1 Property -Kreirati klasu Knjizevnik te argumente ime, prezime te puno_ime (koje nije u konstruktoru) ispisati puno_ime objekta potom promijeniti ime te ponovno ispisati puno_ime objekta

1.1 Prebrojiti koliko je objekata kreirano

```
class Knjizevnik:
    br = 0

    def __init__(self, ime, prezime):
        self.ime = ime
        self.prezime = prezime
        self.puno_ime = self.ime, self.prezime
        Knjizevnik.br += 1

k1 = Knjizevnik("ivo", "andrič")

k2 = Knjizevnik("Lav", "tolstoj")


print(k1.puno_ime)
# ukoliko je argument unesen preko init metode , ono što je nastalo u init metodi se ne mijenja
k1.ime = "andrija"
k2.prezime = "dostojevski"
print(k1.puno_ime, "----------")
print(k2.puno_ime, "----------")
print(k1.ime, "----------")
print(k2.prezime, "----------")
print(Knjizevnik.br)
print(k1.br)
print(k1.br)
```

1.2 Kreiramo metodu puno_ime() i izbrišemo property(argument) puno_ime te uz pomoć @property dekoratora omogućimo pozivanje funkcije kao argumenta (propertya)

```
class Knjizevnik:
    def __init__(self, ime, prezime):
        self.ime = ime
        self.prezime = prezime

    @property
```

```
    def puno_ime(self):
        return self.ime, self.prezime
```

2 Kreirati klasu zaposlenik koja omogućuje objektima unos imena, prezimena i prihoda. Omogućiti ispis emaila koji ima oblik: ime.prezime@vern.hr. Kreirati metodu koja omogućuje ispis emaila te uklanja hrvatske dijakritičke znakove i zamjenjuje ih, ovisno o situacji, u c,s,z Također omogućiti prebrojavanje kreiranih instanci te ispis ukupne plaće svih zaposlenika.

```
class Zaposlenik:
    broj_zaposlenika=0
    placa=0
    def __init__(self,first,last,pay):
        self.first=first
        self.last=last
        self.pay=pay
        #odnosi se na klasu a ne na instancu zato ne self
        Zaposlenik.broj_zaposlenika+=1
        Zaposlenik.placa+=self.pay
    def email(self):
        self.first=self.first.replace('š','s')
        self.last=self.last.replace('č','c')
        self.email=self.first.lower()+"."+self.last.lower()+"@vern.hr"
        return '{}'.format(self.email)
radnik1=Zaposlenik('Siniša','Jovčić',1000)
radnik2=Zaposlenik('Maja','Watz',1000)
print(radnik1.first)
#metoda email je prazna funkcija
print(radnik1.email())
print(Zaposlenik.email(radnik1))
print(Zaposlenik.broj_zaposlenika)
print(Zaposlenik.placa)
```

3 __del__game-kada se objekt ne koristi automatski se poziva __del__ ukoliko smo definirali metodu u klasi i briše se objekt nakon zadnje upotrebe -PORUKA SE JAVLJA NA KRAJU PROGRAMA

3.1 Kreiramo klasu Game koja u konstruktoru nema ništa

```
class Game:
    player = 11

    def __init__(self):
        print("ja igram")

    def players(self):
        print(f'trenutno imamo {self.player} igrača')
    # ako iza definici9je instance nema više ništa (nogomet=Game() objekts se automatski briš)
#izvršava se na kraju programa
    def __del__(self):
        print("objekt je izbrisan")
nogomet = Game()

kosarka = Game()

nogomet.players()
```

