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 – Řetězce

Úvod

Řetězce nejsnadněji zavedeme jako *string literal* pomocí uvozovek či apostrofů. Platí přitom pravidlo, že pro krátké řetězce na jednu řádku stačí uvozovky jednoduché, pro dlouhé řetězce na řádek více uvozovky trojté:

Program *strings1.py* **:**

```
s1_short = 'Ahoj, světe!'
s1_long  = '''Příliš žluťoučký
  úpěl ďábelské ódy.'''

s2_short = "Ahoj, světe!"
s2_long  = """Příliš žluťoučký
  úpěl ďábelské ódy."""

print( s1_short )
print( s1_long )
print()
print( s2_short )
print( s1_long )
```

Výstup:

```
Ahoj, světe!
Příliš žluťoučký kůň
  úpěl ďábelské ódy.

Ahoj, světe!
Příliš žluťoučký kůň
  úpěl ďábelské ódy.
```

→ Všimněte přenosu zalomení řádku a mezer na začátku pro dvouřádkové ukázky!

Kolizní uvozovky uvnitř řetězců je samozřejmě třeba iskejpovat (standardně pomocí \, který samotný pak tudíž musíte zapsat jako \\), jinak můžete použít (téměř) klasické céčkovské řídící sekvence (*escape sequences*):

Program *strings2.py* **:**

```
print( "Ahoj, 'Pavle', jak se m
print( "Ahoj, \"Pavle\", jak se
print()
print( "Ahoj, Pavle,\njak se má
print( "Ahoj, Pavle,\n\tjak se
```

Výstup:

```
Ahoj, 'Pavle', jak se máš.
Ahoj, "Pavle", jak se máš.

Ahoj, Pavle,
jak se máš.
Ahoj, Pavle,
        jak se máš.
```

Pokud **ne**chcete, aby se řídící sekvence uvnitř řetězců vyhodnocovaly,

musíte řetězec označit jako *raw*:

Program *strings3.py* **:**

```
print( r"Ahoj, Pavle,\njak se m
print( R"Ahoj, Pavle,\n\tjak se
```

Výstup:

```
Ahoj, Pavle,\njak se máš.
Ahoj, Pavle,\n\tjak se máš.
```

→ Prefix `r` i `R` fungují stejně.

Řetězce jako sekvence

Řetězce v Python'u 3.X jsou automaticky unicodové => každý jeden viditelný znak je reprezentován proměnným počtem bajtů. Ale z hlediska běžné práce s řetězci jako *sekvenčními typy* (kterými řetězce jsou) nás to nemusí zajímat, protože se chovají právě tak, jak bychom čekali – nejmenší podjednotka řetězce je právě jeden znak.

Jelikož řetězce patří mezi sekvence, máme k dispozici celou armádu sekvenčních operací:

```
>>> xs = "Ahoj, světe!"

# délka sekvence
>>> len(xs)
12

# konkrétní prvek
>>> xs[3]
'j'
>>> xs[-3]
't'

# různé výřezy
>>> xs[3:9]
'j, svě'
>>> xs[3:9:2]
'j v'
>>> xs[3:]
'j, světe!'
>>> xs[-3:]
'te!'

# dotaz na výskyt prvku
>>> 'a' in xs
False
>>> 'A' in xs
True

# dvě spojené kopie
>>> xs * 2
'Ahoj, světe!Ahoj, světe!'
```

→ 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)
...
A
h
o
j
,

s
v
ě
t
e
!
```

Případně pokud je důležitá i pozice výskytu znaku:

```
>>> for (i, x) in enumerate(xs):
...     print(i, x)
...
0 A
1 h
2 o
3 j
4 ,
5
6 s
7 v
8 ě
9 t
10 e
11 !
```

Řetězce jako neměnitelné sekvence

Řetězce patří mezi sekvence **neměnitelné** (*immutable*), což znamená, že jejich obsah po vytvoření již není možné měnit:

```
>>> xs = "řetězec"
>>> xs[3]
'ě'
>>> xs[3] = 'X'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Spojením dvou řetězců vzniká řetězec zcela nový, proto následující kód je v pořádku:

```
>>> xs = xs + " ukázkový"
>>> xs
'řetězec ukázkový'
```

Formátování výstupu

Pro formátované vkládání hodnot z objektů dovnitř řetězce slouží metoda `format()`. Je ji možno použít jak pozičně, tak za pomoci pojmenovaných argumentů (což je ukecanější, ale může to být v některých situacích přehlednější):

```
>>> "Prvek {0} by měl mít průměr {1:.3} cm.".format( 'X', 1/3 )
'Prvek X by měl mít průměr 0.333 cm.'

>>> "Prvek {} by měl mít průměr {:.3} cm.".format( 'X', 1/3 )
'Prvek X by měl mít průměr 0.333 cm.'

>>> "Prvek {jmeno} by měl mít průměr {prumer:.3} cm.".format( jmen
'Prvek X by měl mít průměr 0.333 cm.'
```

