UNIVERZITET METROPOLITAN BEOGRAD

CS324 - SKRIPTING JEZICI

Python i OOP

Lekcija 05

PRIRUČNIK ZA STUDENTE

CS324 - SKRIPTING JEZICI

Lekcija 05

PYTHON I OOP

- Python i OOP
- Poglavlje 1: Razlika između OOP i proceduralnog programiranja
- Poglavlje 2: Osnove OOP-a u Python 3.x jeziku
- Poglavlje 3: Klasa i objekti
- Poglavlje 4: Rad sa klasama i instacama
- Poglavlje 5: Klase naspram modula
- Poglavlje 6: Pokazna vežba #5
- Poglavlje 7: Individualna Vežba #5
- Poglavlje 8: Domaći zadatak #5
- Zaključak

UVOD

Uvod u lekciju #5

U petoj lekciji prelazi se na paradigmu objektno-orjentisanog programiranja u Python 3.x jeziku.

OOP je paradigma koja se bazira na konceptu objekata. Objekti sadrže podatke, u formi polja, odnosno atributa, dok se sam kod, odnosno procedure, nazivaju metodi.

Za razliku od proceduralnog koje prenosi podatke od poziva procedura do poziva procedura dok se ne dobije željeni rezultat, objektno-orjentisano programiranje enkapsulira podatke i ponašanje podataka u objekte.

Većina ovih osnovnih koncepata je verovatno već poznato iz ranijih predmeta, ali u ovoj lekciji biće reči o nekim specifičnostima koje izdvajaju Python.

U Python 3.x jeziku postoji opseg imenskih prostora, (en. namespace), koji se koristi prilikom definisanja klasa. U ovom kontekstu, imenski prostor predstavlja mapiranje imena na objekte.

Ono što je specifično za Python jeste da ukoliko ne postoji izjava koje ja globalna ili ne-lokalna, imenovanje uvek ide u opsegu koji je najmanji (en. innermost scope).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 1

Razlika između OOP i proceduralnog programiranja

RAZLIKE IZMEĐU OOP I PROC. PROGRAMIRANJA

Programi napisani u OOP jezicima jesu projektovani tako da su sastavljeni od objekata i međusobne interakcije tih objekata.

Pitanje:

U koju paradigmu spadaju proceduralno i objektno-orjentisano programiranje?

Koje su glavne osobine te paradigme?

Objektno-orjentisano programiranje (en. object.-oriented programming, OOP) jeste paradigma programiranja koja se zasniva na konceptu objekta.

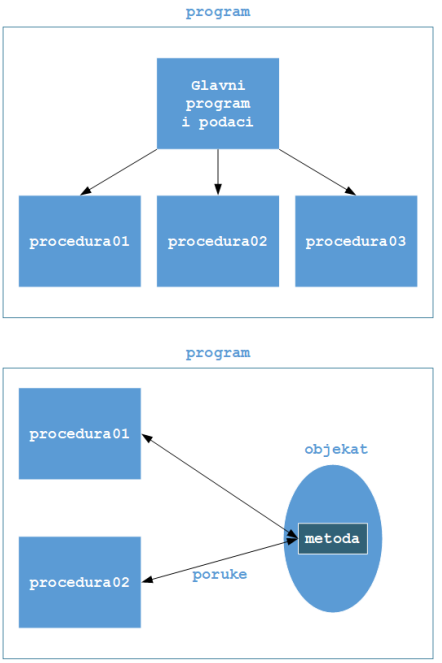
Objekat (en. object) predstavlja osnovni gradivni element ove paradigme. Objekat može sadržati *podatke* (u obliku *polja*, tj. *atributa* ili *osobina*) i *kod* (u obliku *procedura* tj. *metoda*).

Programi napisani u OOP jezicima jesu projektovani tako da su sastavljeni od objekata i međusobne interakcije tih objekata.

Glavna razlika između paradigmi proceduralnog programiranja i OOP-a jeste ta da se kod proceduralnog programiranja program sastoji od poziva procedura dok se ne dobije krajnji rezultat. Kod OOP-a objekti i njihova interakcija slanjem poruka čine sastav programa.

Podsetnik:

Na slici je vizuelno opisana razlika između programa napisanog na proceduralni način, i programa napisanog kroz OOP.



Slika 1.1 Proceduralno programiranje naspram OOP. [Izvor: Autor.]

PRIMER PISANJA PROGRAMA U PROC. I OOP PARADIGMI

Proceduralno programiranje nema fleksibilnost koje pruža OOP pristup.

Primer:

Napisati proceduralni i OOP program koji će računati obim i površinu pravougaonika na osnovu unetih stranica.

