

early_bird_menu = Menu('Early Bird', early_bird_items, 1500, 1800)
```

```
# Dinner Menu

dinner_items = {

   'crostini with eggplant caponata': 13.00, 'ceaser salad': 16.00, 'pizza with quattro formaggi': 11.00,
'duck ragu': 19.50, 'mushroom ravioli (vegan)': 13.50, 'coffee': 2.00, 'espresso': 3.00}

dinner_menu = Menu("Dinner", dinner_items, 1700, 2300)

print(early_bird_menu.calculate_bill())

menus = [early_bird_menu, dinner_menu]


r1 = Restoran("Kod Ljube", menus)

mesni = {"pajcek": 20.00, "kokoš": 15}

only_meat = Menu("mesni meni", mesni, 1800, 1900)

r2 = Restoran("Šećerna trska", [dinner_menu, only_meat])

f1 = Fransiza("čevosi", [r1, r2])

f1.ispis(1600)
```

Kolokvijalno

9 (vježba)Kreirati klasu rimski koja će uneseni cijeli arapski broj pretvarati u rimski broj:

```
arapski_polje=[1000,900,500,400,100,90,50,40,10,9,5,4,1]

rimski_polje = [ "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" ]

class rimski:
    def __init__(self,num):
        self.num=num
    def arapski_u_rimski(self):
        num=self.num
        arapski_polje=[1000,900,500,400,100,90,50,40,10,9,5,4,1]
        rimski_polje = [ "M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV", "I" ]
        rimski_broj=' '
        i=0
        while num>0:
            for element in range(num//arapski_polje[i]):
                rimski_broj+=rimski_polje[i]
                num-=arapski_polje[i]
```

```
            i+=1
        return rimski_broj
broj=rimski(61)

print(broj.arapski_u_rimski())
```

R:LXI

10 (vježba)Kreirati klasu cjelobrojni koja će uneseni rimski broj pretvarati u cijeli broj:

```
rimske_vrijednosti_rjecnik = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}

class cjelobrojni:
    def __init__(self,s):
        self.s=s
    def rimske_u_arapske(self):
        s=self.s
        rimske_vrijednosti_rjecnik = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
        cijeli_broj=0
        for i in range(len(s)):
            #za slučaj da je znak manji od prethondok npr IV ili CD =400 ili CM=900
           #ide 2* taj manji broj jer ga je već unesen u prethodnom koraku pa da izbrišemo tu
         #vrijendost
            #npr.
            if i>0 and rimske_vrijednosti_rjecnik[s[i]]>rimske_vrijednosti_rjecnik[s[i-1]]:
                cijeli_broj+=rimske_vrijednosti_rjecnik[s[i]]-2*rimske_vrijednosti_rjecnik[s[i-1]]
            else:
                cijeli_broj+=rimske_vrijednosti_rjecnik[s[i]]
        return cijeli_broj
rimski1=cjelobrojni('IM')

rimski2=cjelobrojni('XCIX')

print(rimski1.rimske_u_arapske())

print(rimski2.rimske_u_arapske())
```

11 (vježba)Kreirati klasu zagrade te metodu valjano koja vraća true ili false, ovisno o tome da li je raspored zagrada valjan ili ne. U stringu se mogu nalaziti samo zagrade.

```
class zagrade:
    def __init__(self,string):
        self.string=string
```

```
    def valjano(self):
        string=self.string
        stack=[]
        pchar={"(": ")", "{": "}", "[": "]"}
        for zagrada in string:
            if zagrada in pchar: #može i pchar.keys()
                stack.append(zagrada)
            #ispitujemo da li je zagrada (koja je sada ili ] ili } ili) ista kao i ono što treba izbrisati
            elif len(stack)==0 or pchar[stack.pop()] !=zagrada:
                return False
        return len(stack)==0
znak1=zagrade("(){}[]")

znak2=zagrade("()[{)}")

znak3=zagrade("()[{}]")

print(znak1.valjano())

print(znak2.valjano())

print(znak3.valjano())
```

12 Kreirati klasu razlomak te metodu umnožak koja vraća umnožak dvaju razlomaka(instanci). Rezultat također treba biti razlomak (objekt).

