České vysoké učení technické v Praze FIT

Programování v Pythonu

Jiří Znamenáček

Příprava studijního programu Informatika je podporována projektem financovaným z Evropského sociálního fondu a rozpočtu hlavního města Prahy.

Praha & EU: Investujeme do vaší budoucnosti

Python – N-tice

Úvod

N-tice (aneb taply, angl. *tuples*) jsou v Python'u snad ještě důležitější typ než seznamy. Jen asi často nejsou na první pohled tak vidět. Stejně jako seznamy je to typ sekvenční, ale narozdíl od seznamů neměnitelný (*immutable*), což je jeho zcela základní a vůbec nejvýznamější vlastnost.

> → Popravdě to, že n-tice jsou sekvenční typ, je spíše na škodu jejich logičtějšímu záměru – totiž jako „lehkotonážních“ datových struktur (ke kterým se v Python'u 3.x více blíží díky zavedení *named tuples*).

Z daných objektů vytvoříme n-tici nejsnadněji jejich uzavřením do kulatých závorek:

```
>>> xs = ( 'Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 6
>>> xs
('Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 666')
```

Kromě toho je možné za pomoci funkce `tuple()` převádět jiné typy na n-tice, například řetězce:

```
>>> xs = "Ahoj, světe!"
>>> xs
'Ahoj, světe!'
>>> tuple(xs)
('A', 'h', 'o', 'j', ',', ' ', 's', 'v', 'ě', 't', 'e', '!')
```

N-tice jako sekvence

Jelikož n-tice patří mezi sekvence, máme k dispozici celou armádu sekvenčních operací:

```
>>> xs = ('Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 66

# délka sekvence
>>> len(xs)
5

# konkrétní prvek
>>> xs[3]
['hokus', 'pokus']
>>> xs[-3]
(12, 'tři')

# různé výřezy
>>> xs[1:3]
(123, (12, 'tři'))
>>> xs[1::2]
(123, ['hokus', 'pokus'])
>>> xs[3:]
(['hokus', 'pokus'], 'lokus 666')
>>> xs[-3:]
((12, 'tři'), ['hokus', 'pokus'], 'lokus 666')

# dotaz na výskyt prvku
>>> 123 in xs
True
>>> (12, 'tři') in xs
True
>>> 'fokus' in xs
False

# dvě spojené kopie
>>> xs * 2
('Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 666', 'Ahoj
```

→ Komentáře zde pochopitelně nejsou součástí výstupu interaktivního interpretru, slouží pouze k popisu zde zapsaného kódu.

Po sekvencích se navíc přirozeně prochází smyčkou:

```
>>> for x in xs:
...     print(x)
...
Ahoj!
123
(12, 'tři')
['hokus', 'pokus']
lokus 666
```

Případně pokud je důležitá i pozice výskytu prvku:

```
>>> for (i, x) in enumerate(xs):
...     print(i, x)
...
0 Ahoj!
1 123
2 (12, 'tři')
3 ['hokus', 'pokus']
4 lokus 666
```

N-tice jako neměnitelné sekvence

Podobně jako řetězce patří n-tice mezi sekvence **neměnitelné** (*immutable*), což znamená, že jejich obsah po vytvoření již není možné měnit:

```
>>> xs = ( 'Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 6
>>> xs[3]
['hokus', 'pokus']
>>> xs[3] = 'X'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Spojením dvou n-tic vzniká n-tice zcela nová, proto následující kód je v pořádku:

```
>>> xs = xs + ("svět", 666)
>>> xs
('Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 666', 'svět
```

Dotazování na prvky

Kromě již dříve uvedeného a všem sekvencím společného operátoru `in`:

```
>>> xs = ('Ahoj!', 123, (12, 'tři'), ['hokus', 'pokus'], 'lokus 66

>>> 123 in xs
True
>>> (12, 'tři') in xs
True
>>> 'fokus' in xs
False
```

...mají n-tice k dispozici i metodu `index(PRVEK)`, která slouží k navrácení pozice prvního výskytu prvku s uvedenou hodnotou:

```
>>> xs = ('jedna', 'dvě', 'tři', 'jedna', 'pět', 'šest', )

>>> xs.index('jedna')
0
>>> xs.index('pět')
4
```

→ Není-li takový prvek v seznamu k dispozici, obdržíme výjimku *ValueError*.

Kromě toho můžeme i počítat výskyty zadaného prvku pomocí metody `count(PRVEK)`:

```
>>> xs = ('jedna', 'dvě', 'tři', 'jedna', 'pět', 'šest', )

