**Reťazce**

už vieme:

* reťazec == postupnosť znakov uzavetá v ' ' alebo " "
* priradiť do premennej
* zreťaziť (zlepiť) dva reťazce
* násobiť (zlepiť viac kópií) reťazca
* načítať zo vstupu (pomocou **input**()) a vypisovať (pomocou**print**())
* vyrobiť z čísla reťazec (**str**()), z reťazca číslo (**int**(), **float**())
* rozobrať reťazec vo **for**-cykle

**znakové reťazce**

Znakový reťazec postupnosť znakov uzavetá v ' ' alebo " "

* môže obsahovať ľubovoľné znaky (okrem znaku ' v ' ' reťazci, a znaku " v " ")
* musí sa zmestiť do jedného riadku (nesmie prechádzať do druhého riadku)
* môže obsahovať špeciálne znaky:
  + \n - nový riadok
  + \' - apostrof
  + \" - úvodzovka
  + \\ - opačná lomka

**Napríklad**

>>> 'Monty\nPython'

'Monty\nPython'

>>> print('Monty\nPython')

Monty

Python

>>> print('Monty\\nPython')

Monty\nPython

**viacriadkové reťazce**

* reťazec, ktorý začína trojicou buď ' ' ' alebo " " " môže obsahovať aj ' a ", môže prechádzať cez viac riadkov (automaticky sa doplní \n)
* musí byť ukončený rovnakou trojicou ' ' ' alebo " " "

>>> macek = '''Išiel Macek

do Malacek

šošovičku mlácic'''

>>> macek

'Išiel Macek\ndo Malacek\nšošovičku mlácic'

>>> print(macek)

Išiel Macek

do Malacek

šošovičku mlácic

>>>

**základné operácie**

* štandardná funkcia **len()** vráti dĺžku reťazca (špeciálne znaky ako '\n', '\'', ... reprezentujú len 1 znak)

>>> a = 'Python'

>>> len(a)

6

>>> len('Peter\'s dog')

11

* reťazec + reťazec - výsledkom je zlepenie kópií oboch reťazcov

>>> 'Monty' + 'Python'

'MontyPython'

>>> 'ma' + 'ma ma' + ' Emu'

mama ma Emu

>>> str(12) + str(89)

'1289'

* reťazec \* číslo - výsledkom je zlepenie n-kópií reťazca

>>> '<=>' \* 10

'<=><=><=><=><=><=><=><=><=><=>'

>>> 'ma' + 'ma ' \* 2 + 'Emu'

mama ma Emu

>>> 'E' + 3 \* ' ma' + 'mu'

'E ma ma mamu'

**operácia in**

* zisťuje, či sa zadaný podreťazec nachádza v nejakom reťazci
  + *podreťazec* **in** *reťazec*
* najčastejšie sa bude využívať v príkaze **if** a v cykle **while**

>>> 'nt' in 'Monty Python'

True

>>> 'y P' in 'Monty Python'

True

>>> 'tyPy' in 'Monty Python'

False

>>> 'pyt' in 'Monty Python'

False

* ak niekedy budeme potrebovať negáciu tejto podmienky, môžeme zapísať

if not 'a' in retazec:

...

if 'a' not in retazec:

...

**indexovanie**

* reťazec je postupnosť znakov uzavretá v ' ' alebo " "
  + znaky v tejto postupnosti môžeme očíslovať a pristupovať k jednotlivým znakom
* hovoríme tomu **indexovanie** - používa sa na to operátor hranaté zátvorky: *reťazec***[** *číslo* **]**
  + celé číslo v hranatých zátvorkách = **index**
  + znaky sú indexované od **0** do **len()-1**
  + výsledkom je vždy 1-znakový reťazec (čo je nový reťazec s kópiou 1 znaku z pôvodného reťazca)

očíslujme znaky reťazca:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **o** | **n** | **t** | **y** |  | **P** | **y** | **t** | **h** | **o** | **n** |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |

Do premennej **abc** priradíme reťazec 12 znakov a pristupujeme ku niektorým znakom pomocou indexu:

>>> abc = 'Monty Python'

>>> abc[3]

't'

>>> abc[9]

'h'

>>> abc[12]

IndexError: string index out of range

>>> abc[len(abc)-1]

'n'

Vidíme, že posledný znak v reťazci má index **dĺžka reťazca - 1**. Ak indexujeme väčším číslom ako 11, vznikne chybová správa [**IndexError**](http://python2013.input.sk/IndexError)**: string index out of range**.

Často sa indexuje v cykle, kde premenná cyklu je správneho rozsahu, napr.

>>> a = 'Python'

>>> for i in range(len(a)):

print(i, a[i])

0 P

1 y

2 t

3 h

4 o

5 n

>>>

Funkcia **range(len(a))** zabezpečí, že cyklus prejde postupne pre všetky **i** od **0** do **len(a)-1**.

**indexovanie so zápornými indexmi**

* keďže často potrebujeme pristupovať ku znakom na konci reťazca, môžeme to zapisovať záporné indexy:
  + abc[-5] == abc[len(abc)-5]
* znaky reťazca sú takto indexované od **-1** do **-len()**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **o** | **n** | **t** | **y** |  | **P** | **y** | **t** | **h** | **o** | **n** |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

napríklad

>>> abc = 'Monty Python'

>>> abc[len(abc)-1]

'n'

>>> abc[-1]

'n'

>>> abc[-7]

' '

>>> abc[-13]

IndexError: string index out of range

alebo aj for-cyklom:

>>> a = 'Python'

>>> for i in range(1, len(a)+1):

print(-i, a[-i])

-1 n

-2 o

-3 h

-4 t

-5 y

-6 P

>>>

alebo for-cyklom so záporným krokom:

>>> a = 'Python'

>>> for i in range(-1, -len(a)-1, -1):

print(i, a[i])

-1 n

-2 o

-3 h

-4 t

-5 y

-6 P

>>>

**podreťazce**

* indexovať môžeme nie len jeden znak, ale aj nejaký podreťazec:
  + reťazec[prvý : posledný]
* **prvý** - index začiatku podreťazca
* **posledný** - index prvku **jeden za**, t.j. o 1 viac
* takejto operácii hovoríme **rez** (slice)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **o** | **n** | **t** | **y** |  | **P** | **y** | **t** | **h** | **o** | **n** |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  |  |  |  |  |  |  | ↑ |  |  |  |  | ↑ |

>>> abc[6:11]

'Pytho'

>>> abc[6:12]

'Python'

>>> abc[6:len(abc)]

'Python'

Podreťazce môžeme vytvárať aj v cykle:

>>> a = 'Python'

>>> for i in range(len(a)):

print(str(i)+':'+str(i+3), a[i:i+3])

0:3 Pyt

1:4 yth

2:5 tho

3:6 hon

4:7 on

5:8 n

>>>

alebo

>>> a = 'Python'

>>> for i in range(len(a)):

print(str(i)+':'+str(len(a)), a[i:len(a)])

0:6 Python

1:6 ython

2:6 thon

3:6 hon

4:6 on

5:6 n

>>>

**predvolená hodnota**

* ak neuvedieme prvý index v podreťazci => **od začiatku reťazca**
  + reťazec[ : posledný]
* ak neuvedieme druhý index v podreťazci => **až do konca reťazca**
  + reťazec[prvý : ]
* ak neuvedieme ani jeden index v podreťazci => **celý reťazec**
  + reťazec[ : ]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **o** | **n** | **t** | **y** |  | **P** | **y** | **t** | **h** | **o** | **n** |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

napríklad

>>> abc = 'Monty Python'

>>> abc[6:]

'Python'

>>> abc[:5]

'Monty'

