Основне академске студије
Информациони инжењеринг

Основи рачунарске интелигенције

Парадигме програмирања у језику и окружењу Python

(материјали за предавања)

Садржај

Увод

Основне парадигме програмирања у језику *Python*

- императивно програмирање
  - процедурално програмирање
  - објектно програмирање
- декларативно програмирање
  - функционално програмирање

Садржај

Процедурално програмирање

- програми организовани кроз процедуре
  - процедура као група програмских наредби која има посебну сврху
  - подршка у језику *Python*
    - кроз податке
      - концепт објекта у језику Python
      - уграђене структуре података
    - кроз функције
    - кроз модуле и пакете
      - велики број уграђених модула
        - конкурентно програмирање
        - обрада текста
        - рад с датотекама
        - рад с маркап форматима (*SGML*, *HTML* и *XML*)
        - коришћење Интернет технологија
        - развој графичког корисничког интерфејса
        - …

Садржај

Објектно програмирање

Објектно оријентисани концепти у језику *Python*

- објекат
- класа
  - атрибут инстанце
  - метода инстанце
- односи између класа
  - наслеђивање
  - агрегација
  - композиција

Објектно програмирање

Објекат

- подаци који се користе у програмима написаним у језику *Python* представљени су као објекти
  - инстанце уграђених класа
    - класе **`int`**, **`str`**, **`dict`**, **`statistics.NormalDist`**, **`csv.DictReader`**, **`urllib.request.Request`**, ...
  - инстанце корисничких класа
    - класе које по потреби додатно дефинишу корисници језика

Објектно програмирање

Класа

- спецификација структуре и понашања за објекте (инстанце класе)
  - структура описана кроз атрибуте инстанце
  - понашање описано кроз методе инстанце

Класа – дефинисање

- дефинисање помоћу резервисане речи **`class`**
  - назив класе
  - тело класе
    - обавезна примена увлачења програмског кода
    - у посебном случају класа може бити без редовног тела
      - потребан нулти исказ **`pass`**
- дефинисана класа је инстанца уграђене класе **`type`**
- у једном модулу може бити дефинисано више класа

Класа – дефинисање

```
>>> class PraznaKlasa():
    pass

>>> PraznaKlasa
<class '__main__.PraznaKlasa'>
>>> type(PraznaKlasa)
<class 'type'>
>>>
```

КОНЗОЛА

Класа – дефинисање

- методе инстанце
  - посебне функције
    - посебна функција иницијализатор **`__init__()`**
      - функција која бива позвана приликом конструисања инстанце
        - позвана након успостављања инстанце а пре враћања инстанце као резултата конструисања
        - обично се користи за постављање почетног стања инстанце
      - први параметар обично има назив **`self`** (конвенција по *PEP8*)
        - односи се на креирану инстанцу
      - може бити додатних параметара
  - корисничке функције
    - први параметар обично има назив **`self`** (конвенција по *PEP8*)
      - односи се на креирану инстанцу
    - може бити додатних параметара

Класа – дефинисање

```
>>> class Zvezda():
    def __init__(self, lat):
        self.lat = lat
    def formiraj_opis(self):
        return "Zvezda '{}'".format(self.lat)

>>> Zvezda
<class '__main__.Zvezda'>
>>>
```

КОНЗОЛА

Инстанца

- инстанцирање класе позивом конструктора
  - назив конструктора одговара називу класе
  - могуће проследити аргументе приликом позива конструктора
    - аргументи ће бити даље прослеђени функцији **`__init__()`**

Инстанца

```
>>> class Zvezda():
    def __init__(self, lat):
        self.lat = lat
    def formiraj_opis(self):
        return "Zvezda '{}'".format(self.lat)

>>> zvezda = Zvezda()
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    zvezda = Zvezda()
TypeError: __init__() missing 1 required
positional argument: 'lat'
>>> aldebaran = Zvezda("α Tauri")
>>>
```