Proceduralno programiranje

```
# Pravougaonik
a = 30
b = 40
obim = 2 * (a + b)
povrsina = a * b

print(f"Pravougaonik stranica {a} i {b} ima obim {obim} i površinu {povrsina}.")
```

Objektno-orjentisano programiranje

```
# Klasa pravougaonik
class pravougaonik:
    def __init__(self, a, b):
        self.a = a
        self.b = b
    def obim(self):
        return 2 * (self.a + self.b)
```

```
    def povrsina(self):
        return self.a * self.b

p = pravougaonik(30, 40)
print(f"Pravougaonik stranica {p.a} i {p.b} ima obim {p.obim()} i površinu
{p.povrsina()}.")
```

Pitanje:

Proširenje zadatka:

Uneti 10 pravougaonika sa različitim dužinama stranica i izračunati im površinu i obim. Kako bi se ovaj zadatak proširio u proceduralnom, a kako u OOP pristupu?

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

VIDEO O RAZLIKAMA IZMEĐU DVE PARADIGME IMPERATIVNOG PROGRAMIRANJA

U nastavku je video lekcija koja objašnjava razlike između proceduralnog i OOP pristupa programiranju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 2

Osnove OOP-a u Python 3.x jeziku

OSNOVNI KONCEPTI OOP-A

Python je i proceduralni i OOP jezik.

Do sada viđeno da je Python 3.x jezik koji podržava proceduralno programiranje, a u prošlom primeru je pokazano da podržava i OOP. Python 3.x kao programski jezik stoga podržava više pristupa programiranju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Objektno-orjentisano programiranje se najbolje može opisati kroz četiri glavna koncepta:

- Enkapsulacija (en. encapsulation)
- Apstrakcija (en. abstraction)
- Nasleđivanje (en. inheritance)
- Polimorfizam (en. polymorphism)

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

ENKAPSULACIJA I APSTRACKIJA PODATAKA

Enkapsulacija i apstrakcija podataka štite kod unutar klasa.

Enkapsulacija

Različiti objekti unutar jednog programa u opštem slučaju moći će da komuniciraju jedan sa drugim. Ukoliko programer želi da zaustavi objekte da međusobno interaguju, onda treba da se ti objekti enkapsuliraju u izdvojene klase.

Kroz proces enkapsulacije, klase ne mogu da promene ili da interaguju sa specifičnim promenljivama i funkcijama objekta.

Analogno kapsuli koja zadržava sadržaj unutar ljuske, princip enkapsulacije pravi zaštitnu barijeru oko informacije koja je razdvaja od ostatka koda.

Programeri na taj način mogu da repliciraju objekat kroz različite delove programa ili ka drugim programima.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Apstrakcija

Apstrakcija (podataka) se može smatrati kao proširenje enkapsulacije zbog toga što sakriva određene osobine i metode ostatku koda u programu, da bi na taj način interfejs ka objektu bio pojednostavljen.

Apstrakcija podataka je korisna prilikom pisanja programa iz više razloga ali prvenstveno pomaže izolaciji uticaja promene koda ukoliko se jave problemi unutar objekta, to će uticati samo na promenljive unutar, a ne na kod glavnog programa.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

NASLEĐIVANJE I POLIMORFIZAM

Nasleđivanje i polimorfizam jesu koncepti OOP-a koji smanjuju pisanje istog koda više puta.

Nasleđivanje

Koncept nasleđivanja omogućava programerima da prilikom pisanja koda produže funkcionalnost postojećih klasa unutar koda da ne bi došlo do stalnog ponavljanja koda.

Može da se desi da elementi koda uključuju iste osobine, koje poseduju više pojedinačnih metoda. Umesto da se te osobine i metode svaki put ponovo definišu svaki put kada se naiđe na element koda koji ih sadrži, mogu se definisati jednom kao generički objekat (en. generic object). Na taj način specifični objekti mogu naslediiti osobine i metode, smanjujući ponavljanje koda.

Glavi objekat postaje nadklasa ili super klasa (en. superclass) a izveden oobjekat postaje podklasa ili izvedena klasa (en. subclass).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Polimorfizam

Koncept polimorfizma omogućava programerima da prilikom pisanja koda pruži mogućnost da jednom napisani kod daje različit rezultate u zavisnosti od ulaza.

Polimorfizam predstavlja obezbeđivanje jedinstvenog interfejsa prema entitetima različitih tipova.

Može biti *ad-hoc* polimorfizam, parametarski polimorfizam, hijerarhijski polimorfizam i implicitni polimorfizam.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 3

Klasa i objekti

POJAM KLASE

Klasa predstavlja šablon (en. blueprint) za pravljenje objekata

Klasa (en. class) pruža mogućnost združivanja podataka i funkcionalnosti.

