**Znakové řetězce**

Řetězec je postupnost znaků uzavřených v apostrofech, nebo uvozovkách. Řetězce se mohou načíst ze vstupu *input()* a vypsat pomocí funkce *print()*. Řetězce se dají spojovat, rozdělovat *for-cykle*, měnit na čísla *int(), float(),* a i z čísel *srt().*

Řetězec může obsahovat jakýkoliv znak, musí se vejít do jednoho řádku a může i obsahovat speciální znaky:

**\n** - nový řádek

**\t** - tabulátor

**\'** - apostrof

**\"** - uvozovka

**** - opačné lomítko

Speciální znaky začínající opačným lomítkem, vždy počítá jako jeden znak.

**Více řádkové řetězce**

jsou řetězce, které začínají a končí 3-mi apostrofy, nebo uvozovkami. Automaticky se dosazuje \n pro určení dalšího řádku.

**Operace - in**

používá se na to, abychom zjistili, zda v textovém řetězci se vyskytuje konkrétní znak, nebo řetězec znaků. Nejčastěji se používá v příkaze *if* a *while -cyklu*.

```
>>>'nt' in 'Monty Python'        #je „nt“ ve slově „Monty Python“ ?
True                             #ano
```

Můžeme také využít negaci podmínky **not**:

```
If 'a' not  in řetězec:          #pokud není „a“ v řetězci
```

**Řetězcové funkce:**

***len()*** - délka řetězce

***int(), float()*** - převod řetězce na celé anebo desetinné číslo

***bool()*** - převod řetězce na ***True*** anebo ***False*** (když je prázdny, výsledek bude ***False***)

***str()*** - převod čísla (i libovolné jiné hodnoty) na řetězec

***ord(), chr()*** - převod do a z Unicode

***bin()*** - převod celého čísla do řetězce, který reprezentuje toto číslo v dvojkové soustavě

***hex()*** - převod celého čísla do řetězce, který reprezentuje toto číslo v šestnáctkové soustavě

***oct()*** - převod celého čísla do řetězce, který reprezentuje toto číslo v osmičkové soustavě

**Řetězcové metody**

reťazec.metóda(parametre)

jedná se o speciální způsob zápisu volání funkce, kde metoda je jméno některé z metod, které jsou už v systému definované pro znakové řetězce.

řetězec.***count(podřetězec)*** - Metoda zjistí počet výskytů podřetězce v daném řetězci.

řetězec.***find(podřetězec)*** - Metoda najde první, nejvíc vlevo, výskyt podřetězce v daném řetězci, nebo vrátí -1, když ho nenajde.

řetězec.***lower()*** - Metoda vyrobí kopii daného řetězce a všechny velká písmena změní na malé. Nepísemné znaky nemění.

řetězec.***upper()*** - Metoda vyrobí kopii daného řetězce a všechny velká písmena změní na velká. Nepísemné znaky nemění.

řetězec.***upper*** - Metoda vyrobí kopii daného řetězce v kterém všechny výskyty podřetězce 1 předělá na podřetězec 2.
Může být volána víckrát za sebou - 'abradabra'.replace('ra', 'y').replace('by', 'ko')

řetězec.***strip()*** - Metoda vyrobí kopii daného řetězce v které vyhodí všechny mezerné znaky a odstraní mezery na začátku a na konci řetězce (mezery, \n, \t).

řetězec.***format(hodnoty)*** - vrátí řetězec, v kterém nahradí formátovací prvky '{}' zadanými hodnotami. Od Python 3.6 již stačí k formátování řetězce zadat jen malým písmenem "f"před úvodním apostrofem řetězce - f'{x}' - viz. formátování řetězce na následující stránce.

Pokud chceme získat **nápovědu k určité metodě**, musíme ji uvést do ***help()*** i s řetězcem (může být i prázdný) a tečkou - ***help(''.find)***

**ŘETĚZCE JSOU V PAMĚTI NEMĚNITELNÉ (IMMUTABLE)**

to znamená, že hodnota řetězce se v paměti nedá změnit, takže pokud chceme řetězec změnit, musíme vyrobit nový řetězec.

**Porovnávání jednoznakových řetězců**

jednoznakové řetězce můžeme porovnávat operátory: **==, !=, <, <=, >, >=,** Python na porovnávání používá reprezentaci znaků **Unicode(UTF-8).** S touto reprezentací můžeme dále pracovat pomocí funkcí **ord() a chr()**

Funkce ***ord()*** vrátí hodnotu reprezentace znaku v kódování paměti počítače

Funkce ***chr()*** je opačná funkce, která vrátí jednoznakový řetězec dle znaku kódování

Při porovnávání dvou znaků se porovnává jejich hodnota reprezentace znaku. Při porovnání více znaků se porovnává nejprve hodnota prvního čísla, pokud jsou stejná, pak hodnota druhého čísla, atd.

**Operace indexování []**

Pomocí indexování můžeme přistupovat k jednotlivým znakům. Zapisuje se:

*řetězec[číslo]*

Znaky v řetězci se indexují ***od 0 do len()-1***, tzn., že první znak má index 0 a poslední je o 1 menší než délka řetězce. Výsledkem indexování je vždy jednoznakový nový řetězec, nebo chybová hláška, pokud indexujeme mimo znaky řetězce - *IndexError: string index out of range*

Indexování se často používá v cykle, kde proměnná cyklu nabývá hodnoty indexů:

```
a ='Python'
for i in range(len(a)):
print(i, a[i])

0 P
1 y
2 t
3 h
4 o
5 n
```

Indexování je možné i **se zápornými hodnotami**. Znaky pak jsou indexované čísli ***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* |

**Podřetězce - řez / slice**

pokud chceme indexovat ne jen jeden znak, ale část řetězce, používá se k zápisu rozpětí znak dvojtečky:

*řetězec[prvý : zaposledný]*

Poslední znak musí být vždy o jednu pozici větší, než pořadí požadovaného znaku, protože první znak má číslo 0. Takto vyčleněnou část řetězce nazýváme **řez**, nebo **slice**.

```
'Python'[1:-1]          >>>  'ytho'
'Python'[-5:4]          >>>  'yth'
'Python'[-3:3]          >>>  ''
'Python'[1:-1][1:-1]    >>>  'th'
```

**Předvolená hodnota**

když v indexování podřetězce neuvedeme první znak, označujeme tak řez od začátku řetězce, a pokud neuvedeme druhý znak, označujeme, že řez se má provést až do konce.