КОНЗОЛА

Инстанца – приступање члановима

- чланови
  - атрибути инстанце и методе инстанце
- начин приступа
  - помоћу оператора `.` и задавањем назива члана
- видљивост чланова
  - чланови су доступни и изван спецификације класе
    - јавни (енгл. *public*)
  - може одредити видљивост члана применом конвенција (*PEP8*)
    - проширивањем основног назива члана префиксом `_`
      - назнака да члан није јавног карактера
        - али и даље могуће приступити члану изван спецификације класе
    - проширивањем основног назива члана префиксом `__`
      - препорука како би било избегнуто евентуално преклапање назива у случају наслеђивања класе
      - није могуће приступити члану преко његовог назива изван спецификације класе
        - али могуће приступити преко назива `_A__b`
          - `A` – назив класе, `b` – основни назив члана

Инстанца – приступање члановима

```
>>> class Zvezda():
    def __init__(self, lat, udalj, temp):
        self.lat = lat
        self._udalj = udalj
        self.__temp = temp
    def formiraj_opis(self):
        return "Zvezda {} ({} ly, {} K)".format(
self.lat, self._udalj, self.__temp)

>>> aldebaran = Zvezda("α Tauri", 65.3, 3900)
>>> aldebaran.lat
'α Tauri'
>>> aldebaran.formiraj_opis()
'Zvezda α Tauri (65.3 ly, 3900 K)'
>>>
```

КОНЗОЛА

Инстанца – приступање члановима

```
>>> class Zvezda():
    def __init__(self, lat, udalj, temp):
        self.lat = lat
        self._udalj = udalj
        self.__temp = temp
    def formiraj_opis(self):
        return "Zvezda {} ({} ly, {} K)".format(
self.lat, self._udalj, self.__temp)

>>> aldebaran = Zvezda("α Tauri", 65.3, 3900)
>>> aldebaran._udalj
65.3
>>> aldebaran.__temp
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    aldebaran.__temp
AttributeError: 'Zvezda' object has no attribute '__temp'
>>> aldebaran._Zvezda__temp
3900
>>>
```

КОНЗОЛА

Инстанца – приступање члановима

- својство (енгл. *property*)
  - механизам додатне контроле приступа преко посебних функција за приступ
    - омогућује подешавање очитавања, промене и брисања података у вези с инстанцом
  - имплементација својства помоћу класе **`property`**
    - увођење у класу члана чији је тип **`property`**
      заједно с потребним функцијама за приступ
      применом механизма декоратора
      - декоратори **`@property`**, **`@b.setter`**, **`@b.deleter`**
        - декорисане методе имају назив **`b`**
        - својство има назив **`b`**

Инстанца – приступање члановима

```
>>> class Zvezda():
    def __init__(self, lat, udalj):
        self.lat = lat
        self._udalj = udalj
    def get_udalj(self):
        return self._udalj
    def set_udalj(self, udalj):
        self._udalj = udalj
    udaljenost = property(get_udalj, set_udalj)

>>> alferac = Zvezda("α Andromedae", 97)
>>> alferac.udaljenost
97
>>> alferac.udaljenost = 97.0
>>> alferac.get_udalj()
97.0
>>>
```

КОНЗОЛА

Инстанца – приступање члановима

```
>>> class Zvezda():
    def __init__(self, lat, udalj):
        self.lat = lat
        self._udalj = udalj
    @property
    def udaljenost(self):
        return self._udalj
    @udaljenost.setter
    def udaljenost(self, udalj):
        self._udalj = udalj

>>> alferac = Zvezda("α Andromedae", 97)
>>> alferac.udaljenost
97
>>> alferac.udaljenost = 97.0
>>> alferac.udaljenost
97.0
>>>
```

КОНЗОЛА

Класа – чланови

- атрибут класе
  - уграђени атрибути
    - **`__name__`**, **`__module__`**, **`__dict__`** ...
  - кориснички атрибути
- метода класе
  - функција означена декоратором **`@classmethod`**
  - први параметар обично има назив **`cls`** (конвенција по *PEP8*)
    - односи се на класу
  - може бити додатних параметара
- статичка метода
  - функција означена декоратором **`@staticmethod`**