Kreiranjem nove klase kreira se novi tip objekta, koji potom dozvoljava kreaciju novih instanci tog tipa podataka.

Svaka instanca klase ima atribute koje su joj dodeljeni. Instance klase takođe imaju metode, koje su definisane unutar klase.

U poređenju sa ostalim programskim jezicima, mehanizam klasa u Python 3.x jeziku je sličan mehanizmima u C++ jeziku i Modula-3 jeziku.

Klase u Python jeziku pružaju sve standarde objektno-orjentisane paradigme programiranja. Osobina nasleđivanja dozvoljava više baznih)(super) klasa, izvedena klasa može izvršiti redefinisanje metode (en. method override) svoje (ili svojih) super klasa, a metoda može pozvati metodu bazne klase istog imena.

Objekat (en. object) može sadržati proizvoljan broj podataka bilo kog tipa.

Klase (i moduli) u Python 3.x jeziku putem osobine dinamičkog tipiziranja se prave prilikom izvršenja (en. runtime) i mogu se modifikovati nakon kreacije.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

IMENSKI PROSTOR I OPSEG U PYTHON 3.X JEZIKU

Treba voditi računa o opsegu imenskog prostora prilikom definisanja klase - ne pretražuju se svi opsezi podjednako

Imenski prostor (en. namespace) prestavlja mapiranje imana na objekte.

Najveći broj imenskih prostora u Python jeziku jesu implementirani kroz imenike. Primeri imenskih prostora uključuju skup ugrađenih imena (primer može biti ugrađena funkcija), globalna imena (primer unutar modula), i lokalna imena (primer unutar funkcije)

Imenski prostori se kreiraju u različitim trenucima i imaju različit vek trajanja.

Imenski prostor koji sadrži ugrađena imena kreira se prilikom inicijalizacije Python interpretera, i ne briše se.

Globalni imenski prostor za modul se pravi kada se učita definicija modula, i takođe ostaju dok se interpreter ne isključi.

Lokalni imenski prostor funkcije kreira se kada se poziva funkcija, i briše se kada se funkcija završi (vrati povratnu vrednost, ili vrati izuzetak koji je može obraditi).

https://docs.python.org/3/tutorial/classes.html

Opseg (en. scope) predstavlja deo koda u Python jeziku u kome je imenski prostor direktno dostupan.

Iako se opsezi statički određuju, koriste se dinamički. U bilo kom vremenu tokom izvršenja, postoje tri ili četiri ugnježdena opsega čiji su imenski prostori direktno dostupni:

- Unutrašnji opseg (en.innermost scope), koji se prvi pretražuje i sadrži lokalna imena,
- Opseg okružujućih funkcija (en. enclosing functions scope), koji se pretražuje počevši za najbližom okružujućom funkcijom, i sadrži ne-lokalna, ali i ne-globalna imena,
- Opseg globalnih imena modula koji se pretražuje pretposlednji,
- Spoljašnji opseg (en. outermost scope) koji sadrži imenski prostori sa ugrađenim imenima i kojie se pretražuje poslednji.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

3.1 Metode u klasama

ŠTA PREDSTAVLJA METODA KLASE?

Metode pružaju interfejs ka i od objekta.

Definicija:

Metoda (en. method) u objektno-orjentisanom programiranju prestavlja proceduru koja sadrži poruku i objekat.

Metode se definišu u klasi, i pružaju interfejs koji ostale klase koriste da pristupe i izmene osobine objekta.

Zanimljivost:

U Python 3.x jeziku, metoda nije unikatna za instance klase. Drugi tipovi objekata takođe mogu imati metode.

Primer: Objekti tipa list imaju metode .append(), .insert(), .sort() i sl.

U opštem slučaju, može se reći da metoda predstavlja funkciju koja pripada objektu.

Kao i u mnogim drugim OOP jezicima, postoje specijalni tipovi metoda.

Konstruktor

Konstruktor (en. constructor) predstavlja specijalnu metodu koju Python poziva prilikom instanciranja objekta, korišćenjem definicija unutar klase.

Python se oslanja na konstruktor da izvrši inicijalizaciju, odnosno da dodeli vrednosti svakoj instanci objekta kada se napravi.

U Python 3.x jeziku ime konsturktora je ***__init__()***

Konstruktor može prihvatiti argumente kada je neophodno da se napravi objekat. Kada se napravi klasa bez konstruktora, Python automatski napravi default-ni konstruktor koji zapravo ništa ne radi.

Svaka klasa mora imati konstruktor, makar i default konstruktor. Sintaksa konstruktora je sledeća:

```
def __init__(self):
    # telo konstruktora
```

VRSTE KONSTRUKTORA

Konstruktor može biti default-ni ili parametrizovan

Default konstruktor

Ovakav konstruktor ne prihvata nikakve argumente. Njegova definicija ima samo jedan argument (***self***) koji je refenca na instancu koja se konstruiše.

