Kurs: Programiranje igara sa programskim jezikom Pajton
014 čas: Program Ljubimac.py
Teme: objekti, klase, metode, atributi, instancijacija

Osnove orjentacije prema objektima

OOP (Objektno orjentisano programiranje) ima reputaciju da je kompleksno, ali je zapravo jednostavnije od mnogih do sada naučenih koncepata. OOP omogućava predstavljanje stvari na način koji je bliži realnosti.

Sve što se želi predstaviti programom su realni objekti u stvarnosti. OOP omogućava predstavljanje ovih objekata kao softverskih objekata. Poput stvarnih objekata, softverski objekti kombinuju karakteristike (zvanih i **atributi** u OOP) i ponašanja (zvanih **metode** u OOP). Npr, za svemirski brod atributi mogu biti njegova lokacija i količina energije, dok su njegove metode mogućnost kretanja ili da li trenutno puca iz svojih oružija.

Objekti su kreirani (**instancirani** u OOP) iz definicije koja se naziva **klasa** – programerski kod koji definiše atribute i metode. Klase su poput planova za izgradnju. Klasa nije objekat, već je dizajn za objekat. Programer može kreirati veliki broj objekata iz iste klase. Kao rezultat, svaki objekat (koji se još zove i **instanca**) koji je instanciran iz iste klase ima sličnu strukturu.

Ali, kao što se dve kuće mogu napraviti prema istim planovima izgradnje a dekorisati drugačije, mogu se i dva objekta iste klase razlikovati sa sopstvenim jedinstvenim setom vrednosti atributa.

Kreiranje klasa, metoda i objekata

Za izgradnju objekta, potrebno je imati plan izgradnje tj klasu. Klase skoro uvek uključuju metode, tj radnje koje objekat može da radi.

program Mali_ljubimac.py

```
#Mali_ljubimac
#Prikazuje osnove klasa i objekata
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def prica(self):
        print("Zdravo. Ja sam instanca klase Ljubimac.")

mali = Ljubimac()
mali.prica()
```

001 Definisanje klase

Program počinje definicijom klase, planom izgradnje prvog ljubimca. Prva linija definicije je heder klase:

```
class Ljubimac(object):
```

koristi se ključna reč class i ime klase po izboru, Ljubimac. Izbor prvog slova je veliko slovo. Pajton to ne vidi kao pravilo ali se upotrebljava kao standardna konvencija, pa zato treba započeti sva imena klasa sa velikim slovom.

U zagradi je rečeno Pajtonu da zasniva klasu na object, koji je osnovni ugrađeni tip. Nova klasa se može zasnivati na object ili bilo kojoj prethodno definisanoj klasi.

Sledeća linija je docstrig, koji dokumentuje klasu. Dobar docstring opisuje koji tip objekta klase se može koristiti za kreiranje: `"""Virtuelni ljubimac"""`

002 Definisanje metoda

Poslednji deo klase definiše metod. Veoma liči na funkciju:

```
def prica(self):
```

```
        print("Zdravo. Ja sam instanca klase Ljubimac.")
```

Zapravo se može razmišljati o metodama kao funkcijama pridodatim nekom objektu. Metoda prica() samo štampa string.

Metoda prica() ima samo jedan parametar, self (koji se u ovom primeru i ne koristi). Svaki **metod instance** , metod koji svaki objekat klase ima, mora imati poseban prvi parametar, koji se po konvenciji naziva self. Ovaj parametar omogućava da se metod uputi na sam objekat. Ako se kreira metod instance bez ikakvog parametra, generiše se greška prilikom pokretanja. Sve metode instanci moraju imati poseban prvi parametar, koji se naziva self.

003 Instancijacija objekta

Posle pisanja klase, instancijacija novog objekta zahteva samo jednu liniju koda:

```
mali = Ljubimac()
```

Ova linija kreira potpuno nov objekat klase Ljubimac i dodeljuje ga promenjivoj mali. Vidi se da postoje zagrade posle imena klase Ljubimac u iskazu dodele. Oe se moraju koristiti ako se želi napraviti novi objekat.

Može se dodeliti novo instancirani objekat promenjivoj sa bilo kakvim imenom. Ime promenjive ne mora biri bazirano na nazivu klase. Ipak treba izbegavati davanje istih imena klasama i njihovim objektima, tj promenjivima.

004 Podizanje metode

Novi objekat ima metod koji se zove prica(). To je zapravo funkcija koja pripada objektu. Može se podići (invoke) ovaj metod kao bilo koji drugi, korišćenjem dot notacije:

```
mali.prica()
```

Ova linija podiže metod prica() objekta Ljubimac koji je dodeljen mali. Metod samo štampa string.