Класа – чланови

```
>>> class Zvezda():
    def __init__(self, lat, udalj):
        self.lat = lat
        self._udalj = udalj

>>> Zvezda.__name__
'Zvezda'
>>> Zvezda.__dict__
mappingproxy({'__module__': '__main__',
'__init__': <function Zvezda.__init__ at
0x0000000003054160>, '__dict__': <attribute
'__dict__' of 'Zvezda' objects>, '__weakref__':
<attribute '__weakref__' of 'Zvezda' objects>,
'__doc__': None})
>>>
```

КОНЗОЛА

Класа – чланови

```
>>> class Zvezda():
    lats = []
    def __init__(self, lat):
        self.lat = lat
        Zvezda.lats.append(lat)
    @classmethod
    def dobavi(cls, rev):
        return sorted(cls.lats, reverse=rev)

>>> hadar = Zvezda("β Centauri")
>>> altair = Zvezda("α Aquilae")
>>> Zvezda.dobavi(False)
['α Aquilae', 'β Centauri']
>>>
```

КОНЗОЛА

Класа – чланови

```
>>> class Zvezda():
    def __init__(self, lat, udalj):
        self.lat = lat
        self.udalj = udalj
    @staticmethod
    def ly_pc(distance):
        return distance * 0.307

>>> algol = Zvezda("β Persei", 90.0)
>>> Zvezda.ly_pc(algol.udalj)
27.63
>>>
```

КОНЗОЛА

Класа – наслеђивање

- подржано и једноструко и вишеструко наслеђивање
- навођење родитељских класа приликом дефинисања класе
  - ако није наведена ниједна родитељска класа подразумевана родитељска класа је уграђена класа **`object`**
- провера да ли је објекат инстанца дате класе
  - помоћу функције **`isinstance()`**
    - узимају се у обзир и родитељске класе

Класа – наслеђивање

```
>>> class NebeskoTelo():
    def __init__(self, naziv):
        self.naziv = naziv

>>> class Zvezda(NebeskoTelo):
    def __init__(self, naziv, sazvežđe):
        self.naziv = naziv
        self.sazvežđe = sazvežđe

>>> kohab = Zvezda("Kohab", "Mali medved")
>>> type(kohab)
<class '__main__.Zvezda'>
>>> isinstance(kohab, Zvezda)
True
>>> isinstance(kohab, NebeskoTelo)
True
>>>
```

КОНЗОЛА

Класа – наслеђивање

- подржано директно коришћење метода из родитељских класа
  - помоћу уграђене функције **`super()`**
- подржано додавање метода инстанце које нису постојале у родитељским класама
- подржано редефинисање метода инстанце

Класа – наслеђивање

```
>>> class NebeskoTelo():
    def __init__(self, naziv):
        self.naziv = naziv
    def formiraj_opis(self):
        return self.naziv

>>> class Zvezda(NebeskoTelo):
    def __init__(self, naziv, sazvežđe):
        super().__init__(naziv)
        self.sazvežđe = sazvežđe
    def očitaj_vrstu(self):
        return "zvezda"
    def formiraj_opis(self):
        return "{} ({})".format(self.naziv, self.sazvežđe)

>>> kohab = Zvezda("Kohab", "Mali medved")
>>> kohab.očitaj_vrstu()
'zvezda'
>>> kohab.formiraj_opis()
'Kohab (Mali medved)'
>>>
```

КОНЗОЛА

Полиморфизам

- могуће ослањање на механизам наслеђивања
  - коришћење наслеђених или редефинисаних метода
  - коришћење редефинисаних апстрактних метода
    - подршка за апстрактне родитељске класе и апстрактне методе кроз модул **abc**
- могуће ослањање на механизам *duck typing*
  - приликом коришћења објекта битно да ли објекат подржава тражену радњу а не ког је типа објекат
    - ако објекат подржава тражену радњу, радња ће бити изведена

Полиморфизам

```
>>> import abc
>>> class NebeskoTelo(abc.ABC):
    @abc.abstractmethod
    def očitaj_naziv():
        pass

>>> class Zvezda(NebeskoTelo):
    def __init__(self, naziv, sazvežđe):
        self.naziv = naziv
        self.sazvežđe = sazvežđe
    def očitaj_naziv(self):
        return "{} ({})".format(self.naziv, self.sazvežđe)

>>> denebola = Zvezda("Denebola", "Lav")
>>> isinstance(denebola, NebeskoTelo)
True
>>> denebola.očitaj_naziv()
'Denebola (Lav)'
>>>
```

