Programska potpora komunikacijskim sustavima

Dr. sc. Adrian Satja Kurdija

Programski jezik Python - 4. predavanje (Objektno orijentirano programiranje)

SVEUČILIŠTE U ZAGREBU
Fakultet elektrotehnike i računarstva

Sadržaj predavanja

- Moduli
- Paketi
- Klase
- Principi objektno orijentiranog programiranja:
  - nasljeđivanje
  - polimorfizam
  - enkapsulacija
  - apstrakcija

Moduli, paketi

Moduli

- Mnogi Python programi započinju import naredbama
- Time se uključuje neki drugi Python kod ("naš" ili "tuđi")
- Sintaksa:

```
import moj_modul
```

--> objekte dohvaćamo s prefiksom (`moj_modul.funkcija`)

```
from moj_modul import *
from moj_modul import funkcija2, klasa3
```

- Moduli standardne biblioteke: *os, sys, math, pickle, time, collections...*
  - instalirani zajedno s Python interpreterom
- Korištenje kraćeg imena:

```
import numpy as np
```

Paketi

- Modul je neka .py datoteka; paket je direktorij modula
- Sintaksa:

```
import paket.modul
import scipy.stats
scipy.stats.variation(a)

from paket import modul
from scipy import stats
stats.variation(a)

from paket.modul import objekt
from scipy.stats import variation
variation(a)
```

Paketi

- Instalacija paketa iz Python Package Index (PyPI):

```
pip install ime_paketa
pip install ime_paketa==verzija
pip install --upgrade ime_paketa
```

- Provjera instaliranih paketa:

```
pip freeze
pip show ime_paketa
```

- Deinstalacija paketa:

```
pip uninstall ime_paketa
```

Import putanje

- Import će uspjeti ako se direktorij modula/paketa nalazi u nizu sys.path
- To će biti zadovoljeno:
  - ako je modul dio standardne biblioteke
  - ako je instaliran nekim od standardnih alata (npr. pip, apt)
  - ako se nalazi u istom direktoriju kao i pokrenuti program (kao dio istog projekta)
- Ako se nalazi negdje drugdje, možemo ažurirati niz sys.path:

```
import sys
print(sys.path)
sys.path.append('/home/adrian/my_helper_modules')
import my_module
```

Import putanje

- Tipična situacija:
  - glavni program je *main.py*
  - pomoćni kodovi nalaze se u poddirektorijima (npr. *data_models/myClass.py*) istog direktorija

```
from data_models.myClass import ImeKlase
```

(unutar *main.py*)

Vježba

- U modulu *pomocni.py* napisati funkciju `median(x, y, z)` koja vraća srednji po veličini od triju brojeva.
- Pozvati funkciju iz programa *glavni.py* u istom direktoriju/projektu.
- Modul premjestiti u neki poddirektorij istog projekta.
- Modul premjestiti u neki direktorij izvan projekta (npr. Desktop).

Klase

Klase

- Klasa je skup organiziranih podataka i operacija nad tim podatcima
  - npr. račun u trgovini
- Konkretan primjerak klase je **objekt**
- Podatci unutar klase zovu se **atributi**

```
objekt.ime_atributa
moj_racun.artikli
```

- Funkcije unutar klase zovu se **metode**

```
objekt.ime_funkcije(...)
moj_racun.dodaj_artikl(bajadera)
```

Klase

- Definiranje klase:
  - Ključna riječ `class` + ime klase + dvotočka

```
class Trokut:
    ...
```

  - Izvode se unutrašnje naredbe, obično definicije atributa i metoda
- Stvaranje objekta:

```
ime_objekta = ImeKlase(...parametri...)
```

--> poziva se metoda `__init__` unutar klase koja prima navedene parametre (nula ili više njih) te postavlja atribute klase na početne vrijednosti

Primjer klase

```
class Zaposlenik:
  def __init__(self, ime_i_prezime, placa):
    self.ime, self.prezime = ime_i_prezime.split()
    self.placa = placa

  def daj_povisicu(self, postotak):
    self.placa = self.placa * (1 + postotak / 100)

  def porez(self):
    return 0.24 * self.placa

ana = Zaposlenik("Ana Anic", 9000)
ana.daj_povisicu(15)
print(ana.porez())
```

Parametar *self*

- Članske metode na prvom mjestu trebaju imati definiran parametar imena *self*
- On je referenca na objekt unutar kojeg se izvodi metoda
- Pruža pristup metodama i atributima trenutnog objekta, a ne nekog drugog objekta
- Kad pozivamo metodu, ne zadajemo vrijednost za parametar *self* već interpreter automatski definira tu referencu:

```
objekt.metoda(podatak1, podatak2)
```

prevodi se u

```
metoda(self=objekt, podatak1, podatak2)
```

Neobavezni (*optional*) parametri

- Parametri koji se prilikom poziva metode ne moraju zadati
- Funkciji se onda prosljeđuju zadane *default* vrijednosti

```
def __init__(self, ime_i_prezime, placa=6000):
    ...
```

- Atribut se može definirati/promijeniti poslije

```
pero = Zaposlenik("Pero Peric")
pero.placa = 7000
print(hasattr(pero, 'placa'))  # ispis: True
```

