Práce s řetězci

Jiří Zacpal

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

KMI/ZPP1 Základy programování v Pythonu 1

Práce s řetězci

Řetězce

- Řetězce jsou tvořeny sekvencí znaků , které jsou v počítači reprezentovány pomocí čísel.
- Python podporuje řetězce kódované podle ASCII tabulky nebo pomocí Unicode.
- Zápis řetězce:

```
s = ’ahoj’
s = "ahoj"
```

- V řetězci lze uvést i escape sekvence:

```
\n, \t, \a, \\, \", \', \0
```

- Jednoprvkový řetězec je znak.
- Řetězce se tvoří ze třídy `str`.

Operace s řetězci

- Řetězce můžeme spojovat pomocí operátoru +:

```
>>> "Základy " + " programování " + " v " + " Pythonu"
'Základy  programování  v  Pythonu‘
```

- Nelze použít jako operand jiný datový typ:

```
>>> "Základy " + " programování " + " v " + " Pythonu " + 1
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

- Lze však operand převést na řetězec:

```
>>> "Základy " + " programování " + " v " + " Pythonu " + str(1)
'Základy  programování  v  Pythonu 1'
```

Operace s řetězci

- Další operátor, který může mít jako operand řetězec je *:

```
>>> "Python "*3
'Python Python Python '
>>> 3*"Python "
'Python Python Python '
>>>
```

Formátovací řetězce

- Pro spojování řetězců můžeme použít i metodu

string.format(*hodnota0, hodnota1...*)

která do řetězce vloží na místo oblastí náhrad v řetězci hodnoty uvedené jako argumenty.

- Příklad:

```
>>> "{0} {1} {2} {3} {4}.".format("Základy","programování","v","Pythonu",1)
'Základy programování v Pythonu 1.‘
```

- Počet hodnot musí odpovídat počtu oblastí náhrad:

```
>>> "{0} {1} {2} {3} {4}.".format("Základy","programování","v","Pythonu")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: Replacement index 4 out of range for positional args tuple
```

Formátovací řetězce

- Číslování oblastí náhrad musí korespondovat s počtem hodnot:

```
>>> "{0} {1} {2} {3} {5}.".format("Základy","programování","v","Pythonu",1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: Replacement index 5 out of range for positional args tuple
```

- Hodnotou nemusí být jen konstanta, ale obecně jakýkoliv výraz:

```
>>> "Součet čísel {0} a {1} je {2}.".format(a,b,a+b)
'Součet čísel 5 a 3 je 8.'
```

Specifikátory formátu

- V oblasti náhrad můžeme uvést specifikátor formátu:
  - `:f` – určí formátování desetinného čísla

    ```
    >>> "Číslo:{0:.2f}".format(123.456)
    'Číslo:123.46‘
    ```
  - `:b` – vypíše číslo v binární soustavě

    ```
    >>> "Číslo:{0:b}".format(123)
    'Číslo:1111011‘
    ```
  - `:%` - vypíše číslo jako procenta

    ```
    >>> "Číslo:{0:.2%}".format(0.25)
    'Číslo:25.00%'
    ```

Délka řetězce

- Pro délku řetězce použijeme funkci:

```
len(retezec)
```

  Funkce vrátí počet znaků v řetězci.
- Příklad:

```
>>> len('Python')
6
>>> len('')
```

Řetězec jako kolekce

- Řetězce jsou imutabilní kolekce.
- Pro práci s nimi tedy můžeme použít cyklus for:

```
s="Python"
for z in s:
    print(z)
```

Operátor indexu

`r[i]`

- Operátor vrátí i-tý znak z řetězce r.
- Příklad:

```
>>> 'Python'[0]
'P'
>>> s = 'Python'
>>> s[0]
'P'
>>> s[5]
'n'
>>> s[6]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

Příklad

- S pomocí cyklu for vypíšeme všechny znaky řetězce:

```
s = 'Python'

for i in range(len(s)):
    print(s[i])
```

Příklad

- Program pro vytvoření řetězce s obráceným pořadím znaků:

```
string = 'Python'
reverse = ''

for i in range(len(string)):
    reverse = string[i] + reverse

print(reverse)
```

Porovnávání řetězců

- Operátory == a != lze použít k porovnávání řetězců.
- Přitom dva řetězce jsou stejné, pokud mají stejnou délku a znaky na odpovídajících indexech se rovnají.
- Příklad:

```
>>> string = 'Python'
>>> string == 'Python'
True
>>> 'p' != 'P'
True
```

- Porovnávání je citlivé na velikost písmen:

```
>>> 'python' == 'Python'
False
```

Příklad

- Následující program otestuje, zda je řetězec palindrom:

```
string = 'kobylamamalybok'
reverse = ''

for i in range(len(string)):
    reverse = string[i] + reverse

is_palindrom = string == reverse
print(is_palindrom)
```

- Lze program napsat i bez konstrukce řetězce reverse?