Primer:

Napraviti klasu ***CS324*** koja sadrži atribut puno ime predmeta, i metodu koja štampa puno ime predmeta.

```
class CS324:

    # default konstruktor
    def __init__(self):
        self.naziv = "Skripting jezici"

    # netoda za štampanje
    def printNaziv(self):
        print(self.naziv)

# kreiranje objekta
predmet = CS324()
```

```
# poziv metode
predmet.printNaziv()
```

Parametrizovan konstruktor

Konstruktor koji pored self reference sadrži dodatne parametre naziva se parametrizovan konstruktor. Prvi parametar je i dalje referenca na sebe, dok ostale argumente prosleđuje programer.

Primer:

Napraviti klasu ***fitPredmet*** koja kao ulazne parametre ima šifru u naziv predmeta. Klasa sadrži artibut godina (početna vrednost nula), i metode za unos vrednosti godine (na kojoj se služa predmet), kao i metodu za štampanje šifre, naziva i godine na kojoj se sluša.

```
class fitPredmet:
    godina = 0
    def __init__(self, sifra, naziv):
        self.sifra = sifra
        self.naziv = naziv
    def unosGodine(self):
        self.godina = int(input("Uneti godinu na kojoj se sluša predmet: "))
    def predmetPrint(self):
        print(f"Predmet {self.sifra} - {self.naziv} se sluša na {self.godina}.
godini studija.")


cs324 = fitPredmet("CS324", "Skripting Jezici")
cs324.unosGodine()
cs324.predmetPrint()
```

VRSTE KONSTRUKTORA - VIDEO OBJAŠNJENJE

U nastavku su dati video materijali koji objašnjavaju vrste konstruktora u Python 3.x jeziku

Default konstruktor

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Parametrizovan konstruktor

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

3.2 Nasleđivanje, kompozicija, delegacija

NASLEĐIVANJE U PYTHON 3.X JEZIKU

Izvedena klasa nesleđuje metode i atribute osnovne klase

Nasleđivanje (en. inheritance) je jedan od osnovnih principa objektno-orjentisanog programiranja.

Kod nasleđivanja, kao što je rečeno imamo super klasu (nadklasu, ili klasu roditelja) i izvedenu klasu (podklasu, klasu deteta) koja nasleđuje atribute i metode svoje super klase.

Može se reći da izvedena klasa proširuje super klasu.

Super klasu ne treba posebno definisati, već se definiše kao normalna klasa.

Primer:

Napraviti klasu ***vozilo*** koje ima atribute godina proizvodnje i broj točkova.

Izvesti klasu ***automobil*** koje nasleđuje sve od klase vozilo.

```
class vozilo:
    godinaProizvodnje = None
    brojTockova = None

class automobil(vozilo):
    pass


veh1 = vozilo()
veh1.godinaProizvodnje = 2000
veh1.brojTockova = 4

print(type(veh1))

auto1 = automobil()
auto1.godinaProizvodnje = 2020
auto1.brojTockova = 4

print(type(auto1))
```

Kod izvedene klase moguće je redefinisati postojeću metodu super klase..

Primer:

Napraviti klasu ***playerCharacter*** bez ulaznih argumenata, koja ima atribut ***stats*** koji je inicijalno niz od šest nula. Definisati metodu ***setStats*** kojom se ubacuju šest celobrojnih parametra, i metodu ***showStats*** kojom se štampa stats.

Izvesti klasu ***human*** iz klase ***playerCharacter***, i redefinisati ***setStats*** tako da se na svaki ubačeni broj doda jedinica.

Napraviti instancu ***playerCharacter*** i ***human***, i dati im iste ***stats***, i nakon toga ih prikazati na izlazu.

```
class playerCharacter():
    stats = [0, 0, 0, 0, 0, 0]
    def setStats(self):
        for i in range(6):
            self.stats[i] = int(input(f"Uneti {i + 1}. stat: "))
    def showStats(self):
        print(f"Stats: {self.stats}")

pc1 = playerCharacter()
pc1.showStats()

pc1.setStats()
pc1.showStats()

class human(playerCharacter):
    def setStats(self):
        for i in range(6):
            self.stats[i] = int(input(f"Uneti {i + 1}. stat: ")) + 1

pc2 = human()
pc2.setStats()
pc2.showStats()
```

NASLEĐIVANJE NASPRAM KOMPOZICIJE

Veza između dva objekta u nasleđivanju je "je veza" dok je pri kompoziciji "ima veza"

Video iz prethodnog primera

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Nasleđivanje jeste dobro ukoliko želimo da prenesemo sve atribute i metode super klase u izvedenu klasu.