- → Možnost vynechat označení pozice, pokud použijeme argumenty v zadaném pořadí, s sebou přinesl Python 3.1.
- → Formátování pomocí `format()` je mnohem mocnější – jako parametr do `{}` můžete poslat prakticky cokoliv (seznam, slovník, instanci třídy...) a celkem rozumně s tím pak pracovat.
- → Starší způsob formátování, dosud podporovaný, ale určen k (pravděpodobnému) vyřazení, vypadá pro srovnání takto: `'%(language)s has %(#)03d quote types.' % {'language': "Python", "#": 2}` dává výstup `'Python has 002 quote types.'`

Při zarovnávání textu pro výstup nejen na konzoli se vám asi budou hodit následující tři řetězcové metody:

```
>>> xs = "AHOJ"

>>> xs.center(40)
'                   AHOJ                 '
>>> xs.center(40, '*')
'******************AHOJ******************'

>>> xs.ljust(40)
'AHOJ                                    '
>>> xs.ljust(40, '-')
'AHOJ------------------------------------'

>>> xs.rjust(40)
'                                    AHOJ'
>>> xs.rjust(40, '+')
'++++++++++++++++++++++++++++++++++++AHOJ'
```

Analýza obsahu I

Pro obecné posouzení vybraných vlastností řetězce dobře poslouží následující metody (pouze výběr):

```
>>> '123'.isalnum()
True
>>> '123abc'.isalnum()
True
>>> '123abc+-*/'.isalnum()
False

>>> 'abc'.isalpha()
True
>>> 'abc123'.isalpha()
False

>>> '123'.isdigit()
True
>>> '123 '.isdigit()
False

>>> '    '.isspace()
True
>>> ' baf       '.isspace()
False
```

```
>>> 'abc'.islower()
True
>>> 'aBc'.islower()
False

>>> 'ABC'.isupper()
True
>>> 'AbC'.isupper()
False

>>> 'Svět'.istitle()
True
>>> 'Svět Je'.istitle()
True
>>> 'Svět je'.istitle()
False
```

```
>>> 'ahoj'.isidentifier()
True
>>> '123ahoj'.isidentifier()
False
```

Analýza obsahu II

Kromě metod na zjišťování „globálních“ vlastností řetězce z předchozího slajdu máme k dispozici i několik dalších metod zaměřených na vyhledávání podřetězců:

```
>>> 'řetězec'.count('t')
1
>>> 'řetězec'.count('e')
2
>>> 'řetězec'.count('W')
0

>>> xs = "Nesnesu se se sestrou."
>>> xs.count('se')
3
```

```
>>> 'řetězec'.startswith('ř')
True
>>> 'řetězec'.startswith('Ř')
False

>>> 'řetězec'.endswith('c')
True
>>> 'řetězec'.endswith('u')
False

>>> xs = "Nesnesu se se sestrou."
>>> xs.startswith('Nes')
True
>>> xs.startswith('Nest')
False
>>> xs.endswith('ou.')
True
>>> xs.endswith('ou')
False
```

Konkrétně vyhledávat podřetězce umožňují následující metody:

```
>>> xs = "There were 42 monkeys in the airplane."

>>> xs.find('e')   # najdi první výskyt
2
>>> xs.rfind('e')   # najdi poslední výskyt
36

>>> xs.find('e', 10)   # hledej v xs[10:]
18
>>> xs.find('e', 10, 17)   # hledej v xs[10:17]
-1

>>> xs = "Nesnesu se se sestrou."
>>> xs.find('se')
8
>>> xs.rfind('se')
14
>>> xs.find('se', 12)
14
```

- → *Slice*-notace (tedy výběr podsekvence) funguje i pro `rfind`.
- → Varianty `index` a `rindex` fungují úplně stejně jako `find` a `rfind`, pouze místo `-1` v případě neúspěšného vyhledávání vyhodí výjimku *ValueError*.

Úprava obsahu – velikost písmen

Několik metod je zaměřených na práci s velikostí písmen:

```
>>> xs = "Ahoj, světe!"

>>> xs.swapcase()
'aHOJ, SVĚTE!'
>>> xs.upper()
'AHOJ, SVĚTE!'
>>> xs.lower()
'ahoj, světe!'

>>> xs.title()
'Ahoj, Světe!'

>>> 'ahoj, světe!'.capitalize()
'Ahoj, světe!'
```

Úprava obsahu – odstraňování znaků

Řetězec můžeme snadno zbavit na obou koncích nechtěných znaků:

```
>>> xs = '12 to by nešlo 24'

>>> xs.strip('1234')
' to by nešlo '
>>> xs.lstrip('1234')
' to by nešlo 24'
>>> xs.rstrip('1234')
'12 to by nešlo '
```

Bez udání parametru jsou výchozou hodnotou do všech tří funkcí prázdné znaky (*whitespace*):

```
>>> xs = '    to by nešlo    '

>>> xs.strip()
'to by nešlo'
>>> xs.lstrip()
'to by nešlo    '
>>> xs.rstrip()
'    to by nešlo'
```

