Niz

Osnove programiranja

Nejc Ilc

Pa saj niz že poznamo

Na hitro ponovimo, potem odrinemo na globoko

Niz (podatkovni tip `str`) definiramo z enojnimi, dvojnimi ali trojnimi narekovaji:

```
niz = '''Vzkliknil je: "Pr' méj duš'!"'''
```

Nizu, ki ga obdajajo trojni narekovaji (*dvojni* ali *enojni* trojni), pravimo *docstring* in je namenjen dokumentiranju funkcij in programov.

```
def povprecje(seznam):
    """Vrne povprečje števil v seznamu."""
    return sum(seznam) / len(seznam)
```

Do dokumentacije pridemo z `help(povprecje)` ali pa preko lastnosti objekta: `povprecje.__doc__`.

Zaporedje znakov

Niz je zaporedje znakov. Z njim lahko delamo kot z drugimi zaporedji (seznam, terka, obseg, …):

```
>>> niz = 'In cvetja vonj je oba opajal.'
>>> niz[3]                              # Indeksiranje
'c'
>>> niz[-7:-1]                          # Rezine
'opajal'
>>> niz[:-7] + 'navdušil ' + niz[-1]*3 # Seštevanje
'In cvetja vonj je oba navdušil ...'   # in množenje
>>> for znak in niz:                    # Zanka for
...     print(znak, end="")
...
In cvetja vonj je oba opajal.
```

Nespremenljiv (kamen)

```
>>> niz[0] = 'i'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object doesn't support item assignment
```

Delo z nizi `preveri_zacetek`

Zelo pogosto delamo z nizi, zato je temu namenjeno celo predavanje

Napišimo funkcijo `se_zacne(s, p)`, ki preveri, ali se niz `s` začne z določenim podnizom `p` (niz, ki je vsebovan v nizu). Funkcija naj vrne `True`, če to drži, sicer `False`. Primeri:

```
se_zacne('Marko skače', 'Marko') → True
se_zacne('Marko skače', 'marko') → False
se_zacne('Marko skače', 'Marko skače na glavo') → False
```

Preveriti moramo torej ujemanje prvih `n` znakov nizov `s` in `p`, kjer je `n` dolžina niza `p`:

```
def se_zacne(s, p):
    """Vrne True, če se niz s začne s podnizom p."""
    return s[:len(p)] == p
```

Naša rešitev je precej elegantna in kratka, vendar jo je utrujajoče pisati vedno, ko jo potrebujemo.

Morda pa objekti tipa `str` vsebujejo kakšne koristne metode? Poglejmo!

```
>>> dir(str)
[..., 'capitalize', 'casefold', 'center', 'count',
'encode', 'endswith', 'expandtabs', 'find', 'format',
'format_map', 'index', 'isalnum', 'isalpha', 'isascii',
'isdecimal', 'isdigit', 'isidentifier', 'islower',
'isnumeric', 'isprintable', 'isspace', 'istitle',
'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'removeprefix',
'removesuffix', 'replace', 'rfind', 'rindex', 'rjust',
'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase',
'title', 'translate', 'upper', 'zfill']
```

Metode niza

Spoznajmo nekaj najbolj uporabnih

Velike in male črke

```
>>> niz = 'dua lipa'
>>> niz.upper()       # Spremenimo črke v velike.
'DUA LIPA'
>>> niz               # Metoda nam vrne nov objekt.
'dua lipa'
>>> niz = niz.upper() # Če želimo povoziti starega,
>>> niz               # moramo uporabiti prirejanje.
'DUA LIPA'
>>> niz.lower()       # Velike črke spremenimo v male.
'dua lipa'
>>> niz.capitalize()  # Samo prva črka bo velika.
'Dua lipa'
>>> niz = niz.title() # Vsaka beseda bo imela veliko
'Dua Lipa'            # začetnico. Shranimo niz.
>>> niz.swapcase()    # Zamenjamo velike in male črke.
'dUA lIPA'
```

Štetje, iskanje, zamenjave

```
>>> niz = 'priletela muha na zid'
>>> niz.count('a')        # Štetje pojavitev niza
3
>>> niz.index('l')        # Indeks prve pojavitve
3                         # niza 'l' v nizu `niz`.
>>> niz.index('l', 4)     # Lahko dodamo izbirni
7                         # argument - začetek iskanja.
>>> niz.index('l', 4, 7)  # In še konec iskanja.
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

```
>>> niz.find('l', 4, 7) # index vrne napako, če ne najde
-1                      # podniza. Metoda find pa -1.
>>> niz.find('l', 4, 8) # Tokrat je iskanje uspešno.
7                       # Iščemo po rezini niz[4:8].
>>> niz.replace('muha', 'čebela') # Zamenjava podniza.
'priletela čebela na zid'
>>> for samoglasnik in 'aeiou':
...   niz = niz.replace(samoglasnik, 'o')
>>> niz
'prolotolo moho no zod'
```

Metode niza

Ali je ...?