Tada je izvedena klasa ujedno i super klasa, odnosno postoji tzv. *je veza* između super klase i izvedene klase.

Sa druge strane, ukoliko postoji tzv. *ima veza* između dva objekta, onda je u pitanju kompozicija.

Primer za nasleđivanje:

Definiše se super klasa ***povrće***, i izvedena klasa ***krompir***. Postoji *je veza*, jer je krompir povrće, i naslediće sve elemente te klase.

Primer za kompoziciju:

Definišemo klasu ***akumulator***. Zatim, definišemo klasu ***automobil*** čiji jedan atribut jeste instanca klase akumulator. U ovom slučaju postoji*ima veza*, jer će svaki automobil imati akumulator.

Detaljnije razlike između kompozicije i nasleđivanja obrađuje se u drugim predmetima.

Poglavlje 4

Rad sa klasama i instacama

INSTANCA KAO KONKRETNI OBJEKAT

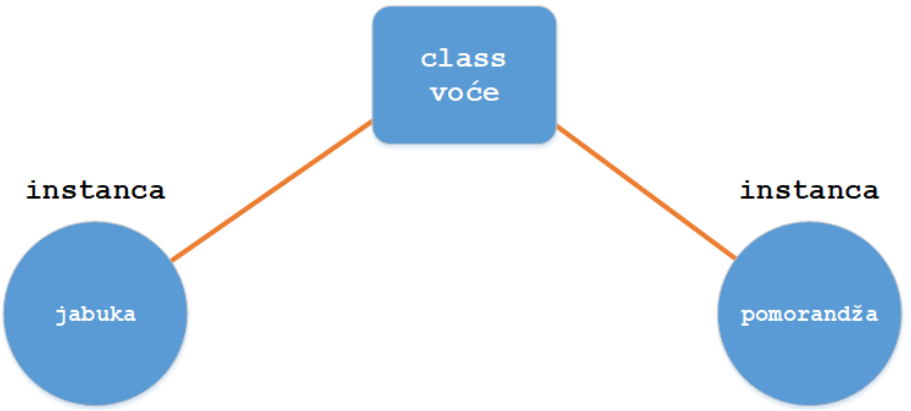
Instanca ima životni ciklus obično tokom izvršenja programa.

U OOP paradigmama, instanca (en. instance) predstavlja konkretan primerak bilo kog objekta.

Instanca kao takva ima životni ciklus obično tokom izvršenja programa.

Često je instanca sinonim za objekat jer oba termina označavaju konkretnu realizaciju neke klase. Međutim, kada se koristi termin instanca, naglašava se da je u pitanju poseban identitet objekta.

Kreacija instance jeste instaciranje (en. instantiation)



Slika 4.1.1 Klasa i instance klase [Izvor: Autor.]

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

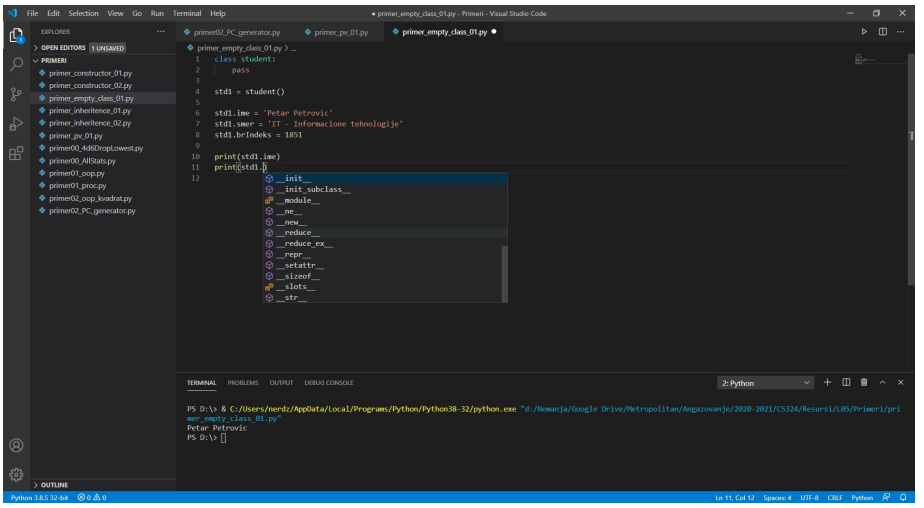
PRAZNA KLASA

Moguće je napraviti praznu klasu koju kasnije mogu

U Python 3.x programskom jeziku moguće je napraviti praznu klasu, odnosno klasu koja ne sadrži nikakve atribute i metode.

Nakon toga, mogu se naknadno dodati atributi.