Konstruktori

Do sada je prikazan proces kreiranja metoda, poput prica(), ali postoje posebne metode koje se mogu kreirati a zovu se konstruktori, koji se automatski podižu odmah posle kreiranja novog objekta. Konstruktor je veoma korisna metoda. Često će se koristiti jedan konstruktor za svaku klasu koja se kreira. Obično se konstruktor koristi za postavke inicijalnih vrednosti atributa nekog objekta, iako se neće za to koristiti u sledećem primeru.

program Konstruktor_ljubimca.py

```
#Konstruktor_ljubimca
#Prikazuje konstruktore
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self):
        print("Novi ljubimac je rodjen!")

    def prica(self):
        print("\nZdravo. Ja sam instanca klase Ljubimac.")

mali1 = Ljubimac()
mali2 = Ljubimac()
mali1.prica()
mali2.prica()
```

```
Novi ljubimac je rodjen!
Novi ljubimac je rodjen!

Zdravo. Ja sam instanca klase Ljubimac.

Zdravo. Ja sam instanca klase Ljubimac.
```

005 Kreiranje konstruktora

Prvi deo koda u definiciji klase je **metod konstruktor** (zvan i metod inicijalizacije):

```
def __init__(self):
        print("Novi ljubimac je rodjen!")
```

Obično se koriste bilo kakva imena za sopstveno napravljene metode, ali se ovde koristi posebno ime metode koje Pajton prepoznaje. Imenovanjem metode __init__, kaže se Pajtonu da je ovo moj metod konstruktor. Kao metod konstruktor, __init__() se automatski poziva od strane svakog novo kreiranog Ljubimac objekta odmah posle njegovog kreiranja. To znači da novo kreirani Ljubimac objekat automatski štampa string iz bloka konstruktora.

Pajton ima niz ugrađenih specijalnih metoda čija imena počinju i završavaju se sa donjim linijama, poput metode konstruktor __init__.

006 Kreiranje više objekata

Kada je jednom napisana klasa, kreiranje više objekata je jednostavno. U programu su kreirana dva objekta:

```
mali1 = Ljubimac()
mali2 = Ljubimac()
```

Kao rezultat, dva objekta su kreirana. Odmah posle instancijajacije svakog od njih, štampa se string `"Novi ljubimac je rodjen!"` kroz metod konstruktor.

Napravljeni objekti su nezavisni jedan od drugog:

```
mali1.prica()
mali2.prica()
```

Vidi se da se mora podići posebno metod prica() za svakog od objekata ponaosob iako se na taj način štampa potpuno isti string, pošto su to različiti objekti.

Atributi

Moguće je urediti da se atributi objekta automatski kreiraju i inicijalizuju odmah posle instancijacije kroz njen metod konstruktor.

program Atribut_ljubimca.py

```
#Atribut_ljubimca
#Prikazuje kreiranje i pristup atributima objekta
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime):
        print("Novi ljubimac je stvoren!")
        self.ime = ime

    def __str__(self):
        ispis = "Objekat ljubimac\n"
        ispis += "ime: " + self.ime + "\n"
        return ispis

    def prica(self):
        print("Zdravo. Ja sam", self.ime, "\n")

mali1 = Ljubimac("Puk")
mali1.prica()
mali2 = Ljubimac("Miki")
mali2.prica()
print("Stampam mali1:")
print(mali1)
print("Direktno pristupam mali1.ime:")
print(mali1.ime)
```

```
Novi ljubimac je stvoren!
Zdravo. Ja sam Puk

Novi ljubimac je stvoren!
Zdravo. Ja sam Miki

Stampam mali1:
Objekat ljubimac
ime: Puk

Direktno pristupam mali1.ime:
Puk
```

007 Inicijalizacija atributa

Konstruktor u ovom programu štampa poruku `"Novi ljubimac je stvoren!"` poput konstruktora iz programa Konstruktor_ljubimac.py, ali sledeća linija metode radi nešto drugo. Ona kreira atribut ime za novi objekat i postavlja je na vrednost parametra ime. Zato linija:

```
mali1 = Ljubimac("Puk")
```

rezultuje u kreaciji novog Ljubimac objekta sa atributom ime postavljenim na "Puk". Na kraju, objekat je dodeljen promenjivoj mali1.

Kao prvi parametar svake metode, self automatski dobija referencu na objekat koji podiže metodu. To znači da, kroz self, metod može doći do objekta koji ga podiže i pristupiti atributima i metodama objekta (čak i kreirati nove atribute za objekat).

Parametar self automatski dobija referencu na novi Ljubimac objekat dok parametar ime dobija "Puk". Zatim u liniji:

```
self.ime = ime
```

se kreira atribut ime za objekat i postavlja ga na vrednost iz ime, koje je "Puk".

U glavnom delu programa, iskaz dodele dodeljuje taj novi objekat u mali1. To znači da mali1 se odnosi prema novom objektu sa sopstvenim atributom ime postavljenim na "Puk". Tako je kreiran ljubimac sa sopstvenim imenom.

Linija u glavnom delu programa: `mali2 = Ljubimac("Miki")`

započinje isti osnovni lanac događaja. Ali, ovoga puta, nov Ljubimac objekat je kreiran sa sopstvenim atributom ime postavljenim na "Miki". A objekat je dodeljen promenljivoj mali2.

008 Pristupanje atributima

Nikakva korist od atributa ako ne mogu da se upotrebljavaju, pa je zato napisana metoda prica() koja koristi atribut ime objekta Ljubimac. Sada, kada se ljubimac pozdravlja istovremeno kaže i svoje ime.