КОНЗОЛА

Полиморфизам

```
>>> class Zvezda():
    def __init__(self, naziv, sazvežđe):
        self.naziv = naziv
        self.sazvežđe = sazvežđe
    def očitaj_vrstu(self):
        return "zvezda"

>>> class Planeta():
    def __init__(self, naziv):
        self.naziv = naziv
    def očitaj_vrstu(self):
        return "planeta"

>>> def očitaj_vrstu(e):
    return e.očitaj_vrstu()

>>> denebola = Zvezda("Denebola", "Lav")
>>> jupiter = Planeta("Jupiter")
>>> očitaj_vrstu(denebola), očitaj_vrstu(jupiter)
('zvezda', 'planeta')
>>>
```

КОНЗОЛА

Редефинисање уграђених метода

- постоје бројне посебне методе уграђене у класу **`object`**
  - називи почињу и завршавају `__`
  - неке подразумевано користе у разним уграђеним модулима
  - неке подржавају коришћење преко оператора
- могуће редефинисање уграђене методе код корисничких класа
  - ако метода подржава позивање преко оператора онда редефинисање методе представља преклапање оператора

Редефинисање уграђених метода и преклапање оператора

неке уграђене методе

`__new__()`
успостављање нове инстанце

`__init__()`
иницијализација успостављене нове инстанце

`__del__()`
чишћење инстанце приликом њеног уништења

`__str__()`
формирање описне знаковне представе инстанце

`__len__()`
одређивање дужине инстанце или броја садржаних појава

Редефинисање уграђених метода и преклапање оператора

- неке уграђене методе с подршком за операторе
  - аритметички оператори

```
+
   __add__()
-
   __sub__()
*
   __mul__()
@
   __matmul__()
/
   __truediv__()
//
   __floordiv__()
%
   __mod__()
…
```

Редефинисање уграђених метода и преклапање оператора

- неке уграђене методе с подршком за операторе
  - релациони оператори

```
==
  __eq__()
!=
  __ne__()
>=
  __ge__()
>
  __gt__()
<=
  __le__()
<
  __lt__()
```

Редефинисање уграђених метода и преклапање оператора

- неке уграђене методе с подршком за операторе
  - оператор за индексирање и сечење (енгл. *slicing*)
    - `[]`
      - **`__getitem__()`**
        очитавање уз оператор `[]`
      - **`__setitem__()`**
        додела уз оператор `[]`
      - **`__delitem__()`**
        уклањање уз оператор `[]`

Садржај

Функционално програмирање у језику *Python*

- постоји подршка за разне концепте који примењују у функционалном програмирању
- функције су објекти
  - функције су непроменљиве
  - неке подржане радње над функцијама
    - дефинисање функције
    - позивање функције
    - прослеђивање функције као аргумената или повратне вредности
    - угњеждавање функције
    - генерисање функције (затворење, енгл. *closure*)
    - употреба декоратора
    - употреба ламбда функција

Функција – дефинисање и позивање

```
>>> def transformisanje(x, y):
    return x ** 2 + 2 * x * y + y ** 2

>>> transformisanje(1, 2)
9
>>> transformisanje
<function transformisanje at 0x0000000002DD41F0>
>>> type(transformisanje)
<class 'function'>
>>>
```

КОНЗОЛА

Функција – дефинисање и позивање

- могућност обједињеног прихватања произвољних аргумената
  - параметар **`*args`**
    - представља торку којом су обухваћени сви аргументи задати у редоследу који нису прихваћени на неки други начин
  - параметар **`**kwargs`**
    - представља речник којим су обухваћени сви аргументи задати преко назива који нису прихваћени на неки други начин
      - кључ одговара називу параметра
      - садржај одговара аргументу прослеђеном преко тог назива

Функција – дефинисање и позивање

```
>>> def transformisanje(*args, **kwargs):
    return sum(args), sum(kwargs.values())

>>> transformisanje(1, 3, 5, x = 2, y = 4)
(9, 6)
>>>
```