Međutim, u klasičnom OOP pristupu ovo se ne preporučuje.



Slika 4.1.2 Definisanje prazne klase student, i dodavanje atributa nakon instaciranja. [Izvor: Autor.]

Pitanje:

Kada i gde se koristi prazna klasa?

```
class student:
    pass

std1 = student()

std1.ime = 'Petar Petrovic'
std1.smer = 'IT - Informacione tehnologije'
std1.brIndeks = 1851

print(std1.ime)
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

4.1 Uvoz klase

UVOZ KLASE U PYTHON 3.X JEZIKU

Prednost OOP-a u Python jeziku jeste ta što može da se iskoriste klase iz drugih datoteka ukoliko je poznaza putanja to tih datoteka

U proceduralnom programiranju čest je slučaj da je potrebno se sve promenljive i funkcije (koje nisu ugrađene) nalaze u istoj datoteci.

U Python 3.x programskom jeziku moguće je izvršiti uvoz klase (en. class import) i iz drugih datoteka.

Takođe, nije neophodno ni da se sve datoteke čije klase želimo koristiti nalaze u istom radnom direktorijumu, već je moguće koristiti datoteke iz drugih direktorijuma.

Potrebno je znati relativnu putanju od izvorne do odredišne datoteke, kao i da u direktorijumu iz koje uzimamo klase postoji prazna ***__init__.py*** datoteka, koja označava je moguće uvoziti datoteke iz tog direktorijuma.

Primer:

Napraviti klasu koja simulira bacanje kockica sa metodom koja pokazuje rezultat. U posebnoj datoteci uvesti klasu iz prve datoteke i napraviti novu instancu te klase. Isprobati funkcionalnost.

Alternativno: Realizovati metodu vraćanja rezultata kao lambda funkcija.

https://stackoverflow.com/questions/4142151/how-to-import-the-class-within-the-same-directory-or-sub-directory

```
# bacanjeKockica
import random

class d6roller():
    def rolld6(self):
        return random.randint(1,6)

# glavniProgram
import d6roller as d6
x = d6.d6roller()
print(x.rolld6())
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 5

Klase naspram modula

KLASA NASPRAM MODULA U PYTHON 3.X JEZIKU

Klasa i modul mogu naizgled da služe istoj svrsi, ali nije tako

Klasa

Klasa u Python 3.x jeziku, kao i kod drugih objektno-orjentisanih programskih jezika predstavlja osnovu za apstrakciju podataka.

Klasa se koristi za apstrakciju zajedničkih karakteristika različitih objekata. Klasa enkapsulira podatke i operacije za buduću upotrebu.

Klasa se može definisati u glavnom programu ili unutar modula koji glavni program uveze.

Iz klasa kreiraju se instance te klase. To su pojedinačni objekti sa jedinstvenim skupom podataka u svojim atributima.

Klasa se može proširiti ili modifikovati korišćenjem osobine nasleđivanja.

Modul

Svaka .py datoteka može se smatrati da je modul (en. module). Nakon pisanja skript datoteka, definišu se funkcije i promenljive. Te funkcije i promenljive se mogu ponovo iskoristiti tako što će se uvesti (en. import) ta datoteka kao modul (bez ekstenzije *.py*)

Može se reći da je modul enkapsulira kod koji se može ponovo iskoristiti. Moduli sadrže funkcije ali i mogu sadržati i klase.

Može postojati samo jedan modul. Kada se uveze modul, to je isti objekat.

Modul se *ne može* proširivati ili modifikovati kao što klasa može (kroz nasleđivanje).

KADA TREBA KORISTITI KLASE, A KADA UVESTI MODULE?

Koristiti klase za šablone, a module za dodatnu funkcionalnost

Podsetnik:

Koristiti klase kao šablone za objekte koje modeluju doen problema.

Korstiti modulezadodatnu funkcionalnost u kodu.

Paket

Za razliku od pojedinačnih modula, koji se čuvaju kao *.py* datoteke, paket predstavlja direktorijum u kome se nalaze pojedinačni moduli (obično slične problematike).

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 6

Pokazna vežba #5

UVOD U POKAZNU VEŽBU #5

U pokaznoj vežbi #5 prolaze se klase, atributi i metode, kao i izvođenje nove klase iz superklase.

U pokaznoj vežbi proći će se kroz zadatak koji obuhvata sve tematske jedinice ove lekcije:

- Klasa
- Metoda klase
- Superklasa i izvedena klasa
- Redefinisanje metode klase

Procenjeno trajanje pokazne vežbe iznosi 90 minuta.

PRIMER 1: KLASSA WARRIOR

Primer klase sa parametrizovanim konstruktorom i različitim metodama

Zadatak #1 *(20 minuta)*

Napraviti klasu ***warrior*** koja sadrži atribute ***name***, ***health***, ***armorClass***, ***attackBonus***, i ***damage***.

