



Distance Learning System



Klase i objekti

Object Oriented programming in Python

Objektno Orjentisano programiranje

- Objektno orjentisano programiranje je programiranje koje omogućava programerima da razvijaju programe, kreirajući objekte i relacije između tih objekata.

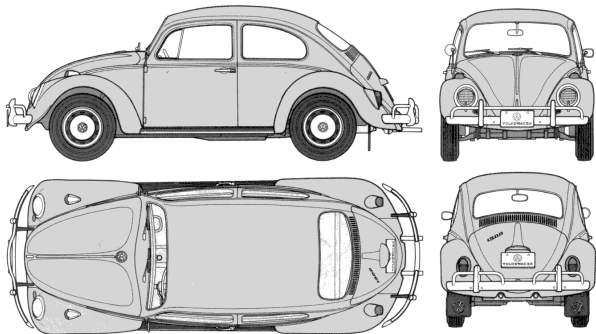
Python u svetu OOP-a

- Python ima punu objektnu podršku, ali je takođe moguće koristiti ga i u proceduralnom maniru
- Jezici koji mogu da se koriste u objektnom kontekstu ili bez njega, nazivaju se **hibridnim jezicima**
 - Poznatiji hibridni jezici su: C++, php, ...
 - Poznatiji potpuno objektno orjentisani jezici su: C#, Visual Basic, Java
- Karakteristike objektnih jezika su: klase, objekti, metode, svojstva

Klase

- Klasa predstavlja šablon po kome se proizvode objekti koji su još poznati i kao instanca klase.
- Klase generalizuju entitete realnog sveta, dok objekti predstavljaju specifične manifestacije ovih entiteta.
- Na klasu se može misliti kao na modlicu (kalup) po kojoj se prave kolačići.

Klasa BUBA



Objekti klase BUBA




Klase

- Objekt ili instanca jeste jedan primerak nečega, napravljen prema određenom uzorku (klasi). Svaka ptica jeste jedan objekt (instanci) pomenute klase ptica, dok je svaki čovek, objekt klase čovek.
- Da li je ptica instanca klase avion? Verovatno ne. Ali i ptica i avion mogu biti instanca klase: leteći objekt

Kreiranje klasa i objekata

- Prvi korak u procesu kreiranja klasa i objekata jeste kreiranje klase.
- Iako je u nekim jezicima moguće kreirati objekat bez prethodnog kreiranja klase, u Python-u to nije moguće
- Da bi u Python-u kreirali klasu, kreiramo novi fajl (koji se zove kao i klasa koju hoćemo da kreiramo), u njemu pišemo:

```
class Cookie:  
    pass
```



Ključna reč pass
predstavlja samo placeholder

Imenovanje klasa

- Kod imenovanja klasa poštujemo ista pravila kao i kod imenovanja promenljivih, pri čemu postoji još jedno "nepisano" pravilo: Nazive klasa pišemo početnim velikim slovom ukoliko je to moguće, a ako naziv ima više reči, onda svaki početak reči takođe.
- Na primer: `class Person`, `class MyClass`, `class SomeOtherClass`, `class Car`...

```
class Person:  
    pass
```

```
class MyClass:  
    pass
```

```
class SomeOtherClass:  
    pass
```

```
class Car:  
    pass
```

Kreiranje klasa i objekata

- Kada kreiramo klasu, možemo je koristiti za kreiranje objekata

```
cookie = Cookie()
```

- *Napisana linija znači: napravi jedan objekat po šablonu **Cookie** i smesti ga u promenljivu koja se zove **cookie***

Kreiranje klasa i objekata

- Klasa koja je kreirana u prethodnom primeru nema naročit smisao jer, iako se zove Cookie, ne sadrži nikakve informacije o bilo kakvim kolačima. Štaviše, ona ne sadrži nikakve informacije.

```
class Person:  
    name = ""  
    surname = ""  
    age = 0  
    height = 0  
    weight = 0
```

- Klasa Person sadrži neke informacije. Tačnije, ona sadrži polja u koja možemo smestiti informacije a u kojima su trenutno podrazumevane vrednosti. Ime, prezime, godište, visina i težina su karakteristike kojima se može opisati jedan čovek

Kreiranje klasa i objekata

- Sledećim kodom bi mogli instancirati jednu osobu po šablonu Person:

```
person = Person()  
person.name = "Peter"  
person.surname = "Jackson"  
person.age = 50  
person.height = 176  
person.weight = 80
```

- Ako ovako nešto uradimo u kodu, ništa se neće dogoditi na izlazu, ali će program znati da postoji promenljiva (objekat) sa nazivom person, koja sadrži podatke koji karakterišu jednog čoveka.