Úprava obsahu – záměna znaků

Výskyty nějakého podřetězce můžeme snadno nahradit podřetězcem jiným:

```
>>> xs = "dog,cat,pig,hippo,chicken"

>>> xs.replace(',', ' - ')
'dog - cat - pig - hippo - chicken'
>>> xs.replace(',', ' - ', 2)
'dog - cat - pig,hippo,chicken'

>>> xs = "333 stříbrných stříkaček stříkalo přes 333 stříbrných st
>>> xs.replace('333', 'třistatřiatřicet')
'třistatřiatřicet stříbrných stříkaček stříkalo přes třistatřiatři
```

splitting & joining I

Řetězec můžeme podle zadaného podřetězce rozdělit na více částí.
Nejobecnější metodou je metoda `split()`:

```
>>> xs = "Ahoj, světe! Jak se máš?"

>>> xs.split()
['Ahoj,', 'světe!', 'Jak', 'se', 'máš?']
>>> xs.split(' ')
['Ahoj,', 'světe!', 'Jak', 'se', 'máš?']
>>> xs.split(' ', 2)
['Ahoj,', 'světe!', 'Jak se máš?']

>>> xs.split(',')
['Ahoj', ' světe! Jak se máš?']

>>> xs.split(' s')
['Ahoj,', 'věte! Jak', 'e máš?']
```

→ Bez udání parametrů rozdělí `split()` zadaný řetězec podle

prázdných znaků (*whitespaces*).

→ Druhý nepovinný parametr určuje maximální počet rozdělení, ke kterým dojde.

→ `split()` vrací *seznam*.

Pro srovnání ještě metoda `rsplit()` – až na neúplné rozřezání řetězce zprava se chová stejně jako `split()`:

```
>>> xs.split(' ', 2)
['Ahoj,', 'světe!', 'Jak se máš?']
>>> xs.rsplit(' ', 2)
['Ahoj, světe! Jak', 'se', 'máš?']
```

Trochu podobná metoda `partition()`, resp. `rpartition()`, rozdělí zadaný řetězec podle podle prvního, resp. posledního, výskytu udaného separátoru:

```
>>> xs = "Ahoj, světe! Jak se máš?"

>>> xs.partition('e')
('Ahoj, svět', 'e', '! Jak se máš?')
>>> xs.rpartition('e')
('Ahoj, světe! Jak s', 'e', ' máš?')

>>> xs.partition('Jak')
('Ahoj, světe! ', 'Jak', ' se máš?')
```

→ Narozdíl od `split()` tedy `partition` nezahodí separátor a navíc vrací *tupl*.

K dispozici máme ještě metodu specializovanou na rozdělování řetězců obsahujících odřádkování:

Program *splitlines.py* **:**

```
xs = """Tady je velmi dlouhý te
který zabírá více řádek. Aby ta
nezabíral, když je v něm napsán
nesmyslů, že.
"""

# standardně za použití separát
x1 = xs.split("\n")
print( x1 )

# za pomoci vestavěné metody
x2 = xs.splitlines()
print( x2 )
```

Výstup:

```
['Tady je velmi dlouhý text,',
['Tady je velmi dlouhý text,',
```

→ Všimněte si odlišného chování k poslednímu *\n*!

splitting & joining II

Doplňkovou metodou ke `split()` je metoda `join()`:

```
>>> xs = "Ahoj, světe! Jak se máš?"

>>> ys = xs.split()
>>> ys
['Ahoj,', 'světe!', 'Jak', 'se', 'máš?']

>>> ' '.join(ys)
'Ahoj, světe! Jak se máš?'
```

→ Syntaxe je tedy `'SEPARÁTOR'.join( ITERABLE )` (kde ITERABLE je jakýkoli objekt schopný vracet své členy po jednom, v tomto případě seznam).

Nic nám samozřejmě nebrání použít jiný separátor:

```
>>> '-|-'.join(ys)
'Ahoj,-|-světe!-|-Jak-|-se-|-máš?'
```

Typické je použití k normalizaci řetězce:

```
>>> xs = "    Hello    World,    I       am   here.  "
>>> xs
'    Hello    World,    I       am   here.  '
>>> ' '.join( xs.split() )
'Hello World, I am here.'
```

ord() & chr()

Pro operaci s řetězci **na úrovni jednotlivých znaků** slouží dvě standardní funkce `ord()` a `chr()`.

I. Funkce `ord()` převede zadaný jeden znak (tedy řetězec o délce jedna) na *codepoint*-číslo odpovídající tomuto znaku v Unicode-tabulce:

```
>>> ord('a')
97
>>> ord('ř')
345
```

II. Komplementární funkce `chr()` převádí naopak zadané celé číslo představující unicodový *codepoint* na odpovídající znak (tedy řetězec délky jedna):

```
>>> chr(97)
'a'
```

→ Pro více o Unicode'u viz Kódování, Unicode.