Klasa ima sledeće metode:

- ***rollForAttack***: Vraća vrednost attackBonus i nasumičnog celog broja između 1 i 20 (simulira bacanje 20-strane kockice)
- ***checkHit***: proverava da li je svoj AC manji od vrednosti napada.
- ***takeDamage***: smanji svoj health za ulazni parametar damage.

Napraviti dva objekta klase warrior. Neka jedan napadne drugog. Štampati preostali ***health***.

```
class warrior():
    def __init__(self, name, health, armorClass, attackBonus, damage):
        self.name = name
        self.health = health
        self.armorClass = armorClass
        self.attackBonus = attackBonus
        self.damage = damage
```

```
    def rollForAttack(self):
        import random
        print(self.name, "attacks!")
        return self.attackBonus + random.randint(1, 20)

    def checkHit(self, attackRoll):
        if attackRoll >= self.armorClass:
            return True
        else:
            return False

    def takeDamage(self, damage):
        self.health -= damage

warr1 = warrior("Rogar", 30, 14, 5, 10)
print(f"{warr1.name} has", warr1.health, "HP left.")
warr2 = warrior("Nimble", 20, 12, 6, 8)
print(f"{warr2.name} has", warr2.health, "HP left.")

# Rogar attacks Nimble
atk1 = warr1.rollForAttack()
if warr2.checkHit(atk1):
    print(f"Rolled {atk1}, AC is {warr2.armorClass}, hit!")
    warr2.takeDamage(warr1.damage)
else:
    print(f"Rolled {atk1}, AC is {warr2.armorClass}, miss!")

print(f"{warr2.name} has", warr2.health, "HP left.")
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PRIMER 2: PLAYER CHARACTER I IZVEDENA HUMAN KLASA

Primer u pokaznoj vežbi pokazuje inicijalizaciju klase, ali i redefinisanje metode u izvedenoj klasi.

***Zadatak #2* (25 minuta)**

Napraviti klasu ***playerChar*** koja ima sledeće atribute:

- Ime
- Pol
- Godine

I jedan niz za statistiku (*Strength, Dexterity, Constitution, Intelligence, Wisdom i Charisma*) koji je inicijalno popunjen nulama.

Napraviti metodu za popunjavanje niza na sledeći način:

Simulirati bacanje kockice četiri puta, odbacivanje najmanjeg broja, i sabiranja preostala tri, kao posebnu metodu, i onda metodu koja poziva bacanje kockica, i popunjava statistiku lika.

Napraviti izvedenu klasu ***human*** iz klase ***playerChar***, koja redefiniše popunjavanje statistike, tako što na svaki rezultat bacanja kockice doda 1.

U glavnom programu napraviti objekte pc1 koji je tipa ***playerChar***, i pc2, koji je ***human***, i popuniti im statistiku. Na kraju štampati statistiku.

```
# D&D 5e player character class

class playerChar:
    # stats are: Strength, Dexterity, Constitution, Intelligence, Wisdom & Charisma
    stats = [0, 0, 0, 0, 0, 0]
    # initialization
    def __init__(self, name, gender, age):
        self.name = name
        self.gender = gender
        self.age = age

    # stat roller method 4d6 drop lowest
    def statRoller(self):
        import random
        stat = []
        for _ in range(4):
            stat.append(random.randint(1 , 6))
        stat.sort()
        stat.pop(0)
        return sum(stat)

    # roller for all stats
    def rollStats(self):
        allStats = []
        for _ in range(6):
            x = self.statRoller()
            allStats.append(x)
        return allStats

# human class from playerChar has all stats + 1
class human(playerChar):
    def rollStats(self):
        allStats = []
        for _ in range(6):
            x = self.statRoller()
            allStats.append(x + 1)
        return allStats

# main program
pc1 = playerChar("Random Name Placeholder the Third", "Male", 56)
pc1.stats = pc1.rollStats()
print(f"The character {pc1.name} has the followings stats: \n {pc1.stats}")
```

```
pc2 = human("Gilmithrie of Ashtramoor", "Male", 21)
pc2.stats = pc2.rollStats()
print(f"The character {pc2.name} has the followings stats: \n {pc2.stats}")
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

Poglavlje 7

Individualna Vežba #5

UVOD U INDIVIDUALNU VEŽBI #5

U individualnoj vežbi 5 rade se tri zadatka koje bi studenti mogli samostalno da odrade u toku dva časa.

Individualna vežba #5 odnosi se na rad sa klasama i objektima.

Procenjeno trajanje individualnih vežbi iznosi 90 minuta.