```
>>> '314'.isdigit()
True
>>> '3.14'.isdigit()
False
>>> 'abc'.isdigit()
False
>>> 'abc'.isalpha()
True
>>> '42b'.isalnum()
True
>>> 'Kolodvorska 42b'.isalnum()
False
>>> 'Kolodvorska 42b'.startswith('Kolo')
True
>>> 'Kolodvorska 42b'.startswith('kolo')
False
>>> 'Kolodvorska 42b'.endswith('42')
False
>>> 'Kolodvorska 42b'.endswith('42', -3, -1)
True
```

Tesanje

Odstranjevanje ponavljajočih znakov na začetku ali na koncu niza.

```
>>> niz = '  C-vitamin-----'
>>> niz.lstrip()    # Obtešemo niz z leve strani;
'C-vitamin-----'    # če ni argumenta, odstranimo
                    # "beli prostor": presledke,
                    # tabulatorje, nove vrstice.
>>> niz.rstrip('-') # Tesanje z desne strani.
'  C-vitamin'       # V argumentu lahko podamo
                    # znak, ki naj se odstrani.
>>> niz.strip(' -') # Tesanje z obeh strani.
'C-vitamin'         # Odstranijo se vsi znaki
                    # v argumentu metode.
```

Odstranjevanje predpon in pripon.

```
>>> 'img_01234.jpg'.removeprefix('img_')
'01234.jpg'
>>> 'img_01234.jpg'.removesuffix('.jpg')
'img_01234'
```

Metode niza

Razdelitev

Niz lahko razdelimo na podnize glede na razmejitveni znak (ang. *delimiter*).

```
# Definirajmo niz. Z uporabo trojnih narekovajev
# lahko niz zapišemo preko več vrstic.
# Macuo Bašō: Star ribnik
>>> haiku = """starodavni ribnik-
... skok žabe:
... pljusk vode"""
>>> haiku[:50]   # Izpišimo niz, vidimo znake \n
'starodavni ribnik-\nskok žabe:\npljusk vode'
# Klic metode split() brez argumentov razdeli
# niz po presledkih, tabulatorjih in novih
# vrsticah.
>>> haiku.split()
['starodavni', 'ribnik-', 'skok', 'žabe:',
'pljusk', 'vode']
```

```
# Lahko tudi podamo razmejitveni znak ali niz.
# Razdelimo haiuku samo po vrsticah:
>>> haiku.split('\n')
['starodavni ribnik-', 'skok žabe:', 'pljusk vode']
# Uporabimo razmejitveni niz: pomišljaj, ki mu sledi
# znak za novo vrstico.
>>> haiku.split('-\n')
['starodavni ribnik', 'skok žabe:\npljusk vode']
# Kaj pa, če bi radi razdelili po več kriterijih,
# denimo po novih vrsticah, presledkih, ločilih ...
>>> haiku.split('\n -:')
['starodavni ribnik-\nskok žabe:\npljusk vode']
# Hm, ne gre!
# Uporabimo trik: replace(). Vse željene razmejitve
# zamenjamo za presledke. Nato razdelimo niz.
>>> haiku2 = haiku.replace('\n',' ')
>>> haiku2 = haiku2.replace('-',' ').replace(':',' ')
>>> haiku2
'starodavni ribnik  skok žabe  pljusk vode'
>>> haiku2.split()
['starodavni', 'ribnik', 'skok', 'žabe', 'pljusk',
'vode']
# Tako! Dobili smo seznam vseh besed!
```

Metode niza

Spojitev

Zaporedje nizov lahko spojimo v enega z uporabo metode `join()`.

Osnovna sintaksa: `locilo.join(zaporedje)`

kjer je `zaporedje` pač zaporedje: niz, seznam, terka, množica, slovar, … Pomembno je, da so elementi zaporedja nizi.

```
>>> '-'.join('Petra')
'P-e-t-r-a'
>>> '+'.join(['Petra', 'Miha'])
'Petra+Miha'
>>> ' in '.join(['Petra', 'Miha'])
'Petra in Miha'
>>> niz = '2  3\n  6 \t\t 3'
>>> ', '.join(niz.split())  # Razdelimo in nato spojimo
'2, 3, 6, 3'
```

Poravnava `preglednica`

Metode `ljust`, `center` in `rjust` nam pomagajo poravnati niz levo/sredinsko/desno. Zelo priročno za izpis v obliki preglednice.