Kreiranje klase i objekata

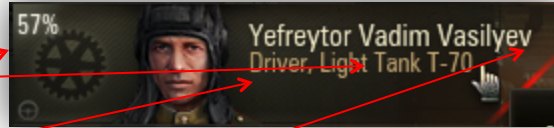
- Iako Klase / objekti oslikavaju objekte iz realnog sveta, oni ih ne moraju mapirati u potpunosti, već treba da odgovaraju potrebama sistema
- Na primer, za osobu sa slike, mogli bi kreirati klasu Person, ali za nju nam ne bi bila važna težina i visina, dok bi nam sa druge strane, bilo važno iskustvo



Kreiranje klasa i objekata

- Pogledajmo kako bi izgledala implementacija ove klase

```
class TankCrewPerson:  
    vehicle_id = 0  
    experience = 0  
    category = 0  
    name_and_surname = ""
```



- Kreiranje objekta bi moglo izgledati ovako:

```
tc_person = TankCrewPerson()  
tc_person.experience = 57  
tc_person.category = 1  
tc_person.name_and_surname = "Yefreytor Vadim Vasilyev"  
tc_person.vehicle_id = 5
```

Kreiranje klasa i objekata

- Pored toga što imaju karakteristike, ljudi mogu nešto i da rade
- Mogućnost objekata da nešto rade, ostvaruje se pomoću metoda
- Metode su takođe sastavni delovi objekta, ali se malo drugačije ponašaju i označavaju od običnih podataka

```
class Person:
    name = ""
    surname = ""
    age = 0
    height = 0
    weight = 0
    def sleep(self): pass
    def run(self): pass
```

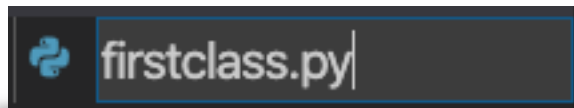
} Polja klase

} Metode klase

Prva Python klasa

(oopp-ex01 firstclass)

- Kreirati novi modul (fajl) pod nazivom firstclass.py



- Unutar fajla kreirati klasu Car

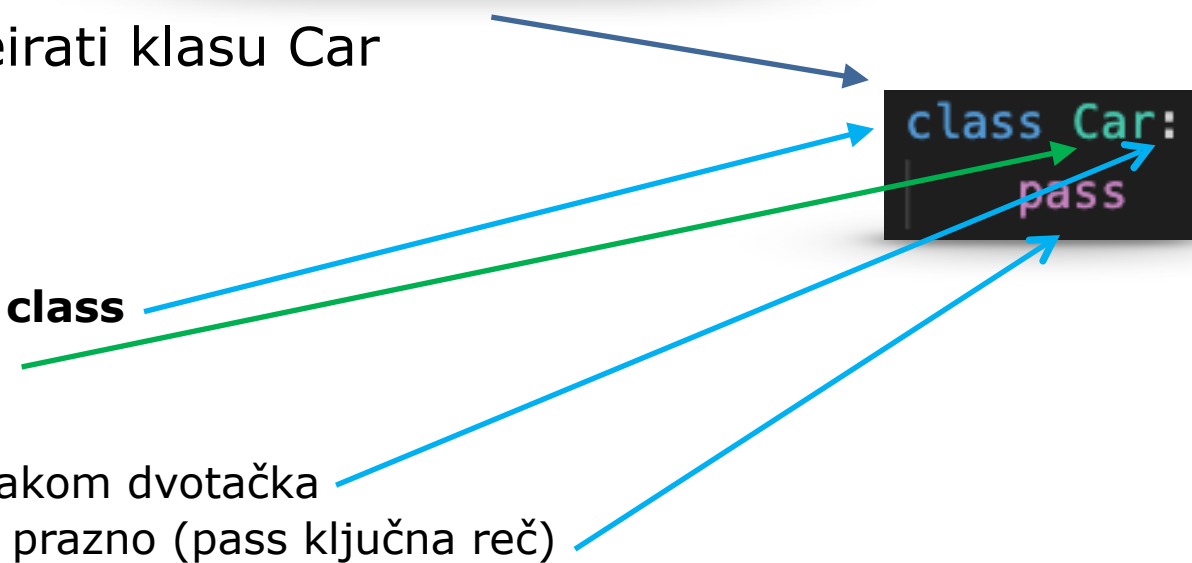
- Definicija klase

- Zaglavlje

- Ključna reč **class**
 - Naziv klase

- Telo

- Počinje oznakom dvotačka
 - Trenutno je prazno (pass ključna reč)