4 Kreirati klasu Krug s osnovnim s osnovnim atributima i metodama (opseg, polumjer, površina). Također omogućiti ispisivanje broja kreiranih instanci i omogućiti brisanje kako instance objekta tako (objekta __del__)

```
class Krug:
    broj_krugova = 0

    def __init__(self, r=0):
        self.r = r
        Krug.broj_krugova += 1

    def opseg(self):
        return 2*self.r*3.14

    def povrsina(self):
```

```
        return self.r**2*3.14

    def __del__(self):
        Krug.broj_krugova -= 1
        print(f"objekt sa radijusom {self.r} je izbrisan")

k1 = Krug(20)
k2 = Krug(21)
k3 = Krug(22)
print(Krug.broj_krugova)
del k2
print(Krug.broj_krugova)
print(k1.r)
print(k3.r)
print(k2.r)
```

5 Kreirati klasu automobil sa podacima: naziv automobila(ime), broj vrata (broj_vrata) te listom sa svim raspoloživim jačinama motora (jacine_motora).

5.1 Kreirati metodu ispisi koja ispisuje sve raspoložive jačine motora za zadani objekt.

5.2 Koliko je ukupno uneseno jačina motora (podesiti sve i u svim ostalim metodama)

5.3 Definirati metode dodaj koja dodaje novu jačinu motora u jacine_motora, te metodu izbrisi koja briše određenu jačinu motora a ako se jačina ne nalazi u listi neka se ispiše “nema vrijednosti”.

5.4 Definirati metodu rjecnik te ju pozvati kao varijablu objekta (ne kao metodu, nego kao argument property). Metoda ispisuje sve varijable zadanog objekta i njihove vrijednosti kao rječnik (__dict__).

```
class Automobil:
    broj_jacina_m = 0

    def __init__(self, ime, broj_vrata, jacine_motora):
        self.ime = ime
        self.broj_vrata = broj_vrata
        self.jacine_motora = jacine_motora
        Automobil.broj_jacina_m += len(jacine_motora)
```

```
    def ispisi(self):
        for element in self.jacine_motora:
            print(element)

    def dodaj(self, element):
        self.jacine_motora.append(element)
        Automobil.broj_jacina_m += 1

    def brisi(self, element):
        self.jacine_motora.remove(element)
        Automobil.broj_jacina_m -= 1

    @property
    def rjecnik(self):
        print(self.__dict__)

a1 = Automobil('a1', 3, [1000, 1200, 1400])

a2 = Automobil('BMW', 5, [1000, 1100, 1300])

a1.dodaj(499)

a1.ispisi()

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

a1.brisi(1200)

a1.brisi(1400)

print(Automobil.broj_jacina_m, "ukupan broj jačina motora")

a1.ispisi()

a1.rjecnik
```

6 Kreirati klasu Firma s tri argumenta: odjel, primanja, naziv_osobe.

6.1 Pod odjel je moguće unijeti samo ili kontroling ili pribava ali to ne testiramo

6.2 kreirati metodu objekta **zarada** koja će nadodati dohodak u argument **primanja**.

6.3 Također kreirati metodu **trosak** koja će oduzeti određeni iznos iz **primanja**.

6.4 Kreirati metodu objekta **utrosak_odjela** koja za svaki dodatak ili utrošak (metode __init__, zarada, trosak) računati ukupnu zaradu svih odjela i zaradu pojedinih odjela

6.5 Omogućiti brisanje objekta tako da odgovaraju i ukupna primanja svih odjela

```
class Firma:
    Ukupno_svi = 0
    Ukupno_kontroling = 0
    Ukupno_pribava = 0

    def __init__(self, odjel, primanja, naziv_osobe):
        self.odjel = odjel
        self.primanja = primanja
        self.naziv_osobe = naziv_osobe
        self.zarada_odjela(primanja)

    def zarada(self, novac):
        self.primanja += novac
        self.zarada_odjela(novac)

    def trosak(self, novac):
        self.primanja -= novac
        self.utrosak_odjela(novac)

    def zarada_odjela(self, novac):
        Firma.Ukupno_svi += novac
        if self.odjel == "kontroling":
            Firma.Ukupno_kontroling += novac
        if self.odjel == "pribava":
            Firma.Ukupno_pribava += novac

    def utrosak_odjela(self, novac):
        Firma.Ukupno_svi -= novac
```

```
        if self.odjel == "kontroling":
            Firma.Ukupno_kontroling -= novac
        elif self.odjel == "pribava":
            Firma.Ukupno_pribava -= novac

    def __del__(self):
        self.utrosak_odjela(self.primanja)

d1 = Firma("kontroling", 1000, "ivica")

d2 = Firma("kontroling", 1000, "marica")

d3 = Firma("pribava", 1000, "Linđo")

d1.zarada(1000)

d2.zarada(1000)

d2.trosak(500)

print(Firma.Ukupno_kontroling)

print(Firma.Ukupno_pribava)

print(Firma.Ukupno_svi)

del d2

print("-----")

print(Firma.Ukupno_kontroling)

print(Firma.Ukupno_pribava)

print(Firma.Ukupno_svi)
```