INDIVIDUALNA VEŽBA

Studenti bi trebalo samostalno da reše sva tri zadatka u vreme trajanja dva časa individualnih vežbi.

Zadatak #1 (25 minuta)

Napraviti klasu ***randomNiz*** koji ima atribut broj elemenata u nizu i tip elemenata (u za pozitivne cele brojeve, i za cele brojeve, f za razlomljene brojeve).Napraviti (ručno) metode sortiranje niza u:

- rastućem redosledu
- opadajućem redosledu
- cik-cak redosledu (najmanji element ima prvi indeks, sledeći ima poslednji indeks, nakon njega ima drugi indeks, itd.)

Zadatak #2 (35 minuta)
Napraviti klasu ***geometrijskaFigura*** koja kao atribute ima dužinu proizvoljnog broja stranica.

Na osnovu broja unetih stranica odrediti o kojoj figuri se radi (trougao, četvorougao, petougao i sl.), i izračunati obim i površinu. Za svaku figuru odrediti da li je moguće konstruisati takvu figuru sa unetim dužinama stranica.

Instancirati nekoliko različitih figura i štampati obim i površinu. Odrediti koja figura ima najmanju površinu i vratiti koja je figura u pitanju. Instancirati trougao, četvorougao, petougao i šestougao sa istim dužinama stranica. Poređati obime u niz u opadajućem redosledu.
Zadatak #3 (30 minuta)

Napraviti klasu ***videoSnimak*** koja ima kao atribute ***ime*** i ***duzinaTrajanja***.

Izvesti klasu ***film*** koji ima i atribute ***godinaIzdavanja***, ***zanr***, ***reziser***.

Izvesti klasu ***serija*** koja ima i atribute ***zanr***, ***brojSezona***, ***brojEpPoSezoni***.

Za obe izvedene klase napraviti ***rejting*** (1 do 5 zvezdica)

Instancirati nekoliko objekata od svake klase.

Optimizovati kod.

Poglavlje 8

Domaći zadatak #5

DOMAĆI ZADATAK

Domaći zadatak #5 se okvirno radi 2h

Domaći zadatak #5

Zadatak #1

Napraviti klasu dokument sa osobinama ime i broj reči. Izvesti klasu knjiga koja ima dodatne osobine autor, žanr, godina izdavanja.

Instancirati 10 knjiga. Napraviti imenik koji će kao ključ imati broj knjige (počevši od lib001) a kao vrednost knjigu. Štampati sve knjige u formatu:

< broj knjige: žanr, autor, naziv.>

Zadatak #2

Napraviti klasu osoba, koja će imati osobine ime i prezime.

Nakon toga, izvesti klasu student koja će imati dodatne osobine broj_indeksa, smer, i položene ispite.

Položene ispite napraviti kao imenik gde je ključ šifra predmeta, a ocena vrednost. Napraviti dva objekta klase student i popuniti sve osobine.

Naći da li su studenti na istom smeru ili ne, koliko je koji student položio ispite, i da li imaju ispite koje su oba studenta položili.

Tradicionalni studenti:

Domaći zadatak treba dostaviti najkasnije nedelju dana nakon predavanja za 100% poena. Nakon toga poeni se umanjuju za 50%.

Online studenti:

Domaći zadatak treba dostaviti najkasnije 10 dana pred polaganja ispita. **Domaći zadaci se brane!**
Svi studenti domaći zadatak poslati dr Nemanji Zdravkoviću: nemanja.zdravkovic@metropolitan.ac.rs

Poglavlje 9

Zaključak

ZAKLJUČAK

Zaključak lekcije #5

Rezime:

U ovo lekciji bilo je reči o objektno-orjentisanom programiranju u Python 3.x jeziku

Uvedeni su pojmovi klase, objekta i instance, i date su osnovne osobine OOP-a kao paradigme programiranja.

Zatim, bilo je reči o elementima klase, metodama i atributima, i njihovom međusobnom komunikacijom.

Bilo je reči o super klasama i izvedenim klasama, razlikama između klasa i modula, o kojima će biti mnogo više reči u sledećim lekcijama.

Primeri u okviru lekcije kao i zadaci za individualni rad treba da osposobe studente za rad u Python 3.x jeziku kroz paradigmu OOP-a.

Literatura:

1. David Beazley, Brian Jones, Python Cookbook: Recipes for Mastering Python 3, 3rd edition, O'Reilly Press, 2013.
2. Mark Lutz, Learning Python, 5th Edition, O’Reilly Press, 2013.
3. Andrew Bird, Lau Cher Han, et. al, The Python Workshop, Packt Publishing, 2019.
4. Al Sweigart, Automate the boring stuff with Python, 2nd Edition, No Starch Press, 2020.