Prvi ljubimac podiže njegovu prica() metodu sa: `mali1.prica()`

Metoda prica() prima automatski poslatu referencu na objekat unutar self parametra:

```
def prica(self):
```

Zatim, funkcija print prikazuje tekst `Zdravo. Ja sam Puk.` Pristupanjem atributu ime objekta kroz self.ime:

```
print("Zdravo. Ja sam", self.ime, "\n")
```

Po difoltu, može se pristupiti i modifikovati atributi objekata izvan njihove klase. U glavnom delu programa, direktno se pristupa ime atributu od mali1:

```
print(mali1.ime)
```

Linija štampa string “Puk“. U principu, za pristup atributu objekta izvan klase objekta, koristi se dot notacija. Kuca se ime promenjive, tačka i naziv atributa.
Najčešće ne želimo da omogućimo direktan pristup atributima objekata izvan definicije klase.

009 Štampanje objekta

Ako se želi štampati objekat sa kodom `print(mali1)` , Pajton bi posalao na izlaz nešto nalik na:
<__main__.Ljubimac object at 0x00A0BA90>
Ovo govori da je odštampan Ljubimac objekat u glavnom delu programa, ali ne daje korisne informacije o samom objektu. Ipak, postoji način da se to promeni. Uključivanjem posebne metode __str__() u definiciji klase, kreira se string prikaz objekta koji će biti prikazan kada god se objekat odštampa. Koji god string se dobije iz metode će biti string koji se štampa za taj objekat.
Metoda __str__() iz programa vraća string koji uključuje vrednost ime atributa objekta. Tako da , kada se izvršava linija `print(mali1)` , pojavljuje se korisniji tekst:

```
Objekat ljubimac
ime: Puk
```

Čak i kada programer ne planira da štampa objekat u programu, kreiranje __str__() metode nije loša ideja. Na taj način se mogu videti vrednosti atributa objekta čime se povećava razumevanja funkcionisanja programa.

Atributi klasa i metoda static

Preko atributa, različiti objekti iste klase mogu imati jedinstvene vrednosti. Može se stvoriti 10 ljubimaca svaki sa svojim imenom. Ali, moguće je imati i informacije koje se ne odnose na individualne objekte nego na celu klasu. Npr, može se voditi računa o broju ljubimaca koji su kreirani, tako što se svakom ljubimac objektu dodeli atribut suma. I onda, kada god je nov objekat instanciran, updejtuje se atribut suma svakog postojećeg objekta. Ovo bi moglo biti komplikovano uraditi. Na sreću, Pajton omogućava način za kreiranje jedne vrednosti koja je povezana sa samom klasom, a zove se **atribut klase**. Ako je klasa poput plan izgradnje, onda je atribut klase je kao nalepnica na planu izgradnje. Postoji samo jedna kopija, bez obzira koliko se stvari napravi pomoću plana izgradnje.
Često je potrebno naprviti metod pridružen klasi: za to Pajton nudi **metodu statik**. Pošto su statik metode pridružene klasi, često se koriste za rad sa atributom klase.

program Klasni_ljubimac.py

```
#Klasni_ljubimac
#Prikazuje atribut klase i metodu statik
class Ljubimac(object):
    """Virtuelni ljubimac"""
    suma = 0
    @staticmethod
    def status():
        print("\nUkupan broj ljubimaca je", Ljubimac.suma)

    def __init__(self, ime):
        print("Nov ljubimac je rodjen!")
        self.ime = ime
        Ljubimac.suma += 1

Ljubimac.status()
print("\nPristupanje atributu klase Ljubimac.suma:", end=" ")
print(Ljubimac.suma)
```

```
print("\nKreiranje ljubimca.")
mali1 = Ljubimac("mali1")
mali2 = Ljubimac("mali2")
mali3 = Ljubimac("mali3")
Ljubimac.status()
print("\nPristupanje atributu klase kroz objekat:", end= " ")
print(mali1.suma)
```

```
Ukupan broj ljubimaca je 0

Pristupanje atributu klase Ljubimac.suma: 0

Kreiranje ljubimca.
Nov ljubimac je rodjen!
Nov ljubimac je rodjen!
Nov ljubimac je rodjen!

Ukupan broj ljubimaca je 3

Pristupanje atributu klase kroz objekat: 3
```

010 Kreiranje atributa klase

Linijom: `suma = 0`
unutar definicije klase, kreira se atribut klase suma i njoj se dodeljuje vrednost 0. Bilo koji drugi iskaz nalik ovom, kao nova promenjiva kojoj se dodeljuje vrednost izvan metode, kreira atribut klase. Iskz dodele se realizuje samo jednom, kada Pajton po prvi put vidi definiciju klase. To znači da atribut klase postoji i pre nego se kreira prvi objekat. Zato, atribut klase se može koristiti bez postojanja bilo kojeg objekta klase.