>>> abc[-4:]

'thon'

**podreťazce s krokom**

* pri indexoch podreťazca môžeme určiť aj krok:
  + reťazec[prvý : posledný : krok]
* **krok** - určuje o koľko sa bude index v reťazci posúvať od **prvý** po **posledný**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **M** | **o** | **n** | **t** | **y** |  | **P** | **y** | **t** | **h** | **o** | **n** |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|  | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

napríklad

>>> abc = 'Monty Python'

>>> abc[2:10:2]

'nyPt'

>>> abc[::3]

'MtPh'

>>> abc[9:-7:-1]

'htyP'

>>> abc[::-1]

'nohtyP ytnoM'

>>> abc[6:] + ' ' + abc[:5]

'Python Monty'

>>> abc[4::-1] + ' ' + abc[:5:-1]

'ytnoM nohtyP'

>>> (abc[6:] + ' ' + abc[:5])[::-1]

'ytnoM nohtyP'

>>> 'kobyla ma maly bok'[::-1]

'kob ylam am alybok'

**reťazce sú v pamäti nemenné**

* nemenný typ (**immutable**) - nemôžeme meniť hodnotu, musíme skonštruovať novú

>>> abc[6] = 'K'

TypeError: 'str' object does not support item assignment

>>>

* všetky doterajšie manipulácie s reťazcami nemenili reťazec, ale vytvárali nový (niekedy vytvárali kópiu), napr.

>>> cba = abc[::-1]

>>> abc

'Monty Python'

>>> cba

'nohtyP ytnoM'

* **zmena v reťazci**

už vieme, že nefunguje:

>>> abc[6] = 'K'

TypeError: 'str' object does not support item assignment

>>>

musíme skonštruovať nový reťazec, napr.

>>> novy = abc[:6] + 'K' + abc[7:]

>>> novy

'Monty Kython'

>>> abc

'Monty Python'

alebo

>>> abc = 'm' + abc[1:-1] + 'N'

>>> abc

'monty PythoN'

**porovnávanie jednoznakových reťazcov**

môžeme porovnávať ==, !=, <, <=, >, >=, napr.

>>> 'x' == 'x'

True

>>> 'm' != 'M'

True

>>> 'a' > 'm'

False

>>> 'a' > 'A'

True

Python na porovnávanie používa vnútornú reprezentáciu: Unicode (UTF-8):

* funkcia **ord(*znak*)** vráti vnútornú reprezentáciu
  + ord('a') => 97
  + ord('A') => 65
* opačná funkcia **chr(*číslo*)** vráti jednoznakový reťazec, v ktorom znak má číselnú reprezentáciu
  + chr(66) vráti 'B'
* pri testoch sa porovnávajú vnútorné reprezentácie, t.j.

>>> ord('a') > ord('A')

True

>>> 97 > 65

True

>>> 'a' > 'A'

True

Vo for-cykle môžeme zisťovať vnútornú reprezentáciu niektorých znakov:

>>> for i in range(ord('A'), ord('J')):

print(i, chr(i))

65 A

66 B

67 C

68 D

69 E

70 F

71 G

72 H

73 I

**porovnávanie dlhších reťazcov**

* porovnáva sa po znakoch:
  + kým sú rovnaké znaky, preskakujem, pri prvom rôznom znaku, porovnám tieto dva znaky, napr. pri porovnávaní dvoch reťazcov 'kocur' a 'kohut':
    - porovná 0. znaky: 'k' == 'k'
    - porovná 1. znaky: 'o' == 'o'
    - porovná 2. znaky: 'c' <  'h'
  + preto platí, že 'kocur' < 'kohut'
* treba si dávať pozor na znaky s diakritikou, lebo, napr. ord('č') = 269 > ord('h') = 104 a teda
  + platí, že 'kočur' > 'kohut'

napr.