КОНЗОЛА

Функција – прослеђивање

```
>>> def transformisanje(x, fun):
    return fun(x)

>>> a = "True"
>>> b = transformisanje(a, bool)
>>> b
True
>>>
```

КОНЗОЛА

Функција – прослеђивање

```
>>> def biranje(x):
    if x == "i":
        return min
    elif x == "x":
        return max
    else:
        return sum

>>> f = biranje("i")
>>> f(3, 2, 1)
1
>>> biranje("s")(range(1,11))
55
>>>
```

КОНЗОЛА

Функција – угњеждавање

```
>>> def ugnježđavanje(a):
    def nf(b):
        return b / 2
    return nf(a)

>>> ugnježđavanje(3)
1.5
>>>
```

КОНЗОЛА

Функција – генерисање

затворење (енгл. *closure*)

функција уз коју везују спољни објекти који су коришћени

```
>>> import math
>>> def generisanje(f):
    def nf(x):
        return f(x) / 2
    return nf

>>> genf1 = generisanje(math.exp)
>>> genf1(2)
3.694528049465325
>>> genf2 = generisanje(math.log10)
>>> genf2(2)
0.15051499783199006
>>>
```

КОНЗОЛА

Декоратор

- функција која преузима функцију и враћа функцију
  - обично се користи за прилагођење задате функције
    - могуће је над једном функцијом редом применити више декоратора
  - постоји додатна синтакса за примену декоратора над задатом функцијом

```
@dekorator
funkcija

@dekorator2
@dekorator1
funkcije
```

Декоратор

```
>>> def generisanje(f):
    def nf(*args, **kwargs):
        return round(f(*args, **kwargs), 2)
    return nf

>>> def računanje(x, y):
    return x ** 2 / y ** 2

>>> zračunanje = generisanje(računanje)
>>> računanje(2, 3)
0.4444444444444444
>>> zračunanje(2, 3)
0.44
>>>
```

КОНЗОЛА

Декоратор

```
>>> def generisanje(f):
    def nf(*args, **kwargs):
        return round(f(*args, **kwargs), 2)
    return nf

>>> @generisanje
def računanje(x, y):
    return x ** 2 / y ** 2

>>> računanje(2, 3)
0.44
>>>
```

КОНЗОЛА

Ламбда функција (ламбда израз)

- анонимна функција
  - дефинисање помоћу резервисане речи **`lambda`**
  - подржава листу параметара
  - тело је израз

Ламбда функција

```
>>> lambda a : a ** 2 + a + 1
<function <lambda> at 0x0000000002DD41F0>
>>> (lambda a : a ** 2 + a + 1)(4)
21
>>> la = lambda a : a ** 2 + a + 1
>>> type(la)
<class 'function'>
>>> la
<function <lambda> at 0x0000000002DD41F0>
>>> la(4)
21
>>>
```

КОНЗОЛА

Ламбда функција

```
>>> def računanje(x, f):
        return x, f(x)

>>> računanje(4, lambda a : a ** 2 + a + 1)
(4, 21)
>>>
```

КОНЗОЛА

Садржај

Извори и литература

Основни извори и литература

- Lubanovic B. Uvod u Python: Moderno računarstvo u jednostavnim paketima. O'Reilly (Sebastopol, CA, USA), CET (Beograd), Računarski fakultet (Beograd); 2015.
- Kovačević MA. Osnove programiranja u Pajtonu. Akademska misao (Beograd); 2017.
- Pilgrim M. Dive Into Python 3. Apress (New York, NY, USA); 2009. Internet: https://diveintopython3.net/
- Welcome to Python.org. Internet: https://www.python.org/
- PEP 8 -- Style Guide for Python Code. Internet: https://www.python.org/dev/peps/pep-0008/
- Python 3.9.2 documentation. Internet: https://docs.python.org/3.9/

Извори и литература

Додатни извори и литература

- Downey AB. Think Python: How to Think Like a Computer Scientist. 2nd edition. Green Tea Press (Needham, MA, USA); 2015. Internet: https://greenteapress.com/wp/think-python-2e/

Основне академске студије
Информациони инжењеринг

Основи рачунарске интелигенције

Парадигме програмирања у језику и окружењу Python

(материјали за предавања)