011 Pristupanje atributu klase

U programu se pristupa atributu klase na nekoliko različitih mesta. U glavnom delu programa štampa se sa: `print(Ljubimac.suma)`
U metodi statik, status(), štampa se vrednost atributa klase suma sa linijom:
`print("\nUkupan broj ljubimaca je", Ljubimac.suma)`
U metodi konstruktora, inkrementira se vrednost atributa klase u liniji:
`Ljubimac.suma += 1`
Kao rezultat ove linije, svaki put kada se instancira nov objekat, vrednost atributa klase se inkrementira.
U principu, za pristup atributu klase, koristi se dot notacija. Ukuca se ime klase, tačka, ime atributa klase.
Na kraju, može se pristupiti atributu klase kroz objekat klase. To se i dešava u glavnom delu programa:
`print(mali1.suma)`
Ova linija štampa vrednost atributa klase suma (a ne neki atribut samog objekta). Može se pročitati vrednost atributa klase kroz bilo koji objekat koji pripada toj klasi. To znači da je moglo da se napiše `print(mali2.suma)` ili `print(mali3.suma)` i dobio bi se isti rezultat.
Iako se može koristiti objekat klase za pristup atributu klase, ne može se dodeliti nova vrednost atributu klase kroz objekat. Ako se želi promeniti vrednost atributa klase, treba mu pristupti pomoću imena klase.

012 Kreiranje metode static

Prva metoda u klasi je status():

```
def status():
        print("\nUkupan broj ljubimaca je", Ljubimac.suma)
```
Definicija je napisana kao deo kreiranja metode static. Vidi se da definicija nema self u svojoj listi parametara. To je zato, što kod svih metoda static, dizajnirana je da se podigne kroz klasu a ne kroz objekat. To znači, da metod neće biti podignut kao referenca objekta i zato mu ne treba parametar poput self da bi dobio referencu. Metoda static može imati parametre, ali u ovom programu se ne koriste.

Dok definicija kreira metodu status, postavljen je **dekorator**, nešto što dopunjuje ili modifikuje funkciju ili metodu, odmah posle definicije. Ovaj dekorator kreira metodu static istog imena:

```
@staticmethod
```

Kao rezultat ove tri linije koda, klasa ima metodu static (status()), koja prikazuje ukupan broj Ljubimac objekata štampanjem atributa klase suma.

Da bi programer napravio svoje metode static, najbolje je da prati ovu šemu. Prvo se upiše @staticmethod dekorator, zatim definicija metode klase. Pošto metoda važi za celu klasu, neće se uključivati self parametar, koji je neophodan samo za metode objekata.

013 Podizanje metode static

U programu se podizanje metode static izvodi u prvoj liniji glavnog dela programa:

```
Ljubimac.status()
```

Linija daje 0 na ekranu, pošto nijedan objekat još nije instanciran. Ali, vidi se da je moguće podići metod bez postojanja ijednog objekta. Pošto su metode static podignute kroz klasu, nijedan objekat klase ne mora postojati pre nego se podigne metoda static.

Zatim se kreiraju tri objekta pa se posle ponovo podigne metoda static. Dobija se poruka o tri kreirana ljubimca. Sada je tokom izvršena metode konstruktora za svaki objekat, atribut klase tri puta inkrementiran.

Enkapsulacija objekata

Do sada se videlo da su funkcije enkapsulirane i sakriven je njihov unutrašnji kod od dela programa koji ih poziva (zove se klijent funkcije). Takođe se videlo da klijent dobro definisane funkcije komunicira sa funkcijom samo preko njenih parametara i povratnih vrednsoti. Uopšteno, objekat bi trebalo da se ponaša na isti način. Klijent treba da komunicira sa objektom kroz parametre metoda i povratne vrednosti. To znači, kod klijenta bi trebalo da izbegava direktnu modifikaciju vrednosti atributa objekata.

Privatni atributi i privatne metode

Po difoltu, svi atributi i metode objekata su **javni** (public), a to znači da im se može direktno pristupiti ili da se mogu podići od strane klijenta. Uključivanjem enkapsulacije, može se definisati atribut ili metod kao **privatan** (private) a to znači da samo druge metode istih objekata mogu da im pristupe ili da ih podignu.

program Privatni_ljubimac.py

```
#Privatni_ljubimac
#Prikazuje privatne atribute i metode

class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime, raspolozenje):
        print("Novi ljubimac je rodjen!")
        self.ime = ime                          #javni atribut
        self.__raspolozenje = raspolozenje  #privatan atribut

    def prica(self):
```

```
        print("\nJa sam", self.ime)
        print("Trenutno se osecam", self.__raspolozenje, "\n")

    def __privatan_metod(self):
        print("Ovo je privatan metod.")

    def javni_metod(self):
        print("Ovo je javni metod.")
        self.__privatan_metod()

mali = Ljubimac(ime = "Puk", raspolozenje = "pospan")
mali.prica()
mali.javni_metod()
```

```
Novi ljubimac je rodjen!

Ja sam Puk
Trenutno se osecam pospan

Ovo je javni metod.
Ovo je privatan metod.
```

014 Kreiranje privatnih atributa

Da bi se sprečio direktan pristup atributima objekta od strane klijenta, koriste se privatni atributi. U metodi konstrukta, kreirana su dva atributa, jedan javni i jedan privatni:

```
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime, raspolozenje):
        print("Novi ljubimac je rodjen!")
        self.ime = ime                          #javni atribut
        self.__raspolozenje = raspolozenje  #privatan atribut
```

U Pajtonu, kada se započne ime atributa sa dve donje linije zna se da je to privatan atribut. Na taj način se mogu napraviti i privatni atributi klase i privatni atributi objekta.