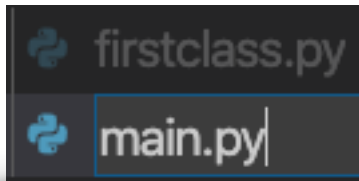


```
class Car:  
    pass
```

Prva Python klasa

(oopp-ex01 firstclass)

- Unutar istog direktorijuma, kreirati još jedan fajl (main.py)



- Unutar fajla izvršiti instanciranje i štampanje objekta na izlazu

```
import firstclass  
car = firstclass.Car()  
print(car)
```

- Na izlazu će biti prikazan naziv modula i klase, kao i hash kod objekta

```
<firstclass.Car object at 0x10b525588>
```

Članovi klase

- Dele se na **instancne** i **klasne (statičke)**
 -  Vezani za objekte klase
 -  Vezani za klasu
- Članovi su:
 - **Polja**
 - **Metode**
 - **Druge klase**

Modifikatori pristupa

- Modifikatori pristupa predstavljaju opseg dostupnosti određenih članova klase
- Modifikatori pristupa su:
 - **Privatni (private)** - članovi su vidljivi samo u okviru klase u kojoj se nalaze
 - **Zaštićeni (protected)** - elementima se može pristupiti iz same klase gde su definisani i iz izvedenih klasa
 - **Javni (public)** - bilo koji objekat ima pristup nad elementom

Polja klase

- Polja su zapravo promenljive koje se nalaze u okviru klase
- Dele se na **instancna** i **statička**

```
class Car:  
    make = ""  
    model = ""  
    numDoors = 0  
    wheels = 4
```

Instancno

```
car = Car()  
car.make = "BMW"
```

Statičko

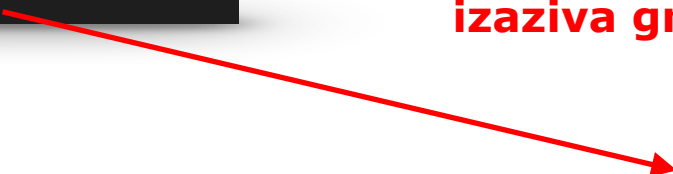
```
print(Car.wheels)
```

Private modifikator pristupa

- Na prethodnom primeru ne postoje modifikatori pristupa, i zato su polja javna
- Ako bi postavili privatni modifikator na neko od polja, ono ne bi bilo dostupno izvan klase (privatni modifikator se označava sa dve donje crte isred naziva polja):

```
class Car:  
    def __init__(self):  
        self.__make = "BMW"
```

**Pokušaj pristupanja
polju izvan klase
izaziva grešku**



```
car = Car()  
print(car.__make)
```

Metode

- Pored polja, među članovima klase nalaze se i **metode**
- Metode nisu ništa drugo do funkcije koje se nalaze unutar klasa
- Obzirom da je Java poptuno objektni jezik, funkcija kao pojam i ne postoji, već samo metoda
- Metod koji se aktivira automatski prilikom kreiranja klase, naziva se **konstruktor**
- Unutar konstruktora, možemo postaviti neke inicijalne vrednosti instanciranom objektu

Konstruktor (oopp-ex01 constructor.py)

- Konstruktor se automatski aktivira prilikom instanciranja klase

```
1 class Car:  
    make = ""  
    model = ""  
    numDoors = 0  
    wheels = 4  
    def __init__(self):  
        print("New car is being created")
```

```
2 car = Car()
```

```
3 New car is being created
```

Statičke metode

- Statički metodi se definišu kao i drugi metodi, ali se pozivaju direktno putem klase, a ne preko njene instance

```
class Car:
    make = ""
    model = ""
    numDoors = 0
    wheels = 4

    @staticmethod
    def HowManyWheels():
        print(Car.wheels)
```