Příklad

```
string = 'kobylamamalybok'
is_palindrom = True
string_len = len(string)
i = 0
n = string_len // 2

while i < n and is_palindrom:
    if string[i] != string[string_len - 1 - i]:
        is_palindrom = False
    i += 1

print(is_palindrom)
```

Metody řetězců

Rozdělení řetězce

- Metoda

```
split(ret)
```

- vrátí seznam podřetězců, které byly v původním řetězci odděleny určitým řetězcem (výchozí je mezera).
- Příklad:

```
>>> 'Základy programování v Pythonu 1.'.split()
['Základy', 'programování', 'v', 'Pythonu', '1.']
>>> 'A-h-o-j'.split("-")
['A', 'h', 'o', 'j']
```

Spojování řetězce

- Metoda

`join(kolekce)`

- vrátí řetězec, který bude tvořen prvky kolekce oddělených řetězcem, který metodu volá.
- Příklad:

```
>>> s=["Jan","Alena","Jirka"]
>>> "-".join(s)
'Jan-Alena-Jirka'
>>> "*".join("Python")
'P*y*t*h*o*n'
```

Převod na malá a velká písmena

- Metoda

.upper()

převede všechna písmena v řetězci na velká

- Metoda

.lower()

převede všechna písmena v řetězci na malá

- Příklad:

```
>>> 'Python'.upper()
'PYTHON'
```

Testování řetězce

- Metoda

`.isalpha()`

- vrací True, pokud řetězec obsahuje pouze písmena, jinak False
- Metoda

`.isdigit()`

- vrací True, pokud řetězec obsahuje pouze číslice, jinak False.

- Příklad:

```
>>> "Python".isalpha()
True
>>> "Python".isdigit()
False
>>> "123".isdigit()
True
```

Hledání v řetězci

- Metoda

```
count(ret)
```

  vrátí počet výskytů řetězce ret v řetězci, který metodu volá.

- Příklad:

```
>>> "Anakonda".count("a")
2
>>> "Arara".count("ra")
2
```

Nahrazování podřetězce v řetězci

- Metoda

```
replace(co, čím)
```

v řetězci nahradí všechny výskyty řetězce co řetězcem čím

- Příklad:

```
>>> 'Základy programování v Pythonu 1.'.replace("Pythonu","C++")
'Základy programování v C++ 1.'
```

Regulární výrazy

Motivace

- Máme seznam firem. U některých je však nesprávně uvedeno as místo a.s. Jak to opravit?

```
firma="Python as"
firma=firma.replace("as","a.s.")
print(firma)
>>Python a.s.
```

- Co ale pro tuto firmu: „Hodiny a čas as“?

```
firma="Hodiny a čas as"
firma=firma.replace("as","a.s.")
print(firma)
>>Hodiny a ča.s. a.s.
```

- Řešením je použít regulární výraz.

Regulární výrazy

- Slouží k vyhledávání, nahrazování a rozkladu textu se složitými vzorci znaků.
- V Pythonu se pro práci s nimi používá modul re.
- V regulární výrazech se používají znaky:
- `^` - odpovídá začátku řetězce,
- `$` - odpovídá konci řetězce,
- `\b` - odpovídá hranici slov,
- `\d` - odpovídá číslici,
- `\D` – odpovídá znaku jinému než číslice,
- `x?` – odpovídá nepovinnému znaku x (0 nebo 1),
- `x*` - vyjadřuje 0 nebo více výskytů znaku x,
- `x+` - odpovídá x jedenkrát nebo víckrát,
- `x{n,m}` – znak x opakovaný n-krát až m-krát,
- `(a|b|c)` – odpovídá přesně jedno možnosti a, b nebo c,

Motivace

- U předchozího názvu firmy použijeme regulární výraz 'as$‘

```
import re
firma="Hodiny a čas as"
firma=re.sub('as$','a.s.',firma)
print(firma)
>>Hodiny a čas a.s.
```

- Co ale pro tuto firmu: „Hodiny as a Čas as“?
- import re
- firma="Hodiny as a Čas as"
- firma=re.sub('\\bas','a.s.',firma)
- print(firma)

```
>>Hodiny a.s. a Čas a.s.
```

- Pokud nechceme používat escape sekvence, můžeme použít surový (raw) řetězec:

```
firma=re.sub(r'\bas','a.s.',firma)
```