015 Pristup privatnim atributima

Sasvim je lako pristupiti privatnim atributima unutar definicije klase nekog objekta (zapamtiti da treba obeshrabriti kod klijenta da direktno pristupi atributima objekta). U programu se pristupa privatnim atributima kroz metod prica():

```
def prica(self):
        print("\nJa sam", self.ime)
        print("Trenutno se osecam", self.__raspolozenje, "\n")
```

Ovaj metod štampa vrednost privatnog atributa objekta, koji predtsavlja raspoloženje ljubimca.

Ako se pokuša pristupiti atributu izvan definicije klase Ljubimac, neće se to moći ostvariti:

```
 >>> sitan = Ljubimac(ime = "Puk", raspolozenje = "sretan")
Novi ljubimac je rodjen!
>>> print(sitan.raspolozenje)
Traceback (most recent call last):
        File "<pyshell#1>", line 1, in <module>
                print(sitan.raspolozenje)
AttributeError: 'Ljubimac' object has no attribute 'raspolozenje'
```

Podizanjem AttributeError izuzetka, Pajton ukazuje da taj atribut ne postoji. Da li to znači da vrednost privatnog atributa je potpuno nedostupna izvan definicije njegove klase? Ne, jer

Pajton sakriva atribute kroz posebnu konvenciju imenovanja, iako je tehnički još uvek moguće prići atributu. To si i radi u sledećoj liniji koda:

```
>>> print(sitan._Ljubimac__raspolozenje)
sretan
```

Ova linija štampa vrednost privatnog atributa, koja je u ovom slučaju sretan.

Pošto je tehnički moguće pristupiti privatnim atributima, postavlja se pitanje čemu onda privatni atributi? U Pajtonu, privatnost je indikator da atribut ili metod su namenjeni samo za internu upotrebu unutar objekta. Još, to pomaže prevenciji nedopuštenog pristupa takvim atributima ili metodama.

016 Kreiranje privatnih metoda

Privatn emetode se mogu napraviti na isti jednostavan način kao i privatni atributi: dodavanjem dve donje linije ispred imena metode:

```
def __privatan_metod(self):
        print("Ovo je privatan metod.")
```

Ovo je privatan metod ali mu se lako može pristupiti preko bilo kojeg drugog metoda iz klase. Poput privatnih atributa, privatne metode su namenjene samo pristupu preko metoda samog objekta.

017 Pristupanje privatnim metodama

Poput privatnih atributa, pristup privatnim metodama objekta unutar njegove definicije klase je jednostavno. U javni_metod() metodi, pristupa se privatnoj metodi klase:

```
def javni_metod(self):
        print("Ovo je javni metod.")
        self.__privatan_metod()
```

Ovaj metod štampa string a zatim podiže privatnu metodu objekta.

Poput privatnih atributa, privatne metode nisu namenjene za direktan pristup od strane klijenata. Ovo je pokušaj pristupa privatnom metodu objekta mali:

```
>>> mali.privatan_metod
Traceback (most recent call last):
        File "<pyshell#0>", line 1, in <module>
                mali.privatan_metod
AttributeError: 'Ljubimac' object has no attribute 'privatan_metod'
```

Ovaj pokušaj podiže Attributeerror izuzetak. Pajton ukazuje da mali nema metod sa tim imenom. Pajton sakriva metod kroz istu, posebnu konvenciju imenovanja. Ako se opet pokuša dodati dve donje linije ispred imena metoda, javlja se ista greška:

```
>>> mali.__privatan_metod()
Traceback (most recent call last):
        File "<pyshell#0>", line 1, in <module>
                mali.__privatan_metod
AttributeError: 'Ljubimac' object has no attribute '__privatan_metod'
```

Ipak, kao i sa privatnim atributima, tehnički je moguće pristupiti privatnim metodama iz bilo koje tačke programa:

```
>>> mali._Ljubimac__privatan_metod()
Ovo je privatan metod.
```

Klijent ne bi trebalo nikad da pokuša direktno pristupiti privatnim metodama objekta.

Može se napraviti privatan metod static kada se na početku imena metode stave dve donje crte.

018 Poštovanje privatnosti objekta

U glavnom delu programa, u kodu nije korišćena mogućnost direktnog pristupa privatnim atributima i metodama. Umesto toga, kreiran je objekat i podignute njegove dve javne metode:

```
mali = Ljubimac(ime = "Puk", raspolozenje = "pospan")
mali.prica()
mali.javni_metod()
```

Metod __init__() Ljubimac objekta, koji se automatski poziva odmah posle kreiranja objekta, daje poruku da je novi ljubimac rođen. Metoda prica() objekta mali ukazuje kako se ljubimac oseća. Metoda javni_metod() objekta mali štampa string “Ovo je javan metod.“ a zatim podiže privatni metod objekta koji štampa “Ovo je privatan metod.“

019 Kada primeniti privatnost

Da li je neophodno da svaki atribut u svakoj klasi bude privatan da bi se zaštitio od spoljnog koda? Ne, Pajton predlaže da se privatnim učine samo one metode za koje se želi da klijent ih ne može da podigne. Ako je važno da atribut nikada ne bude direktno pristupljen od strane klijenta, on treba da bude privatan. Filozofija Pajton programera je da veruju da će klijen koristiti metode objekta a ne direktno pristupiti njegovim atributima.