- Mogu, ali ne moraju biti označeni dekoratorom **@staticmethod**
- Iz statičke metode nije moguće doći do instance

Poziv

`Car.HowManyWheels()`

Instancne metode

- Karakteristika instancnih metoda je da postoje u kontekstu objekta. To znači da su im dostupni svi članovi objekta klase u kojoj se nalaze
- Ove metode ne treba koristiti bez prethodno kreiranog objekta
- Da bi metoda bila instancna, ona mora imati jedan parametar u koji će, prilikom poziva, automatski biti prosleđena referenca na objekat nad kojim je metoda pozvana (obično se imenuje **self**)

```
class Car:  
    model = ""  
    def set_model(self):  
        self.model = "I5"
```

Isti podatak

```
car = Car()  
car.set_model()  
print(car.model)
```

I5

Instancne metode (oopp-ex01 methods1)

```
def printDetails(self):  
    print("Make: " + self.make)  
    print("Model: " + self.model)  
    print("Number of doors: " + self.numDoors)
```

Metoda sa leve strane ima naziv printDetails i služi za prikaz vrednosti polja kreiranog automobila:

Da bi je aktivirali, kucamo:

```
car = Car()  
car.printDetails()
```

Izlaz programa je:

```
Make:  
Model:  
Number of doors:
```

Pokušajte da modifikujete program tako da prikaže smislene rezultate

Parametrizacija metoda

- Jedna od ključnih osobina metoda je mogućnost prihvatanja i vraćanja parametara
- Parametri koji ulaze u metod, nazivaju se **ulazni parametri**
- Parametri koje metod vraća, nazivaju se **izlazni parametri** (u pitanju je zapravo samo jedan izlazni parametar)
- Prilikom kreiranja metode, moramo u njegovom potpisu naznačiti koje će parametre vratiti i koje će prihvatiti
- Metod ne mora vratiti ništa, niti prihvatiti ništa, ali onog trenutka kada ga kreiramo, u obavezi smo da poštujemo njegovu definiciju (njegov potpis)
- Za vraćanje vrednosti iz metode, koristimo ključnu reč **return** kojoj sledi jedna ili više vrednosti

Parametrizacija metoda (oopp-ex01 methodparams.py)

- Jedan parametrizovan instancni metod, može izgledati ovako:

Referenca na tekući objekat

Lista ulaznih parametara

Slanje na izlaz

```
class Calc:
    def add(self, a, b):
        result = a + b
        return result
```

- Metod kasnije možemo pozvati na sledeći način:

```
calc = Calc()
res = calc.add(2,3)
print(res)
```

Vrednosti i reference

- Obzirom da radi isključivo sa objektima, sve promenljive u Python-u su zapravo reference
- Ponašanje ovih promenljivih i njihovih vrednosti, zavisi od toga da li su objekti mutabilni ili nemutabilni
 - Složeni tipovi (instance klasa, liste, setovi, rečnici...) najčešće su mutabilni
 - Prosti tipovi (brojevi, boolean, stringovi...) su nemutabilni

Prosleđivanje nemutabilnih i mutabilnih parametara

- Ako postoje sledeće dve funkcije

```
def f1(p):  
    p = 20  
def f2(p):  
    p[1] = 10
```

- I obratimo mu se na sledeći način

```
x = 10  
y = [1,2,3]  
f1(x)  
f2(y)
```

- Šta možemo da očekujemo kao rezultat?

```
print(x)  
print(y)
```

- Pokušajte da samostalno uradite primer i vidite rezultat

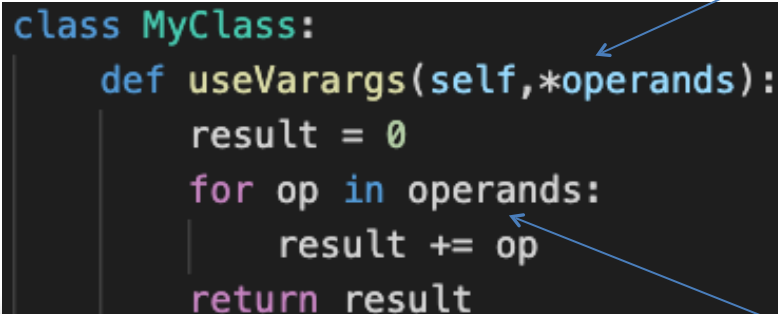
Proizvoljan broj argumenata

- U nekim slučajevima, hoćemo da metod sadrži proizvoljan broj parametara
- Ovo je moguće uraditi na više načina
 - Korišćenjem **varargs** parametrizacije
 - Korišćenjem **podrazumevanih** parametara