```
class razlomak:
   def __init__(self,b,n):
      self.b=b
      self.n=n
   #r podrazumjeva umetnunti objekt u metodu umnozak
   def umnozak(self,r):
      b=self.b*r.b
      n=self.n*r.n
      t=razlomak(b,n)
      return t
a=razlomak(5,6)
b=razlomak(3,10)
c=a.umnozak(b)
print(c.b)
print(c.n)
```

13 prethodnom zadatku ćemo nadodati metodu skrati i umetnuti ju u metodu umnozak a i u __init__metodu kako bi se sam razlomak na početku skratio.

```
def __init__(self,b,n):
    self.b=b
    self.n=n

  #r podrazumjeva umetnunti objekt u metodu umnozak
  def umnozak(self,r):
    b=self.b*r.b
    n=self.n*r.n
    t=razlomak(b,n)
    t.skrati()
    return t
  def skrati(self):
    i=2
    b=self.b
    n=self.n
    while i<b and i<n:
      while b%i==0 and n%i==0:
        b=b//i
        n=n//i
      i+=1
    self.b=b
    self.n=n
    return

a=razlomak(30,20)
b=razlomak(3,10)
c=a.umnozak(b)
print(a.b)
print(a.n)
print(c.b)
print(c.n)
```

14 Ukoliko želimo nadodati funkciju gdje se razlomak sam skraćuje

```
  def podrezi(self):
    i=2
```

```
        b=self.b
        n=self.n
        while i<b and i<n:
            while b%i==0 and n%i==0:
                n=n//i
                b=b//i
            i+=1
        self.b=b
        self.n=n
        t=razlomak(self.b,self.n)
        return t
```

Dodajmo ugrađenu metodu za ispis __str__(self):

```
  def __str__(self):
     if self.n==1:
        return ('{}'.format(self.b))
     else:
        return ('{}/{}'.format(self.b,self.n))
```

Umjesto funkcije umnožak koristimo definiranu metodu __mul__(self,r):

```
  def __mul__(self,r):
     b=self.b*r.b
     n=self.n*r.n
     return razlomak(b,n)
```

Dodajmo metode za zbrajanje, oduzimanje i dijeljenje

```
def __add__(self,r):
        n=self.n*r.n
        b=self.b*r.n+self.n*r.b
        return razlomak(b,n)
    def __sub__(self,r):
        n=self.n*r.n
        b=self.b*r.n-self.n*r.b
        return razlomak(b,n)
    def __truediv__(self,r):
        n=self.n*r.b
        b=self.b*r.n
        return razlomak(b,n)


a=razlomak(4,5)
b=razlomak(7,6)
print(a, " a")
print(b, " b")
print(a*b, " a*b")
```

```
print(a+b," a+b")
print(a-b," a-b")
print(a/b," a/b")
```

15 Definirati klasu tocka i metodu udaljenost (self, t) koja će vraćati udaljenost između dvaju točaka. Kreirati klasu trokut čija su svojstva koordinate vrhova trokuta- objekti tipa tocka.

Distance between two points = $\sqrt{(x_B - x_A)^2 + (y_B - y_A)^2}$

```
class tocka:
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    def udaljenost(self, r):
        return ((self.x-r.x)**2+(self.y-r.y)**2)**0.5
class trokut:
    def __init__(self,a=tocka(0,0),b=tocka(0,0),c=tocka(0,0) ):
        self.a=a
        self.b=b
        self.c=c
a=tocka(0,0)
b=tocka(1,1)
print(a.udaljenost(b))
```

Kreirajmo metodu opseg koja će računati opseg trokuta unutar klase trokut

```
    def opseg(self):
        o=a.udaljenost(b)+b.udaljenost(c)+c.udaljenost(a)
        return o
a=tocka(0,0)
b=tocka(1,1)
c=tocka(1,0)
print(a.udaljenost(b))
t=trokut(a,b,c)
print(t.opseg())
```

16 Kreirati klasu vrijeme čija svojstva su sati i minute. Ispisivati će se u obliku 03:15 što znači da se ispisuje nula (vrijeme se unosi kao broj) u slučaju da je u pitanju jedna znamenka.