7 Kreirati klasu Automobili koja omogućava unos argumenta objekta naziv_kompanije, te proizvoljan niz parova :tip-cijena kao na primjeru

7.1 omogućiti korisniku da upiše cijena u metodu objekta. Pretraga, a da se ispiše tip automobila koji je ili jednak ili manji od cijene. Pretraga je metoda klase znači možemo provjeravati cijene u svim objektima, trebaju se ispisati svi nazivi tipova automobile koji koštaju jednako ili manje od unesene cijene.

7.2 Kreirati metodu objekta koja briše tip automobila (automatski i cijenu/vrijednost tog automobila)

```
class Automobili:
    svi_auti = {}

    def __init__(self, naziv, **tipovi):
        self.tipovi = tipovi
        self.naziv = naziv
        Automobili.svi_auti.update(tipovi)

    @classmethod
    def pretraga(cls, cijena_trazenog):
        for k, v in Automobili.svi_auti.items():
            if v <= cijena_trazenog:
                print(k)

    def brisanje(self, tip):
        self.tipovi.pop(tip)
        Automobili.svi_auti.pop(tip)


o1 = Automobili("audi", a1=30000, a2=50000, a3=55000)

o2 = Automobili("Bmw", z1=50000, z2=60000, z3=700000)

print(Automobili.svi_auti)

Automobili.pretraga(50000)

o1.brisanje("a2")

print(Automobili.svi_auti)

print(o1.tipovi)
```

8 Kreirati klasu Farma te u init metodi kreirati argumente **naziv_farme** i unijeti rječnik **kolicine** kao na predlošku.

8.1 Kreirati rječnik klase koji sadrži ukupan broj pojedinih životinja

8.2 Kreirati metode dodaj i briši koje dodaju pojedinu životinju pojedinom objektu

8.3 Kreirati metodu koja računa ukupnu cijenu pojedine farme(metoda objekta)

8.4 Kreirati metodu klase vrijednost koja računa ukupnu vrijednost životinja. U svim farmama. Cjenik je zadan:

cjenik = {"konj": 1000, "kokoš": 10, "patka": 30, "pajcek": 400,"krava": 500, "koza": 300, "ovca": 600, "zeko": 40}

```
class Farma:
    cjenik = {"konj": 1000, "kokoš": 10, "patka": 30, "pajcek": 400,
              "krava": 500, "koza": 300, "ovca": 600, "zeko": 40}
    rj = {}

    def __init__(self, naziv_farme, **kolicine):
        self.naziv_farme = naziv_farme
        self.kolicine = kolicine
        self.ispuna_rjecnika()

    def ispuna_rjecnika(self):
        for k, v in self.kolicine.items():
            if k not in Farma.rj.keys():
                Farma.rj[k] = v
            else:
                Farma.rj[k] += v

    def dodaj(self, k, v):
        d = {}
        d[k] = v
        self.kolicine.update(d)
        if k in Farma.rj.keys():
            Farma.rj[k] += v
        else:
            Farma.rj[k] = v

    def brisi(self, key):
        value = self.kolicine[key]
        self.kolicine.pop(key)
```

```
        if Farma.rj[key] > value:
            Farma.rj[key] -= value
        else:
            Farma.rj.pop(key)

    def ukupna_vrijednost_objekta(self):
        zb_objekta = 0
        for k, v in Farma.cjenik.items():
            if k in self.kolicine.keys():
                zb_objekta += Farma.rj[k]*Farma.cjenik[k]
        return zb_objekta

    @classmethod
    def ukupna_vrijednost(cls):
        zb_ukupno = 0

        for k, v in Farma.cjenik.items():
            if k in Farma.rj.keys():
                zb_ukupno += Farma.rj[k]*Farma.cjenik[k]
        return zb_ukupno
f1 = Farma("veseli pajcek", pajcek=5, konj=2, kokoš=14)

f2 = Farma("otrovi kod ", pajcek=6, konj=3, zeko=6)

f1.dodaj("puran", 2)

print(f1.kolicine)

print(f2.kolicine)

print(Farma.rj)


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

f1.brisi("konj")

print(f1.kolicine)

print(Farma.rj)

print(f1.ukupna_vrijednost_objekta())

print(Farma.ukupna_vrijednost())
```