```
podatki = [
    ('ime', 'št. ur', 'plačilo (€)'),
    ('Lojze', 40, 320),
    ('Elizabeta', 20, 160),
    ('Ana', 440, 3520)
]
# Širina enega stolpca v preglednici
s = 12
# Gremo po terkah v seznamu in jih odpakiramo
for i, (ime, ure, placa) in enumerate(podatki):
    print(
        ime.ljust(s),
        str(ure).center(s),
        str(placa).rjust(s),
        sep='')
    # Za glavo tabele (indeks 0) pride črta
    if i == 0:
        print('-' * s * len(podatki[0]))
```

Obdelava besedila `lektor`

Uporaba metod niza za koristne namene

Denimo, da pišemo daljše besedilo (seminarska, poročilo, diploma, …). Če uporabljamo urejevalnik besedila (denimo Word, Pages, Writer), nam to orodje že avtomatično popravlja velike začetnice na začetku stavka, omogoča enostavno iskanje in zamenjavo besed, odstranjevanje dvojnih presledkov ipd. Vse to pa znamo sprogramirati tudi sami!

```
besedilo = ' čuk je nočna ptica iz družine sov.med\
 parjenjem slišimo čukov značilen zvok \
»kiuh«,  ki se pojavlja    v kratkih intervalih. '
```

V besedilu je več napak. Med drugim nas kolega opomni, da opis ne govori o čuku, temveč o velikem skoviku. Odpravimo nepravilnosti.

Napišimo funkcijo `lektor()`, ki sprejme niz in vrne popravljen niz. Pri tem:

1. odpravimo dvojne presledke;
2. niz razdelimo po povedih in popravimo veliko začetnico;
3. zamenjamo vse pojavitve podniza 'čuk' z 'veliki skovik'. Kaj pa različni skloni? In velika začetnica?

Vabljeni, da si ogledate rešitev v `lektor.py`.

Oblikovanje izpisa z *f-nizom*

Kako nadziramo število decimalnih mest, poravnavo ipd.

Denimo, da želimo izpisati število π:

```
>>> from math import pi
>>> print('Število π:', pi)
Število π: 3.141592653589793
```

Dodajmo piko na konec izpisa. Ta malenkost precej zaplete klic funkcije `print`:

```
>>> print('Število π: ', pi, '.', sep='')
Število π: 3.141592653589793.
```

Namesto te telovadbe, lahko uporabimo oblikovanje nizov. Spoznajmo f-niz (*f-string*).

```
>>> print(f'Število π: {pi}.')
Število π: 3.141592653589793.
```

F-niz torej definiramo s črko `f` (ali `F`) pred navednicami (ki so lahko `'`, `"`, `'''` ali `"""`). In nato lahko v zavitih oklepajih navedemo izraz, ki se avtomatično pretvori v niz.

F-niz omogoča tudi določanje števila mest in še mnogo dobrot.

```
>>> print(f'Število π: {pi:.3f}.') # 3 decimalna mesta
Število π: 3.142.
>>> print(f'Število π: {pi:.3g}.') # 3 mesta
Število π: 3.14.
>>> print(f'Število π: {pi:.3e}.') # Znanstveni zapis
Število π: 3.142e+00.
>>> print(f'Število π: {pi:08.3f}.') # Vodilne ničle
Število π: 0003.142.
```

F-niz: primer `preglednica_fniz`

```
"""Podobno kot v preglednica.py,
tokrat z uporabo f-niza"""

podatki = [
    ('ime', 'št. ur', 'plačilo (€)'),
    ('Lojze', 40, 320),
    ('Elizabeta', 20, 160),
    ('Ana', 440, 3520)
]
# Širina enega stolpca v preglednici
s = 12
# Gremo po terkah v seznamu in jih odpakiramo
for i, (ime, ure, placa) in enumerate(podatki):

    # Za glavo tabele (indeks 0) pride črta
    if i == 0:
        niz = f'{ime:<{s}}{ure:^{s}}{placa:>{s}}'
        niz += '\n' + '-' * s * len(podatki[0])
    else:
        niz = f'{ime:<{s}}{ure:^{s}}{placa:>{s},.2f}'
    print(niz)
```

Izpis je kar profesionalen:

```
ime            št. ur    plačilo (€)
------------------------------------
Lojze            40           320.00
Elizabeta        20           160.00
Ana             440         3,520.00
```