Prosleđivanje proizvoljnog broja parametara

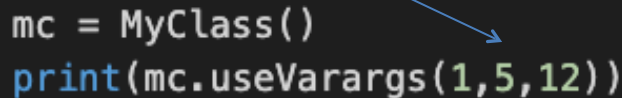
- Da bi prosledili proizvoljan broj parametara metodi, parametre metode moramo označiti ulazni parametar na specifičan način

```
class MyClass:  
    def useVarargs(self,*operands):  
        result = 0  
        for op in operands:  
            result += op  
        return result
```

A diagram with two blue arrows. One arrow points from the asterisk in the method signature `*operands` to the list of arguments `(1, 5, 12)` in the function call `mc.useVarargs(1, 5, 12)`. The second arrow points from the `operands` parameter in the method signature to the `operands` variable in the `for` loop `for op in operands:`.

- Prilikom poziva, umesto jednog parametra, prosleđujemo ni jedan ili više.
- U samoj metodi, parametre ćemo dobiti u formi **niza**

```
mc = MyClass()  
print(mc.useVarargs(1,5,12))
```

A diagram with a blue arrow pointing from the list of arguments `(1, 5, 12)` in the function call `mc.useVarargs(1, 5, 12)` to the asterisk in the method signature `*operands` from the previous code block.

Vežba 1 – Kalkulator (oopp-ex01 calculator.py)

- Potrebno je kreirati klasu kalkulator koja ima dva svojstva: operand1 i operand2.
- Klasa poseduje metode:
 - **add**, koja kao rezultat vraća zbir dva operanda
 - **sub**, koja kao rezultat vraća razliku dva operanda
 - **mul**, koja kao rezultat vraća proizvod dva operanda
 - **div**, koja kao rezultat vraća količnik dva operanda

Vežba 2 – Dogfight (oopp-ex01 dogfight.py)

- Potrebno je kreirati klasu Airplane, sa poljima: name, health i ammo i metodom shoot
- Metod shoot kao parametar prihvata objekat klase airplane
- U metodi shoot, po slučajnom izboru, oduzima se health jednom avionu i ammo drugom avionu (health duplo više nego ammo)
- Izvršavati funkcionalnost, sve dok jednom od aviona health ili ammo ne padne ispod nule

round	name	health/ammo	:	name	health/ammo

1	Ilyushin	86/96		Spitfire	92/93
2	Ilyushin	70/94		Spitfire	88/85
3	Ilyushin	68/93		Spitfire	86/84
4	Ilyushin	54/90		Spitfire	80/77
5	Ilyushin	40/87		Spitfire	74/70
6	Ilyushin	22/83		Spitfire	66/61
7	Ilyushin	12/82		Spitfire	64/56
8	Ilyushin	-2/78		Spitfire	56/49

Vežba 3 – Bankomat (oopp-ex01 bankomat.py)

- Kreirati klase Bankomat i Card
- Klasa Card treba da ima polje za naziv i stanje
- Klasa Bankomat treba da ima polje card (objekat klase Card) i metode set_card i withdraw
- Metod set_card postavlja karticu u bankomat
- Metod withdraw, skida određenu sumu sa kartice
- Ukoliko ne postoji dovoljno sredstava ili kartica nije u bankomatu, metod withdraw prijavljuje grešku

```
Success, new balance: 2
```

Vežba 4 – Soccer (oopp-ex01 soccer.py)

- Kreirati klase Team, Score i Match
- Klasa Team sadži polje, naziv tima
- Klasa Score, sadži polja: home i away, u kojima se nalazi rezultat meča
- Klasa Match sadrži polja: home (objekat klase Team), away (objekat klase Team), minute (trenutni minut utakmice) i score (trenutni rezultat)
- Klasa Match sadrži metod start, kojim započinje utakmica
- Tokom trajanja utakmice, menjaju se rezultati, po slučajnom izboru i prikazuju na izlazu. Utakmica se završava kada broj minuta dođe do 90

```
Manchester United vs Chelsea  
5 - 3 (73 minute)
```



```
Manchester United vs Chelsea  
8 - 4 (90 minute)  
Match ended
```