9 Class method.Kreirati klasu dostava s dva argumenta, kilometraza i grad u koji se dostavlja. Svaka dostava se tretira kao objekt.

9.1 Ispisati koliko smo ukupno prešli kilometara te kreirati property za klasu koja računa koliko puta smo dostavljali u koji grad (posjeceni_gradovi je rječnik, key=grad, kilometraza=value).

9.2 Kreirati (classmethod) metodu klase koja ispisuje rječnik (u kojemu se nalaze gradovi i broj posjeta pojedinom gradu) (petlja)

9.3 Kreirati metodu klase **nasi**. Koristiti zadanu listu (ne kao argument metode) nasi_gradovi te ispisati samo one gradove s liste koje su naši dostavljači posjetili, a da se nalaze unutar te liste.. (uputa: unutar metode klase kreirati polje te u njega smjestiti posjećene gradove iz već kreiranog rječnika (keys) te potom sve polja pretvoriti u skup)

```
class Dostava:
    ukupno_prijedeno = 0
    posjeceni_gradovi = {}
    nasi_gradovi = ["ši", "ri", "zg", "st"]

    def __init__(self, kilometraza, grad):
        self.kilometraza = kilometraza
        self.grad = grad
        Dostava.ukupno_prijedeno += kilometraza
        if self.grad in Dostava.posjeceni_gradovi.keys():
            Dostava.posjeceni_gradovi[self.grad] += 1
        else:
            Dostava.posjeceni_gradovi[self.grad] = 1

    @classmethod
    def ispis(cls):
        for k, v in Dostava.posjeceni_gradovi.items():
            print(k, v)

    @classmethod
    def nasi(cls):
        polje = []
        for k in cls.posjeceni_gradovi.keys():
            polje.append(k)
        print(set(polje) & set(cls.posjeceni_gradovi))
```

```
d1 = Dostava(100, "ri")

d2 = Dostava(110, "ri")

d3 = Dostava(120, "ri")

d4 = Dostava(50, "zg")

d5 = Dostava(30, "os")

d1 = Dostava(310, "ši")

print(Dostava.posjeceni_gradovi)

Dostava.ispis()

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

Dostava.nasi()

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


print(Dostava.posjeceni_gradovi['zg'])

print(Dostava.ukupno_prijedeno)

Dostava.ispis()
```

9.4 (vježba) U slučaju da želimo proširiti prethodni zadatak sa ispisom kilometraže po pojedinom gradu osim broja posjeta npr. rijeku smo posjetili 3 puta i prešli 330km: ri [3, 330]

```
if self.grad in Dostava.posjeceni_gradovi.keys():
    Dostava.posjeceni_gradovi[self.grad][0] += 1
    Dostava.posjeceni_gradovi[self.grad][1] += kilometraza
else:
    Dostava.posjeceni_gradovi[self.grad] = [1]
    Dostava.posjeceni_gradovi[self.grad] += [kilometraza]
```

```
ri  -  [3, 330]
zg  -  [1, 50]
os  -  [1, 30]
ši  -  [1, 310]
```