Pokud bychom v indexu nechali pouze dvojtečku, znamená to, že se má přepsat celý řetězec.

**Podřetězce s krokem**

Pokud při indexování uvedeme druhou dvojtečku, značí se za ní, jak velkým krokem se má textový řetězec procházet.

*řetězec[první : za poslední : krok]*

**Formátování řetězce - několik užitečných prvků:**

*f'formátovací řetězec s hodnotami v {}'*

Formátovací řetězec obsahuje formátovací prvky, které se nacházejí v kudrnatých závorkách ***{}***. V nich se musí nacházet přímo nějaká hodnota a za ní se může nacházet určitá specifika oddělená dvojtečkou.

Python nejprve vyhledá všechny výskyty v kudrnatých závorkách a vyhodnotí (vypočítá hodnoty) výrazů, které jsou uvnitř závorek. Tyto hodnoty následně dosadí do řetězce na dané místo a ze všeho vyrobí jeden řetězec.

**Specifikace formátu**

*>>>r, g, b =100, 150, 200*
>>>farba =f'#{r:02x}{g:02x}{b:02x}'

V kudrnatých závorkách se mohou nacházet různé upřesnění formátování, kterými určujeme detaily, jak se budou vypočítané parametry převádět na řetězec.

**První číslo** za dvojtečkou většinou **označuje šířku** (počet znaků), do které se vloží řetězec.
Dále zde pak mohou být znaky na zarovnání ('<', '>', '^') a znaky na typ hodnoty ('d', 'f', …).

Například:

***'{hodnota:10}'*** - šířka výpisu 10 znaků

***'{hodnota:>7}'*** - šířka 7, zarovnané vpravo

***'{hodnota:<5d}'*** - šířka 5, zarovnané vlevo, parametr musí byť celé číslo (bude se vypisovat v 10-ové soustavě)

***'{hodnota:12.4f}'*** - šířka 12, parametr desetinné číslo vypisované na 4 desetinné místa

***'{hodnota:06x}'*** - šířka 6, zleva doplněná nulami, parametr celé číslo se vypíše v 16-ové soustavě

***'{hodnota:^20s}'*** - šířka 20, vycentrované, parametrem je řetězec

Nejpoužívanější písmena při označovaní typu parametrů:

***d*** - celé číslo v desítkové soustavě

***b*** - celé číslo v dvojkové soustavě

***x*** - celé číslo v šestnáctkové soustavě

***s*** - znakový řetězec

***f*** - desetinné číslo (možno specifikovat počet desetinných míst, jinak default 6)

***g*** - desetinné číslo ve všeobecném formáte

**Dokumentační řetězec v definování funkce**

Slouží pro popis funkce a jejích parametrů **a zadává se do těla funkce na první řádek**. Stává se tak dokumentačním řetězcem (*docstring*) a je možné jej vyvolat pomocí funkce ***help()***

Při vykonávání těla funkce se takovéto řetězce ignorují (přeskakují)

V případě víceřádkového textu se používá formát s 3 apostrofy, který umož zapsat text ve více řádcích.

**Automatické číslování přechodů v for-cyklu za pomoci enumerate()**

Nejpoužívanější zápis je používající funkci ***enumerate()***

```
for i, znak in enumerate('Python'):
print(i, znak)
```

Tato funkce očekává, že jako parametr dostane nějakou posloupnost (například posloupnost znaků) a její úlohou je každý prvek této posloupnosti očíslovat. Díky tomu *for-cyklus* dokáže naráz nastavovat 2 proměnné cyklu - proměnnou očíslování *(i)* a proměnnou pro hodnotu *(znak).*

Tento cyklus tedy projde 6x a pokaždé nastaví dvě proměnné:

```
0 P
1 y
2 t
3 h
4 o
5 n
```

**Procházení řetězce while-cyklem**

```
a ='.....veľa bodiek'
while len(a) !=0 and a[0] =='.':
   a = a[1:]
print(a)
>>> veľa bodiek
```

Cyklus se bude opakovat, dokud se na počátku objevuje tečka. Dalo by se to tak zapsat i tímto způsobem:

```
while a[0:1] =='.':
   a = a[1:]
```

**Některé vlastní funkce:**

Funkce vrátí hodnotu **True**, pokud je daný znak **číslicí**:

```
def je_cifra(znak):
return'0'<= znak <='9'
```

Funkce vrátí hodnotu **True**, pokud je daný znak malé nebo velké **písmenu** anglické abecedy:

```
def je_pismeno(znak):
return'a'<= znak <='z'or'A'<= znak <='Z'
```

Funkce na **prohození** jména a příjmení:

```
def meno(r):
ix =0
while ix <len(r) and r[ix] !=' ':     # najde mezeru
      ix +=1
return r[ix+1:] +' '+ r[:ix]
```

Funkce vrátí **první slovo** ve větě:

```
def slovo(veta):
for i inrange(len(veta)):
ifnot je_pismeno(veta[i]):
return veta[:i]
return veta
```