>>> 'kocur' < 'kohut'

True

>>> 'PYTHON' < 'Python' < 'python'

True

**prechádzanie reťazca v cykle**

Už sme videli, že prvky znakového reťazca môžeme prechádzať for-cyklom, v ktorom indexujeme celý reťazec postupne od **0** do **len()-1**

>>> a = 'Python'

>>> for i in range(len(a)):

print('.'\*i, a[i])

P

. y

.. t

... h

.... o

..... n

>>>

For-cyklus funguje aj keď priamo prechádzame nie postupnosť indexov (t.j. **range(len(a))**), ale postupnosť znakov, napr.

>>> for znak in 'python':

print(znak \* 5)

ppppp

yyyyy

ttttt

hhhhh

ooooo

nnnnn

>>>

Reťazec môžeme prechádzať aj while-cyklom, napr.

>>> a = '.....veľa bodiek'

>>> print(a)

.....veľa bodiek

>>> while len(a)!=0 and a[0]=='.':

a = a[1:]

>>> print(a)

veľa bodiek

>>>

Cyklus sa opakoval, kým bol reťazec neprázdny a kým boli na začiatku bodky '.': vtedy reťazec skracoval o prvý znak

**funkcie**

Zhrňme, čo vieme o štandardných funkciách:

* **len**() - dĺžka
* **int**() - prevod reťazca na celé číslo
* **float**() - prevod reťazca na desatinné číslo
* **str**() - prevod čísla na reťazec
* **ord**(), **chr**() - prevod do a z unicode

Ešte trojica funkcií:

* **bin**(), **hex**(), **oct**() - prevod do číselných sústav: dvojková, šestnástková, osmičková

napríklad

>>> bin(123)

'0b1111011'

>>> hex(123)

'0x7b'

>>> oct(123)

'0o173'

>>>

**vlastné funkcie**

Môžeme vytvárať vlastné funkcie, ktoré majú aj reťazcové parametre, resp. môžu vracať reťazcovú návratovú hodnotu. Niekoľko námetov:

* funkcia vráti True ak daný znak (jednoznakový reťazec) je číslica

def je\_cifra(znak):

return znak>='0' and znak<='9'

* funkcia vráti True ak daný znak (jednoznakový reťazec) je malé alebo veľké písmeno (anglickej abecedy)

def je\_pismeno(znak):

return znak>='a' and znak<='z' or znak>='A' and znak<='Z'

* parametrom funkcie je reťazec s menom a priezviskom (oddelené medzerou) - funkcia vráti reťazec priezvisko a meno (oddelené medzerou)

def meno(r):

ix = r.find(' ')

return r[ix+1:] + ' ' + r[:ix]

* funkcia vráti prvé slovo vo vete, kde slovo obsahuje len malé a veľké písmená (využijeme funkciu je\_pismeno)

def slovo(veta):

for i in range(len(veta)):

if not je\_pismeno(veta[i]):

return veta[:i]

return veta

**metódy**

* špeciálne funkcie, ktoré pracujú s reťazcami - vždy vracajú nejakú hodnotu
* ich formát:
  + reťazec.metóda(parametre)
  + **metóda** je meno niektorej z metód, ktoré sú už v systéme definované pre znakové reťazce

My si ukážeme niekoľko užitočných metód, s niektorými ďalšími sa zoznámime neskôr

* **count** - počet výskytov
* **find** - index prvého výskytu
* **lower**, **upper** - na malé, resp. veľké písmená
* **replace** - nahraď výskyty iným reťazcom
* **strip** - odstráň medzery na začiatku a na konci
* **format** - formátovaný výstup