10 Kreirati klasu Student uz pomoć init metode. Unutar konstruktora definirati: (fname,lname, reg).

10.1 Također definirati argument objekta (unos se ne zahtjeva ali se varijabla nalazi u init metodi) self.izborni_studenta = ['Obavezan'] (sadrži odmah obavezan predmet u polju). To znači da svaki student ima obavezan kolegij

10.2 Kreirati također dva argumenta klase smjer = "IT" te svi mogući kolegiji koje jedan student može upisati

izborni_kolegiji = ['Algoritmi', 'Multimedija','Python', 'Machine Learning', 'Java', 'PHP', 'JavaScript' ,'Obavezan']

10.3 Kreirati metodu registracija_izbornog koja omogućuje studentu unos više izbornih predmeta koji su ponuđeni unutar izborni_kolegiji (*izborni_predmeti). Sve unesene izborne predmete unosimo u self.izborni_studenta[]

10.4 Kreirati metodu klase **neizabrani_kolegiji_studenata** koja omogućuje ispis svih kolegija koje niti jedan student nije upisao (kreirati argument klase prazno polje naziva neizabrani_kolegiji gdje smještamo neizabrane izborne)

10.5 Kreirati metodu klase **ispis_svih_studenata** koja ispisuje imena i prezimena svih upisanih studenata i broj izbornih (obavezni preskočiti) npr.

```
class Student:
    smjer = "IT"
    izborni_kolegiji = ['Algoritmi', 'Multimedija', 'Python',
                        'Machine Learning', 'Java', 'PHP', 'JavaScript', 'Obavezan']
    neizabrani_kolegiji = []
    izabrani_kolegiji = []
    popis_svih_studenata = {}

    def __init__(self, fname, lname, reg):
        self.izborni_studenta = ['Obavezan']
        self.fname = fname
        self.lname = lname
        self.reg = reg
        self.ime_prezime = self.fname+" " + self.lname
        Student.popis_svih_studenata[self.ime_prezime] = 0

    def registracija_izbornog(self, *kwargs):
        for element in kwargs:
            self.izborni_studenta.append(element)
            Student.izabrani_kolegiji.append(element)
        Student.popis_svih_studenata[self.ime_prezime] += len(kwargs)
```

```
    @classmethod
    def neizabrani_kolegiji_studenata(cls):
        skup_neizabranih = set(Student.izborni_kolegiji) - \
            set(Student.izabrani_kolegiji)

        for element in skup_neizabranih:
            print(element)

    @classmethod
    def ispis_svih_studenata(cls):
        for k, v in Student.popis_svih_studenata.items():
            print(k, v)

stud1 = Student('ivo', 'pero', 2)

stud2 = Student("pero", "perić", 3)

stud3 = Student("maja", "majić", 4)

stud1.registracija_izbornog("Multimedija", "Python")

stud2.registracija_izbornog("Multimedija", "Python", "Machine Learning")

stud3.registracija_izbornog("Algoritmi")

print(stud1.izborni_studenta)

print("**********************")


print("izabrani")

print(Student.izabrani_kolegiji)

print("izborni")

print(Student.izborni_kolegiji)

print("neizabrani")

Student.neizabrani_kolegiji_studenata()

Student.ispis_svih_studenata()
```

JavaScript
Java

Obavezan
PHP
ivo pero 2
pero perić 3
maja majić 1

11 radimo u kafiću "posljednja šansa" gdje svaki gost naručuje piće s liste.