```
class vrijeme:
```

```
    def __init__(self,h,m):
        self.h=h
        self.m=m
    def __str__(self):
        h=str(self.h)
        m=str(self.m)
        if len(h)<2:
            h="0"+h
        if len(m)<2:
            m="0"+m
        return ('{}:{}'.format(h,m))
a=vrijeme(12,0)
print(a)
```

Nad kreiranom klasom vrijeme definirati relacijske operacije >,< i ==. Upisati dva objekta te ispisati da li je prvo vrijeme veće ili manje ili jednako drugom vremenu.

```
def __lt__(self,r):
        return self.h*60+self.m<r.h*60+r.m
    def __eq__(self,r):
        return self.h*60+self.m==r.h*60+r.m
    def __rt__(self,r):
        return self.h*60+self.m>r.h*60+r.m
a=vrijeme(12,34)
b=vrijeme(12,34)
print(a)
print(b)
if a>b:
    print('veće')
elif a==b:
    print('jednako')
else:
    print('manje')
```

Omogućiti unos broja minuta te dodati metodu __add__ koja će dodati uneseni broj u minutama te ih pribrojiti objektu

```
   def __add__(self,m):
        ukupno=self.h*60+self.m+m
        return vrijeme(ukupno//60,ukupno%60)
a=vrijeme(12,34)
b=vrijeme(12,34)
m=int(input())
```

```
print(a)
print(b)
if a>b:
   print('veće')
elif a==b:
   print('jednako')
else:
   print('manje')
print(a+m)
```

Dodati metodu __sub__ koja će računati razliku između dva unesena vremena

```
   def __sub__(self,r):
      v= abs(self.h*60+self.m - (r.h*60+r.m))
      return vrijeme(v//60,v%60)
print(a-b)
```

17 Metode klase -klasa zaposlenik

```
class Zaposlenik:
    broj_zaposlenika=0
    povisica=1.04
    def __init__ (self,first,last,pay):
        self.first=first
        self.last=last
        self.pay=pay
        Zaposlenik.broj_zaposlenika+=1
    def primjena_povisice(self):
        self.pay=int(self.pay*self.povisica)
        return self.pay

    @classmethod
  #prvi argument je klasa a ne instanca
    def postaviti_postotak_povišice(cls,iznos):
       cls.povisica=iznos
    #metodu klasa koristimo kada trebamo inicijalizirati instance iz stringa npr.
    @classmethod
    def od_stringa(cls,zaposlenik_string):
        first,last,pay=zaposlenik_string.split("-")
        #ova linija kreira novog zaposlenika (mogli smo staviti Zaposlenik umjesto cls
        return cls(first,last,pay)
```

```
zap1=Zaposlenik('ivica','marica',1000)
zap2=Zaposlenik('ana','marija',2000)

Zaposlenik.postaviti_postotak_povišice(1.5)
print(Zaposlenik.povisica)
print(zap1.povisica)
print(zap2.povisica)
#mogli smo i ovako samo tada ne možemo kontrolirati unos (getter setter)
Zaposlenik.povisica=2
print(Zaposlenik.povisica)
print(zap1.povisica)
print(zap2.povisica)
#--------------------------------
#metoda klase za kreiranje ovih objekata
zaposlenik_string1='Pero-perić-2100'
zaposlenik_string2='anja-anjić-2020'

novi_zaposlenik1=Zaposlenik.od_stringa(zaposlenik_string1)
print(novi_zaposlenik1.pay)
```