Příklad – Římská čísla

- U římských čísel se používá sedm znaků, které se opakují a kombinují různými způsoby, aby vyjádřily číselnou hodnotu.
  - `I = 1`
  - `V = 5`
  - `X =10`
  - `L = 50`
  - `C = 100`
  - `D = 500`
  - `M = 1000`

Příklad – Římská čísla

- Následují základní pravidla pro konstrukci římských čísel:
  - V některých případech se znaky sčítají. I je 1, II je rovno 2 a III znamená 3. VI se rovná 6 (doslova „5 a 1"), VII je 7 a VIII je 8.
  - Desítkové znaky (I, X, C a M) se mohou opakovat nanejvýš třikrát.
  - Hodnotu 4 musíme vyjádřit odečtením od dalšího vyššího pětkového zapsat jako IV („0 1 méně než 5"). 40 se zapisuje jako XL („0 10 méně než 50"), 41 jako XLI, 42 jako XLII, 43 jako XLIII a následuje 44 jako XLIV („0 10 méně než 50 a k tomu 0 1 méně než 5").
  - Někdy znaky vyjadřují opak sčítání. Když některé znaky umístíme před jiné, provádíme odčítání od konečné hodnoty. Například hodnotu 9 musíme vyjádřit odečtením od dalšího vyššího desítkového znaku: 8 zapíšeme jako VIII, ale 9 zapíšeme IX („0 1 méně než 1 0") a ne jako VIII I (protože znak I nemůžeme opakovat čtyřikrát). 90 je XC, 900 je CM.
  - Pětkové znaky se nesmí opakovat. 10 se vždy zapisuje jako X a nikdy jako W. 100 je vždy C, nikdy LL.
  - Římská čísla se čtou zleva doprava, takže na pořadí znaků velmi záleží. DC znamená 600, ale CD je úplně jiné číslo (400, „0 100 méně než 500"). CI je 101; IC není dokonce vůbec platné římské číslo (protože 1 nemůžeme přímo odčítat od 100; musíme to napsat jako XCIX, „o 10 méně než 100 a k tomu 0 1 méně než 10").

Příklad – Římská čísla

- Našim úkolem je napsat regulární výraz, který by ověřil, zda daný řetězec je římským číslem.

1. Kontrola tisícovek

```
import re
vzor="^M?M?M?$"
test=re.search(vzor,"M")
print(test)
test=re.search(vzor,"MM")
print(test)
test=re.search(vzor,"MMM")
print(test)
test=re.search(vzor,"")
print(test)
```

  - pokud řetězec odpovídá vzoru, vrátí metoda search objekt typu match, jinak none

Příklad – Římská čísla

2. Kontrola stovek

- Kontrola stovek je obtížnější než kontrola tisícovek.
- Je to tím, že v závislosti na hodnotě existuje několik vzájemně se vylučujících způsobů, kterými mohou být stovky vyjádřeny.

```
100 = C
200 = CC
300 = CCC
400 = CD
500 = D
600 = DC
700 = DCC
800 = DCCC
900 = CM
```

Příklad – Římská čísla

- Máme tedy čtyři různé vzory:
  - CM
  - CD
  - 0-3 znaky C
  - D následované 0 až 3 znaky C.

```
import re
vzor="^M?M?M?(CM|CD|D?C?C?C?)$"
test=re.search(vzor,"MCM")
print(test)
test=re.search(vzor,"MMD")
print(test)
test=re.search(vzor,"MMMDC")
print(test)
test=re.search(vzor,"MCMC")
print(test)
```

  - Vzor můžeme zjednodušit:

```
vzor="^M{0,3}(CM|CD|D?C{0,3})$"
```

Příklad – Římská čísla

3. Kontrola desítek

- Je to analogie se stovkami:

```
import re
vzor="^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})$"
test=re.search(vzor,"MCMXC")
print(test)
test=re.search(vzor,"MMDXL")
print(test)
test=re.search(vzor,"MMMDCL")
print(test)
test=re.search(vzor,"LXX")
print(test)
```

Příklad – Římská čísla

4. Kontrola jednotek

- Opět analogie:

```
import re
vzor="^M{0,3}(CM|CD|D?C{0,3})(XC|XL|L?X{0,3})(IX|IV|V?I{0,3})$"
test=re.search(vzor,"MCMXCIX")
print(test)
test=re.search(vzor,"MMDXLIV")
print(test)
test=re.search(vzor,"MMMDCLV")
print(test)
test=re.search(vzor,"LXXIII")
print(test)
```