Kada se piše klasa:

- Kreirati metode kojima se smanjuje potreba klijenta da direktno pristupi atributima objekta
- Koristiti privatnost za one atribute i metode koji su potpuno interni za operacije na objektu

Kada se koristi objekat:

- Minimizirati direktno čitanje atributa objekta
- Izbegavati direktno menajnje atributa objekta
- Nikad ne pokušavati direktan pristup privatnim atributima ili metodama objekta

Kontrola pristupa atributima

Ponekad umesto potpunog blokiranja pristupa nekom atributu, lakše je samo limitirati pristup. Npr, možda postoji atribut kojem je potrebno da klijent kod pristupi, ali ne i da ga menja. Pajton omogućava nekoliko alata za realizaciju ovog postupka, uključivanjem **svojstava** (properties). Svojstva omogućavaju kontrolu kako se atributu može prići ili kako se atribut može promeniti.

program Svojstva_ljubimac.py

```
#Svojstva_ljubimac
#Prikazuje svojstva (properties)
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime):
        print("Novi ljubimac je rodjen!")
        self.__ime = ime

    @property
    def ime(self):
        return self.__ime

    @ime.setter
    def ime(self, novo_ime):
        if novo_ime == "":
            print("Ime ljubimca ne moze biti prazan string.")
```

```
        else:
            self.__ime = novo_ime
            print("Ime uspesno promenjeno.")

    def prica(self):
        print("\nZdravo, ja sam", self.ime)

mali = Ljubimac("Puk")
mali.prica()
print("\nIme mog ljubimca je:", end= " ")
print(mali.ime)
print("\nPokusavam da promenim ime mog ljubimca u Miki...")
mali.ime = "Miki"
print("Ime mog ljubimca je:", end= " ")
print(mali.ime)
print("\nPokusavam da promenim ime mog ljubimca u prazan string...")
mali.ime = ""
print("Ime mog ljubimca je:", end= " ")
print(mali.ime)
```

```
Novi ljubimac je rodjen!

Zdravo, ja sam Puk

Ime mog ljubimca je: Puk

Pokusavam da promenim ime mog ljubimca u Miki...
Ime uspesno promenjeno.
Ime mog ljubimca je: Miki

Pokusavam da promenim ime mog ljubimca u prazan string...
Ime ljubimca ne moze biti prazan string.
Ime mog ljubimca je: Miki
```

020 Kreiranje svojstava

Jedan od načina da se kontroliše pristup privatnim atributima je kreiranje **svojstva** – to je objekat sa metodama koje omogućavaju indirektan pristup atributima i često čine neku vrstu ograničenja tom pristupu. Posle kreiranja konstruktora klase Ljubimac, kreirano je svojstvo ime kojim se omogućava indirektan pristup privatnom atributu __ime:

```
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime):
        print("Novi ljubimac je rodjen!")
        self.__ime = ime

    @property
    def ime(self):
        return self.__ime
```

Kreirano je svojstvo pisanjem metoda koji vraća kao vrednost atribut kojem se traži indirektan pristup (u ovom slučaju to je __ime) a pre svega toga je naveden dekorator @property. Svojstvo ima isti naziv kao i metod, u ovom slučaju, ime. Sada se može koristiti ime svojstvo bilo kojeg Ljubimac objekta za dobijanje vrednosti privatnog atributa __ime, unutar ili izvan definicije klase korišćenjem dot notacije.

Da bi se samostalno napravilo svojstvo, treba napisati metod koji vraća vrednost kojom se želi omogućiti indirektan pristup a pre toga napisati dekorator @property. Svojstvo će imati isti naziv kao i metod. Ako se omogućava pristup privatnom atributu, po konvenciji je dati svojstvu isti naziv koji ima i privatni atribut samo bez vodećih donjih crtica.

Posle kreiranja svojstva, može se omogućiti pristup čitanju privatnog atributa. Ipak, svojsvo može takođe omogućiti pristup i upisivanju – čak i nametnuti neka ograničenja tom pristupu. U programu je to urađeno za privatni atribut __ime kroz svojstvo ime:

```
@ime.setter
    def ime(self, novo_ime):
        if novo_ime == "":
            print("Ime ljubimca ne moze biti prazan string.")
        else:
            self.__ime = novo_ime
            print("Ime uspesno promenjeno.")
```

Kod započinje sa dekoratorom @ime.setter. Priatupanjem setter atributu svojstva ime, nagoveštavam da sledeća definicija metoda će omogućiti način za postavljanje vrednosti svojstva ime. Može se napraviti sopstveni dekorator za postavljanje vrednosti svojstva prema sledećem primeru: početi sa @ simbolom, pa naziv svojstva, zatim tačka pa reč setter.

Posle dekoratora, treba definisati metod sa nazivom ime koji se poziva kada klijent kod pokuša postaviti novu vrednost korišćenjem svojstva. Ovaj metod se naziva ime kao i svojstvo, i tako mora biti. Pri uspostavljanju setter metode na ovaj način, metod mora imati isti naziv kao odgovarajuće svojstvo.