>>> xs.count('jedna')
2
>>> xs.count('dvě')
1
```

Přiřazování více hodnot najednou

Snad nejpřekvapivější, ale přitom nejčastější, je použití n-tic pro přiřazování více hodnot najednou. Než se pokoušet o vysvětlování, radši rovnou pár příkladů (podle DiP3):

I. Nejběžnější použití n-tic pro *přiřazení* více hodnot najednou:

```
>>> v = ('a', 2, True)
>>> v
('a', 2, True)
>>> type(v)
<class 'tuple'>

>>> (x, y, z) = v
>>> x
'a'
>>> y
2
>>> z
True
```

→ V podstatě jsme do n-tice o třech proměnných `(x, y, z)` přiřadili n-tici `v`, která má právě tři prvky. Ve výsledku do každé ze tří proměnných byl přiřazen odpovídající prvek z n-tice `v`. A jelikož konstruktorem n-tice v literální notaci je čárka, můžete napsat též `x, y, z = v` a výsledek bude úplně stejný.

II. Podobně můžeme více hodnot najednou *předat*, např. jako návratovou hodnotu z funkce:

Program *tuples1.py* **:**

```
def fce(x):
    return x, x**2, x**3

# A) vrácení n-tice
y = fce(2)
print(y)
print( type(y) )

print()    # prázdná řádka pro p

# B) vrácení a rozbalení n-tice
prvni, druha, treti = fce(3)
print(prvni, druha, treti)
```

Výstup:

```
(2, 4, 8)
<class 'tuple'>

3 9 27
```

→ Zde v prvním případě předáváme vracenou n-tici o třech prvcích do jedné proměnné *y* - stane se z ní tudíž také n-tice o třech prvcích. Ve druhém případě provedeme rovnou „rozbalení“ n-tice na jednotlivé prvky.

III. A dokonce můžeme takhle přiřazovat i objekty, které nejsou n-ticemi, ale dá se po nich iterovat, např. seznamy:

```
>>> x, y, z = [1, 2, 3]

>>> x
1
>>> y
2
>>> z
3
```

...nebo třeba i rozsahy (*range*):

```
>>> (MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDA

>>> MONDAY
0
>>> FRIDAY
4
```

→ Inspirace z modulu `calendar`.

IV. Do stejné „vlny“ patří i typicky pythonovské prohození hodnot dvou proměnných:

```
a, b = b, a
```

...nebo přiřazení blíže neurčeného počtu dat do menšího počtu proměnných za pomoci operátoru `*`:

```
>>> a, *b, c = 'první', 1, 2, 'ahoj', 'poslední'

>>> a
'první'
>>> b
[1, 2, 'ahoj']
>>> c
'poslední'
```

→ Zde máme vpravo n-tici o pěti prvcích, ale nalevo provádíme „rozbalení“ pouze do tří proměnných. Při tom je jasně určeno přiřazení prvního, resp. posledního, prvku do proměnných *a*, resp. *c*, a vše ostatní pod sebe „schlamstne“ díky operátoru `*` proměnná *b* (a to v podobě *seznamu*).

Význam n-tic

Jelikož jsou n-tice neměnitelné (*immutable*), tak se z hlediska sekvenčních typů chovají velmi podivně:

- žádné nové prvky do nich přidávat nemůžeme
- žádné prvky z nich nemůžeme mazat
- nemůžeme je řadit

Jinými slovy n-tice slouží k „zapouzdření“ několika (heterogenních) hodnot do společné „zmražené“ datové struktury.

Ostatně když se podíváme, co „všechno“ n-tice podporují:

```
>>> dir(xs)
['__add__', '__class__', '__contains__', '__delattr__', '__doc__',
```

→ Pouze dvě metody – `count()` a `index()` !

...tak nebýt předchozího slajdu o přiřazování více hodnot najednou (tedy právě onoho zmíněného „zapouzdřování“), kvůli kteréžto vlastnosti by jiné jazyky snad i vraždily, mohli bychom propadnout depresi, k čemu jsou vlastně vůbec dobré.

Shrneme-li to (podle DiP3):

- n-tice jsou rychlejší než seznamy (takže postačí-li vám, nebo dokonce vyžadujete konstantní sadu hodnot, použijte spíše n-tici než seznam)
- narozdíl od seznamu **n-tici nemůžeme měnit** (možná toto neustálé zdůrazňování už zní hloupě, ale jsou situace, kdy budete rádi, že na data nemůžete ani vy, ani nikdo jiný nijak „šáhnout“)
- právě díky své neměnnosti (jsou tak totiž hešovatelné) můžete n-tice složené z neměnných typů použít jako klíče do slovníků (a jestli se vám z toho nezatočila hlava, tak už asi z ničeho)