11.1 # kreiramo klasu Gost, kreira se argument klase cjenik koji je tipa rječnik(dictionary) kojem su key,values=naziv_pica,cijena_pica

```
cjenik = {"pivo": 18, "vino": 25, "kava": 13, "juice": 17, "čaj": 13}
```

11.2 # argument objekta je ime (init metoda)

11.3 # te izvan konstruktora varijablu objekta **racun** (dictionary->key,values=naziv_pica,kolicina) gdje se bilježi Koliko čega je popio pojedini klijent(object)

11.4 # izvan konstruktora kreiramo varijablu objetka klijent_ukupno (argument objekta) gdje se računa koliko je pojedini klijent ukupno platio

11.5 Kreirati metodu klijent_racun koja unosi stavke (nazive pića I količinu) u obliku ** koja računa Koliko je pojedini klijent platio

11.6 #unutar metode klijent_racun također se ispisuje Gost.ukupno (argument klase) gdje se ispisuje koliko su svi gosti ukupno platili

```
class Gost:
    cjenik = {"pivo": 18, "vino": 25, "kava": 13, "juice": 17, "čaj": 13}

    ukupno = 0


    def __init__(self, ime):
        self.ime = ime
        self.racun = {}
        self.klijent_ukupno = 0


    def klijent_racun(self, **unos_racuna):
        for key, value in unos_racuna.items():
            self.racun[key] = value
            if key in Gost.cjenik.keys():
                self.klijent_ukupno += Gost.cjenik[key]*value
        Gost.ukupno += self.klijent_ukupno
ivo = Gost("ivo")
```

```
ivo.klijent_racun(pivo=2, vino=2)

print(ivo.racun)

print(ivo.klijent_ukupno)

ana = Gost("ana")

ana.klijent_racun(kava=1)

print(Gost.ukupno)
```

12 Kolokvijalno LIFO-Private metoda- kreirati stack klasu te argument objekta stack_list pretoviriti u private . Pokušati izračunati duljinu bez „__“ te potom dodati „__stack_list=[]. Također kreirati metodu koja ispisuje listu (drugi dio zadatka

```
class Stack:
   def __init__(self):
      self.stack_list=[]
stack_object=Stack()
print(len(stack_object.stack_list))
```

Izmijeniti self__stack_list=[] i uočiti kako je sada argument zaštićen (enkapsulacija

```
class Stack:
   def __init__(self):
      self.__stack_list = []
   def push(self, val):
      self.__stack_list.append(val)
   def pop(self):
      val = self.__stack_list[-1]
      del self.__stack_list[-1]
      return val
   def ispis(self):
      print(self.__stack_list)
stack_object=Stack()

#print(len(__stack_object.stack_list))



stack_object.push(3)

stack_object.push(2)
```

```
stack_object.push(1)


print(stack_object.pop())

print(stack_object.pop())

stack_object.ispis()
```

13 Kolokvijalno- queue First In, First Out (FIFO

```
class Queue:
   def __init__(self):
      self.stack_list=[]
   def push(self,val):
      self.stack_list.append(val)
   def brisanje(self):
      value=self.stack_list[0]
      self.stack_list=self.stack_list[1:]
      return value
def ispis(self):
      print(self.__stack_list)


stack_objeckt=Queue()
stack_objeckt.push(33)
for i in range (10):
   stack_objeckt.push(i)

print(len(stack_objeckt.stack_list))

stack_objeckt.brisanje()
print(stack_objeckt.brisanje())

print(len(stack_objeckt.stack_list))
```

13.1 kreirajmo methodu klase koja ispisuje sve objetke(studente) koji nisu u niti jednoj grupi

```
   @classmethod
   def withoutGroupMembers(cls):
      return (list(filter(lambda s: s.groupMember == None, Student.all_students)))
```

```
std3 = Student("Kocka")
std4 = Student("Brid")
print(Student.withoutGroupMembers())
```

14 (vježba)Kreirati klase Knjiznica te Student.

14.1 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'])
```

14.2 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.

14.3 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**

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

```
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()
```

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

15.1 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.

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

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

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

15.4 Metoda prikaži prikazuje sve karte u špilu

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

15.6 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,**

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

15.8 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()
```

16 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).