U metodi, parametar novo_ime dobija vrednost za novo ime ljubimca. Ako je vrednost prazan string, __ime ostaje nepromenjeno a poruka se javlja o neuspelom pokušaju promene imena ljubimca. U drugom slučaju, metod postavlja privatni atribut __ime na novo _ime i prikazuje poruku da je promena uspela. Kada se kreira metoda za postavljanje vrednosti svojstva, definicija metode mora imati parametar za primanje nove vrednosti.

021 Pristup svojstvu

Kreiranjem svojstva ime, može se dobiti ime ljubimca preko dot notacije:

```
def prica(self):
        print("\nZdravo, ja sam", self.ime)

mali = Ljubimac("Puk")
mali.prica()
```

Kod self.ime pristupa svojstvu ime i indirektno poziva metod koji vraća __ime. U ovom slučaju, vraća string “Puk“. Ne samo da se može koristiti svojstvo ime objekta unutar definicije klase, već se može koristiti i izvan definicije klase:

```
print("\nIme mog ljubimca je:", end= " ")
print(mali.ime)
```

Iako je ovaj kod izvan klase Ljubimac, dešava se u suštini ista stvar – kod mali.ime pristupa svojstvu ime objekta Ljubimac i indirektno poziva metod koji vraća __ime. U ovom slučaju, još uvek vraća string “Puk“.

Dalje, pokušana je promena imena ljubimca:

```
print("\nPokusavam da promenim ime mog ljubimca u Miki...")
mali.ime = "Miki"
```

Iskaz dodele mali.ime = “Miki“ pristupa svojstvu ime objekta i indirektno poziva metod koji pokušava postaviti __ime. U ovom slučaju, ukazivanje na string “Miki“ se dostavlja novo_ime parfametru metode, a pošto “Miki“ nije prazan string, objektov atribut __ime postaje “Miki“ a poruka o uspešnoj promeni imena se štampa.

Dalje, prikazuje se ime ljubimca korišćenjem svojstva ime:

```
print("Ime mog ljubimca je:", end= " ")
print(mali.ime)
```

Program prikazuje: Ime mog ljubimca je: Miki.

Sledeće, pokušava se promeniti ime ljubimca u prazan string:

```
print("\nPokusavam da promenim ime mog ljubimca u prazan string...")
mali.ime = ""
```

Kao i pre, iskaz pristupa svojstvu ime objekta i indirektno poziva metod koji pokušava postaviti __ime. U ovom slučaju, ukazivanje na prazan string je dostavljeno u parametar novo_ime. Kao rezultat, metod prikazuje poruku Ime ljubimca ne moze biti prazan string a objektov __ime atribut ostaje nepromenjen.

program Ljubimac.py

```
#Ljubimac
#Virtuelni ljubimac
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime, glad = 0, dosada = 0):
        self.ime = ime
        self.glad = glad
        self.dosada = dosada

    def __tokom_vremena(self):
        self.glad += 1
        self.dosada += 1

    @property
    def raspolozenje(self):
        nesreca = self.glad + self.dosada
        if nesreca < 5:
            m = "sretan"
        elif 5 <= nesreca <= 10:
            m = "u redu"
        elif 11 <= nesreca <= 15:
            m = "frustriran"
        else:
            m = "ljut"
        return m

    def prica(self):
        print("Ja sam", self.ime, "i osecam se", self.raspolozenje, "sada.\n")
        self.__tokom_vremena()

    def hraniti(self, hrana = 4):
        print("Burp. Hvala.")
        self.glad -= hrana
        if self.glad < 0:
            self.glad = 0
        self.__tokom_vremena()

    def igrati(self, zabava = 4):
        print("Jupi!")
        self.dosada -= zabava
        if self.dosada < 0:
            self.dosada = 0
        self.__tokom_vremena()

def main():
    mali_ime = input("Kakvo ime dajes svom ljubimcu?: ")
    mali = Ljubimac(mali_ime)

    izbor = None
    while izbor != "0":
        print \
        ("""
```

```
        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem
        """)
        izbor = input("Izbor: ")
        print()
        if izbor == "0":
            print("Dovidjenja.")
        elif izbor == "1":
            mali.prica()
        elif izbor == "2":
            mali.hraniti()
        elif izbor == "3":
            mali.igrati()
        else:
            print("\nZao mi je", izbor, "nije dozvoljena opcija.")

main()
```

```
Kakvo ime dajes svom ljubimcu?: Puk

        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem

Izbor: 1

Ja sam Puk i osecam se sretan sada.


        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem
Izbor: 2

Burp. Hvala.

        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem

Izbor: 3

Jupi!

        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem
```

```
Izbor: 1

Ja sam Puk i osecam se sretan sada.


        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem

Izbor: 1

Ja sam Puk i osecam se u redu sada.


        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem
Izbor: 0

Dovidjenja.
```

Klasa Ljubimac

Klasa Ljubimac je plan izgradnje za objekat koji predstavlja korisnikovog ljubimca. Klasa nije komplikovana, i njen najveći deo je prepoznatljiv.