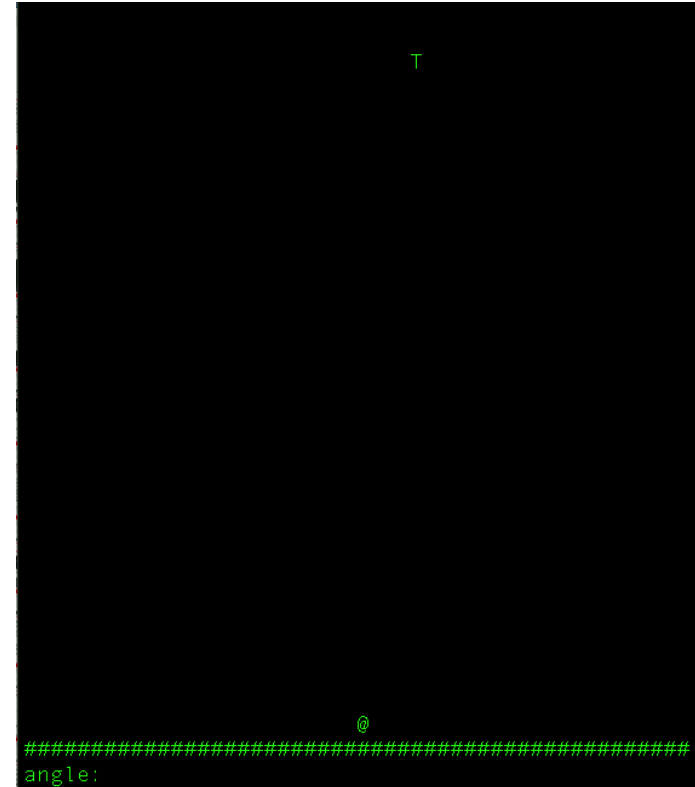
Vežba 5 – Bingo (oopp-ex01 bingo.py)

- Kreirati aplikaciju bingo
- Aplikacija prihvata imena korisnika i brojeve sve dok postoji unos
- Aplikacija zamišlja pet od N brojeva i prikazuje ih na izlazu
- Aplikacija prikazuje koji su korisnici pogodili koje brojeve (ukoliko postoje pogoci)
- Ukoliko neki korisnik pogodi sve brojeve, na izlazu se ispisuje poruka BINGO

```
Round: 2
Entering ticket:
Your name: Peter
Numbers (x,y,z..): 1,3,5,7,9
Entering ticket:
Your name: Sally
Numbers (x,y,z..): 5,7,9,10,11
Entering ticket:
Your name:
Numbers (x,y,z..):
No tickets entered
Starting draw...
6 1 2 8 10
User Peter hit numbers: [1]
User Sally hit numbers: [10]
```

Vežba 6 – Shooter (oopp-ex01 shooter.py)

- Računar crta metu na ekranu
- Korisnik unosi ugao
- Projektil se kreće pod uglom koji je korisnik uneo
- Ukoliko projektil pogodi metu, ispisuje se poruka pogotka
- Ukoliko projektil izađe iz ekrana, ispisuje se poruka promašaja



Zadatak – Library

- Napraviti program, biblioteka
- Program na početku prikazuje meni sa nekoliko stavki: 1 nova knjiga, 2 izmena knjige, 3 brisanje knjige, 4 listanje svih knjiga, 5 izlaz iz programa
- Sve knjige se čuvaju u memoriji (sadržaj se briše kada program prekine sa radom)

Vezba – Pathfinding

- Napraviti dve tačke u 2D prostoru
- Napraviti program koji će prikazivati putovanje od jedne tačke do druge

Vezba – Pathfinding 1 (oopp-ex01 pathfinding.py)

- Napraviti dve tačke u prostoru
- Postaviti prepreku u prostoru
- Napraviti program koji će prikazivati putovanje od jedne tačke do druge

```
0
0#0
00# 0
00# 0
00# 0
00# 0
5000000000# 0
#
#
#
#
#
```