Statički atributi i metode

- Statička metoda ne odnosi se na određeni objekt
- Definira se bez parametra `self`
- Poziva se kao `ImeKlase.imeMetode(...)`
- Statički atributi ne odnose se na određeni objekt, nego na klasu općenito
- Definiraju se bez `self`-a, ispod definicije klase, izvan svih metoda
- Pristupamo im kao `ImeKlase.imeAtributa`

Statički atributi i metode

- Primjer: klasi *Zaposlenik* dodajmo statičku metodu `izbroji()` koja vraća broj objekata - konkretnih zaposlenika

```
class Zaposlenik:
    brojac = 0
    def __init__(self, ime_i_prezime, placa):
        Zaposlenik.brojac += 1
        ...
    def izbroji():
        return Zaposlenik.brojac
...
print(Zaposlenik.izbroji())
```

Posebne metode klase

- Python magic methods - ne pozivaju se eksplicitno, nego automatski u određenim slučajevima (ako ih definiramo)
- Predefinirana imena oblika `__metoda__`
- Najčešći primjeri:
- `__init__`
- `__str__`
- `__add__, __mul__, __sub__, __div__`
- `__cmp__, __eq__, __ne__, __lt__, __gt__`
- `__getitem__, __setitem__`

Vježba

- Krenimo od klase *Zaposlenik* kao što je definirana na jednom od prethodnih slajdova.
- Zadavanjem *default* plaće u `__init__` funkciji omogućimo stvaranje zaposlenika definirajući mu samo ime i prezime.
- Dodatno ostvarimo da, u slučaju plaće koja nije zadana, uopće ne definiramo odgovarajući atribut.
- Klasi Zaposlenik dodajmo statičku metodu `izbroji()` koja vraća broj objekata - konkretnih zaposlenika.
- Implementirati metodu `__str__` tako da npr. poziv `print(ana)` ispiše ime, prezime i plaću zaposlenika.

Principi objektno orijentiranog programiranja

Nasljeđivanje

- U zagradi je moguće definirati nadklasu (ili više njih):

```
class Zaposlenik(Osoba):
```

- Tada podklasa nasljeđuje sve atribute i metode nadklase (uz vlastite)
- Klasa *Zaposlenik* nasljeđuje i proširuje klasu *Osoba*
  - svaki *Zaposlenik* je *Osoba* (obrat ne vrijedi)

Nasljeđivanje: primjer (1/2)

```
class Poligon:
  def __init__(self, broj_stranica):
    self.n = broj_stranica

  def unesi_duljine_stranica(self):
    self.stranice = [float(input()) for i in range(self.n)]

  def opseg(self):
    return sum(self.stranice)

p = Poligon(5)
p.unesi_duljine_stranica()
print(p.opseg())
```

Nasljeđivanje: primjer (2/2)

```
class Trokut(Poligon):
  def __init__(self):
    self.n = 3
    # alternative:
    # Poligon.__init__(self, 3)
    # super().__init__(self, 3)

  def povrsina(self):
    a, b, c = self.stranice
    s = (a + b + c) / 2
    return (s*(s-a)*(s-b)*(s-c)) ** 0.5

t = Trokut()
t.unesi_duljine_stranica()      # naslijeđena metoda!
print(t.opseg())                # naslijeđena metoda!
print(t.povrsina())
```

Vježba

- Nasljeđivanjem omogućiti da neki zaposlenici budu studenti, kojima se porez računa kao 10% umjesto 24% od plaće.
- Napisati drugu klasu *Poduzece* čiji su atributi ime i lokacija poduzeća te lista zaposlenika. Klasa treba metodama podržati:
  - dodavanje novog zaposlenika,
  - davanje povišice svim zaposlenicima za zadani postotak,
  - računanje ukupnog poreza na plaće zaposlenika,
  - izbacivanje zaposlenika.
- Implementirati metodu `__str__` koja omogućuje da npr. poziv `print(moje_poduzece)` ispiše podatke o poduzeću i zaposlenicima.

Polimorfizam

- Princip koji omogućuje da neku klasu koristimo kao i njezinu nadklasu
- Objekti podklase formalno pripadaju i nadklasi
- Objekt klase *Student* može bilo gdje zamijeniti objekt klase *Zaposlenik* i odigrati njegovu ulogu jer je naslijedio sve njegove metode
- Npr. imamo listu zaposlenika poduzeća u kojoj su i neki studenti
- Svaku metodu koju treba pozvati na svim zaposlenicima (npr. davanje povišice) moguće je izvesti i na onima koji su studenti

```
branko = Student("Branko Brankic", 8000)
print(type(branko))       # <class '__main__.Student>
print(isinstance(branko, Zaposlenik))  # True
```

Enkapsulacija

- **Enkapsulacija**: "skrivamo" atribute klase, smatramo ih privatnim i ne pristupamo im izvan tijela klase
- Stavljamo dvije donje crte ispred imena (npr. `self.__placa`)
- Smanjuje mogućnost pogreške
- Smanjuje međuovisnosti različitih komponenata
- Metode također mogu biti privatne - pozivaju se samo interno

Apstrakcija

- **Apstrakcija**: javne su samo one metode koje su nam potrebne da izvana koristimo klasu
- Primjer:

```
class Zaposlenik:
    ...
    def __obavijesti_racunovodstvo(self):
        ...
    def postavi_placu(self, vrijednost):
        self.__placa = vrijednost
        self.prirez = 0.18 * self.__placa
        self.__obavijesti_racunovodstvo()
```