Základy jazyka Python

pokročilé programovanie v Pythone

prednáška 12

Katedra kybernetiky a umelej inteligencie
Technická univerzita v Košiciach
Ing. Ján Magyar

Metódy a triedy ako premenné

- v Pythone každá metóda a trieda je zároveň aj objektom, vieme s nimi narábať ako s hodnotami
- metódy a triedy vieme uložiť do premenných a následne ich používať cez danú premennú

```
def my_func():
    return 5
```

```
result = my_func()
result ← 5
```

```
result = my_func
result ← <function my_func>
```

Metódy ako premenné

- ak v čase písania kódu nevieme, ktorá funkcia sa má zavolať
- zvyčajne to používame, ak chceme používateľom umožniť prepínať medzi viacerými funkcionalitami (napr.: rôzne testy pri simuláciach)
- vieme to urobiť v rámci jednej funkcie, pomocou vetvenia
- ak funkcie uložíme do premennej, mali by mať približne rovnaké rozhranie API
  - rovnaký počet a typ parametrov
  - rovnaká návratová hodnota

Triedy ako premenné

- ak v čase písania kódu nevieme, z ktorej triedy chceme vytvoriť inštanciu
- napr.: máme niekoľko výpočtových modelov, ktoré definujú špecifikáciu toho istého hlavného konceptu, a chceme používateľom umožniť, aby si vybrali konkrétnu špecifikáciu počas behu programu (viď. podtriedy `Drunk` v simulácii biased random walk)
- ekvivalent je využiť návrhový vzor abstract factory, resp. builder
- API rozhranie tried musí byť rovnaké
  - definovať rovnaké metódy s rovnakými parametrami a návratovými hodnotami
  - definovať rovnaké atribúty
  - triedy môžu definovať funkcionalitu navyše, ale zdieľaná časť funkcionality by mala byť rovnaká

Menný priestor

- menný priestor mapuje mená na objekty a hodnoty
- v Pythone je menný priestor implementovaný ako dictionary
- menný priestor môže obsahovať
  - defaultné mená (kľúčové slová, funkcie, výnimky)
    - vytvorí sa pri spustení interpretera, zanikne po skončení programu
  - globálne mená v moduloch
    - vytvorí sa pri importe, zanikne po skončení programu
  - lokálne mená vo funkciách
    - vytvorí sa pri spustení funkcie, zanikne po skončení vykonávania

Oblasť - scope

- oblasť definuje časť programu, z ktorej je daný menný priestor priamo dostupný
- oblasť slúži na vyhľadávanie mien počas behu programu:
  - vnútorná oblasť
  - zapuzdrujúca oblasť
  - globálne názvy v module
  - vonkajšia oblasť
- nové atribúty sa vytvoria defaultne v rámci lokálnej oblasti

Global a nonlocal

- Python umožňuje určiť oblasť pre vytvorené atribúty
- `global`
  - atribút bude dostupný z oblasti modulu (nie z interpretera!)
- `nonlocal`
  - upravuje atribúty, ktoré sú dostupné z inej ako vnútornej oblasti
  - bez `nonlocal` sa vytvorí nový atribút s rovnakým názvom
  - bez `nonlocal` budú atribúty read-only

```
def scope_test():
    def do_local():
        spam = "local spam"

    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"

    def do_global():
        global spam
        spam = "global spam"

    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)

scope_test()
print("In global scope:", spam)
```

Vnorené funkcie

- Python umožňuje definíciu funkcie v tele inej funkcie
- vnútorná funkcia bude dostupná iba z vonkajšej funkcie
- ideálne pre riešenie podúlohy, ak podúloha sa má riešiť iba v rámci jednej funkcie