**formátovanie reťazca**

* metóda **format** má zvyčajne takýto zápis:
  + 'formátovací reťazec'.format(parametre)
* kde **formátovací reťazec** môže obsahovať ľubovoľný text, ale pre metódu **format** sú zaujímavé len dvojice znakov  {}
* **parametre** pri volaní metódy **format** sú ľubovoľné hodnoty (ľubovoľných typov) - týchto parametrov by malo byť presne rovnaký počet ako dvojíc  {}  (špeciálnymi prípadmi sa tu zaoberať nebudeme)
* metóda **format** potom dosadí hodnoty svojich parametrov za zodpovedajúce dvojice  {}

**špecifikácia formátu**

V zátvorkách  {}  sa môžu nachádzať rôzne upresnenia formátovania, napr.:

* {:10}  - šírka výpisu 10 znakov
* {:>7}  - šírka 7, zarovnané vpravo
* {:<5d}  - šírka 5, zarovnané vľavo, parameter musí byť celé číslo (bude sa vypisovať v 10-ovej sústave)
* {:12.4f}  - šírka 12, parameter desatinné číslo vypisované na 4 desatinné miesta
* {:06x}  - šírka 6, zľava doplnená nulami, parameter celé číslo sa vypíše v 16-ovej sústave
* {:^20s}  - šírka 20, vycentrované, parametrom je reťazec

Zhrňme najpoužívanejšie písmená pri označovaní typu parametra:

* **d** - celé číslo v desiatkovej sústave
* **b** - celé číslo v dvojkovej sústave
* **x** - celé číslo v šestnástkovej sústave
* **s** - znakový reťazec
* **f** - desatinné číslo (možno špecifikovať počet desatinných miest, inak default 6)
* **g** - desatinné číslo vo všeobecnom formáte

**dokumentačný reťazec pri definovaní funkcie**

* ak funkcia vo svojom tele hneď ako prvý riadok obsahuje znakový reťazec (zvykne byť viacriadkový s ' ' '), tento sa stáva, tzv. **dokumentačným reťazcom** (docstring)
* pri vykonávaní tela funkcie sa takéto reťazce ignorujú (preskakujú)
* tento reťazec (docstring) sa vypíše napr. pri vyvolaní štandardného príkazu **help()**

def pocet\_vyskytov(podretazec, retazec):

'''funkcia vráti počet výskytov podreťazca v reťazci

prvý parameter podretazec - ľubovoľný neprázdny reťazec, o ktorom sa

bude zisťovať počet výskytov

druhý parameter retazec - reťazec, v ktorom sa hľadajú výskyty

ak je prvý parameter podretazec prázdny reťazec, funkcia vráti len(retazec)

'''

pocet = 0

for ix in range(len(retazec)):

if retazec[ix:ix+len(podretazec)] == podretazec:

pocet += 1

return pocet

>>> help(pocet\_vyskytov)

Help on function pocet\_vyskytov in module \_\_main\_\_:

pocet\_vyskytov(podretazec, retazec)

funkcia vráti počet výskytov podreťazca v reťazci

prvý parameter podretazec - ľubovoľný neprázdny reťazec, o ktorom sa

bude zisťovať počet výskytov

druhý parameter retazec - reťazec, v ktorom sa hľadajú výskyty

ak je prvý parameter podretazec prázdny reťazec, funkcia vráti len(retazec)

* okrem takéhoto spôsobu samodokumentácie funkcií si treba uvedomiť, že Python v tele funkcie ignoruje nielen všetky reťazce, ale aj iné konštanty
  + ak napr. zavoláme funkciu, ktorá vracia nejakú hodnotu a túto hodnotu ďalej nespracujeme (priradenie, použitie ako parametra inej funkcie, ...), vyhodnocovanie funkcie takúto návratovú hodnotu ignoruje
  + ak si uvedomíme, že meno funkcie bez okrúhlych zátvoriek nespôsobí volanie tejto funkcie, ale len hodnotu referencie na funkciu, tak aj takýto zápis sa ignoruje
  + napr. všetky tieto zápisy sa v tele funkcie ignorujú:

s.replace('a', 'b')

print

g.pack

pocet + 1

mat.sin(uhol)