022 Metod konstruktor

Metod konstruktor klase inicijalizuje tri javna atributa objekta Ljubimac: ime, glad, dosada. Glad i dosada imaju i difolt vrednost jednaku 0, čime se omogućava ljubimcu da startuje igru u dobrom raspoloženju:

```
class Ljubimac(object):
    """Virtuelni ljubimac"""
    def __init__(self, ime, glad = 0, dosada = 0):
        self.ime = ime
        self.glad = glad
        self.dosada = dosada
```

Ovaj metod oslikava više opušteniju verziju Pajton programiranja pošto se ostavljaju atributi u njihovom difolt javnom statusu. Ideja je da se obezbede sve metode koje korisniku mogu zatrebati, što će ohrabriti klijent kod da interaguje sa Ljubimcem objektom samo kroz ove metode.

023 Metoda __tokom_vremena()

Ova metoda je privatna metoda koja povećava nivoe gladi i dosade ljubimca. Podiže se na kraju svake metode kada ljubimac uradi nešto (jede, igra se ili priča) da bi simuliralo protok vremena. Ova metoda je napravljena kao privatna pošto bi trebalo da se podigne samo pomoću druge metode klase.

```
def __tokom_vremena(self):
        self.glad += 1
        self.dosada += 1
```

024 Svojstvo raspolozenje

Ovo svojstvo predstavlja trenutno raspoloženje ljubimca. Sledeći metod izračunava raspoloženje. Dodaje se vrednost glad i dosada atributa Ljubimac objekta i na osnovu sume, vraća string koji govori o raspoloženju: “sretan“, “u redu“, “frustriran“ i “ljut“.
Važna stavka ovog svojstva je da ne omogućava pristup privatnom atributu. To je zato što string koji predstavlja raspoloženje ljuibimca nije smešten kao dao Ljubimac objekta, nego je izračunat u toku izvršavanja koda. Svojstvo rasplozenje samo daje pristup stringu koji se vraća preko metode.

```
@property
    def raspolozenje(self):
        nesreca = self.glad + self.dosada
        if nesreca < 5:
            m = "sretan"
        elif 5 <= nesreca <= 10:
            m = "u redu"
        elif 11 <= nesreca <= 15:
            m = "frustriran"
        else:
            m = "ljut"
        return m
```

025 Metod prica()

Metod prica() objavljuje raspoloženje ljubimca pristupanjem svojstvu raspolozenje objekta Ljubimac. Zatim podiže __tokom_vremena():

```
def prica(self):
        print("Ja sam", self.ime, "i osecam se", self.raspolozenje, "sada.\n")
        self.__tokom_vremena()
```

026 Metod hraniti()

Ovaj metod smanjuje nivo gladi ljubimca za iznos koji je pridodat parametru hrana. Ako nikakva vrednost nije pridodata, hrana dobija difolt vrednost 4. Nivo gladi ljubimca se kontroliše i ne sme otići ispod 0. Na kraju metod podiže __tokom_vremena().

```
def hraniti(self, hrana = 4):
        print("Burp. Hvala.")
        self.glad -= hrana
        if self.glad < 0:
            self.glad = 0
        self.__tokom_vremena()
```

027 Metod igrati()

Ovaj metod smanjuje nivo dosade ljubimca za iznos dodeljen parametru zabava. Ako se ne doda nikakava vrednost, zabava dobija difolt vrednsot 4. Nivo dosade ljubimca se kontroliše i nije dopušteno da padne ispod 0. Na kraju metod podiže __tokom_vremena().

```
def igrati(self, zabava = 4):
        print("Jupi!")
        self.dosada -= zabava
        if self.dosada < 0:
            self.dosada = 0
        self.__tokom_vremena()
```

028 Kreiranje ljubimca

Glavni deo programa je u sopstvenoj funkciji, main(). Na početku programa, dobija se ime ljubimca od strane korisnika. Dalje, instancira se nov objekat Ljubimac. Pošto se ne dodeljuju vrednosti za glad i dosada, atributi započinju od 0 i ljubimac startuje sretan i zadovoljan.

```
def main():
    mali_ime = input("Kakvo ime dajes svom ljubimcu?: ")
    mali = Ljubimac(mali_ime)
```

029 Kreiranje menija

Kreiran je poznat meni. Ako korisnik unese 0, program se završava. Ako korisnik unese 1, objekatova metoda prica() se podiže. Ako se unese 2, objektova metoda hraniti() se podiže. Ako se unese 3, objektova metoda igrati() se podiže. Ako korisnik unese bilo šta drugo, pojavljuje se poruka o lošem unosu.

```
    izbor = None
    while izbor != "0":
        print \
        ("""
        Ljubimac

        0 - Izlaz
        1 - Poslusaj svog ljubimca
        2 - Nahrani svog ljubimca
        3 - Igraj se sa svojim ljubimcem
        """)
        izbor = input("Izbor: ")
        print()
        if izbor == "0":
            print("Dovidjenja.")
        elif izbor == "1":
            mali.prica()
        elif izbor == "2":
            mali.hraniti()
        elif izbor == "3":
            mali.igrati()
        else:
            print("\nZao mi je", izbor, "nije dozvoljena opcija.")
```