16.1 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

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

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

16.3 Kreirati argument klase meniji kao praznu listu koju potom punimo s instancama kreiranih objekata.

```
Menu.Meniji.append(self)
```

16.4 kreirati klasu Franchise koja prihvaća kao argumente adresu i listu koja se sastoji od objekata klase Menu.

```
print("2.----------")
#early bird 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)
```

```
menus = [brunch_menu, early_bird_menu, dinner_menu]

fransiza =Franchise("Ilica 0", menus)
```

16.5 Ispisuje adresu uz pomoć __repr__ .

```
print(fransiza)
```

16.6 Potrebno je kreirati metodu available_menus koja ispisuje koji meniji su dostupni s obzirom na vrijeme (vrijeme unosimo u funkciju kao argument) kada se poslužuju.

```
print(fransiza.available_menus(1700))
```

rješenje: [Early Bird menu available from 1500 to 1800, Dinner menu available from 1700 to 2300]

16.7 Kreirati klasu Business koja ima argument ime i franšizu od kojih se sastoji objekt klase Business

```
class Menu():
    meniji=[]
    def __init__(self,name,items,start_time,end_time):
        self.name=name
        self.items=items
        self.start_time=start_time
        self.end_time=end_time
        Menu.meniji.append(self)
    def __repr__(self):
       return self.name + ' menu available from ' + str(self.start_time) + ' to ' + str(self.end_time)
    def calculate_bill(self):
                bill=0
                for purchased_item in self.items:
                    bill+=self.items[purchased_item]
                return bill
print("1.----------------------------------------------------")
print(brunch_menu)
print(brunch_menu.calculate_bill())
 #u objekt klase franchise možemo unijeti proizvoljan broj menija (lista)
class Franchise:
    def __init__(self,address,menus):
        self.address=address
        self.menus=menus
    def __repr__(self):
```

```
        return self.address
    def available_menus(self,time):
          #unutar polja menus smještamo dostupne menije
        dostupni_meniji=[]
        for menu in self.menus:
            if time>=menu.start_time and time <=menu.end_time:
                dostupni_meniji.append(menu)
        return dostupni_meniji

print(fransiza.available_menus(1700))

#provjera ispisa objekata klase Menu
for menu in Menu.meniji:
   print(menu)
class Business:
    def __init__(self,name,franchises):
        self.name=name
        self.franchises=franchises



# Kids Menu

kids_items = {

  'chicken nuggets': 6.50, 'fusilli with wild mushrooms': 12.00, 'apple juice': 3.00

}

kids_menu = Menu("Kids", kids_items, 1100, 2100)

#možemo i sami kreirati svoje menije…

menus = [brunch_menu, early_bird_menu, dinner_menu, kids_menu]

flagship_store = Franchise("1232 West End Road", menus)

new_installment = Franchise("12 East Mulberry Street", menus)

basta = Business("Basta Fazoolin' with my Heart", [flagship_store, new_installment])

# Arepa

arepas_items = {

  'arepa pabellon': 7.00, 'pernil arepa': 8.50, 'guayanes arepa': 8.00, 'jamon arepa': 7.50
```

```
}

arepas_menu = Menu("Take a’ Arepa", arepas_items, 1000, 2000)


arepas_place = Franchise("189 Fitzgerald Avenue", [arepas_menu])


arepas_business = Business("Take a' Arepa", [arepas_place, fransiza])

print("3.----------")

print(arepas_business.franchises[0].menus[0])

print(arepas_business.franchises[1].menus[0])

print(arepas_business.franchises[1].menus[1])

print(arepas_business.franchises[1].menus[2])
```

Kolokvijalno

17 (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

18 (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())
```

19 (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())
```

20 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)
```

21 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)
```

22 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)
```

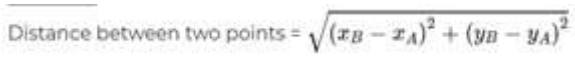
Dodajmo metode za zbrajanje, oduzimanje i dijeljenje

```
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")
```

23 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.




```
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())
```

24 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)
```

25 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)
```