Closure

- špeciálny prípad vnorených funkcií
  - musíme mať vnorenú funkciu (funkciu definovanú v tele inej funkcie)
  - vnorená funkcia musí pracovať s hodnotou definovanou vo vonkajšej funkcii
  - vonkajšia funkcia musí vrátiť vnútornú funkciu
- telo, resp. atribúty vnútornej funkcie existujú aj po vymazaní vonkajšej funkcie
- umožňuje nám predísť použitiu globálnych premenných, podporuje zapuzdrenie, a generuje funkcie (metaprogramovanie - dekorátory)

```
def make_power_of(n):
    def power_of(x):
        return x ** n

    return power_of

power2 = make_power_of(2)
power3 = make_power_of(3)

print(power2(2))
print(power3(2))

print(make_power_of(2)(3))

del make_power_of

print(power2(2))
print(power3(2))

print(make_power_of(2)) -> NameError
```

Numpy

- implementácia polí v Pythone (v skutočnosti sú to matice)
- podobné, ako zoznamy, iba nemeniteľné (hodnoty však vieme aktualizovať)
- vytvorenie polí:

```
import numpy as np
my_array = np.array([2, 3, 4])
```

- konvertovanie vstupu na pole:

```
numpy.asarray(a, dtype=None, order=None)
```

Typ polí

- každé pole má typ `ndarray`, jednotlivé prvky musia byť rovnakého typu

  ```
  my_array.dtype
  ```

- typy prvkov nie sú pythonovské primitívne typy, sú to vlastné implementácie knižnice `numpy`

Tvar polí

- každé pole má jednu alebo viac dimenzií
- tvar poľa je definovaná rozmermi

```
my_array.shape
```

- vracia n-ticu s rozmermi, napr.: `(2, 3)`, kde prvá hodnota je vonkajší rozmer, a postupuje smerom dnu
- ak je pole dvojdimenzionálne, prvý rozmer je počet riadkov, druhý je počet stĺpcov

Úprava rozmerov poľa

- `numpy.reshape(array, newshape, order='C')`
  - upraví tvar poľa na požadované rozmery
  - počet prvkov v poli a počet prvkov v novom tvare musí byť rovnaký
  - order určí poradie pridávania prvkov (`C` alebo `F`)
  - vracia nové pole
- `numpy.flatten(order='C')`
  - vracia nové pole - vektorová reprezentácia (jeden riadok, resp. stĺpec)

zeros() a ones()

- metódy slúžia na inicializáciu matice s hodnotami 0 alebo 1
- `numpy.zeros(shape, dtype=float, order='C')`
- `numpy.ones(shape, dtype=float, order='C')`

Pridávanie prvkov

- implementuje konkatenáciu polí
- `numpy.hstack(tup)`
  `numpy.hstack((array1, array2))`
- `numpy.vstack(tup)`
  `numpy.vstack((array1, array2))`
- pre vytvorenie polí dynamicky je lepšie použiť zoznamy, a následne vygenerovať pole

Indexovanie a slicing v numpy

- zásady sú rovnaké ako v Pythone, numpy ale umožňuje skrátený zápis

```
np.array([(1,2,3), (4,5,6)])
print('First row:', e[0])
print('Second column:', e[:,1])
print('Second row, first two values:', e[1, :2])
```

Aritmetické operácie nad maticami

- primitívne operácie fungujú element-wise

```
test = np.array([1, 2, 3])
test += 1

test [2, 3, 4]
```

Štatistické metódy

- `np.min(array)`
- `np.max(array)`
- `np.mean(array)`
- `np.median(array)`
- `np.std(array)`

Maticové operácie

- `np.transpose(array, axes=None)`
- `np.dot(array1, array2)`
- `np.matmul(array1, array2)`
  - ak polia sú dvojrozmerné - štandardné násobenie
  - ak polia majú jeden rozmer - vygeneruje sa z nich dvojrozmerné pole
  - ak polia majú viac rozmerov, ako dva - považujú sa za zásobník dvojrozmerných matíc

Generovanie rozsahov

- `numpy.arange(start, stop,step)`
  - funguje rovnako, ako `range()`
- `numpy.linspace(start, stop, num=50, endpoint=True)`
  - vygeneruje `num` hodnôt v rovnakej vzdialenosti od seba v intervale `[start, stop]`
  - `endpoint` určí, či interval má obsahovať hodnotu stop

otázky?