 Python akademie - lekce 1 - 17.10.2024

▼

01_02 Řetězce (strings)

jako datové typy

Zajímavé odkazy z této lekce:

- [Seznam všech escaping_characters](#)
 - [Doporučení pep8, pravidla pro obecné pojmenování](#)
-

▼ String

tedy **řetězec** je různě dlouhé uskupení znaků (písmen, čísel, speciálních symbolů).

▼ Jak zapsat string?

```
print(Python)
```



```
-----  
NameError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_13336\1731216898.py in <module>  
----> 1 print(Python)
```

```
NameError: name 'Python' is not defined
```


```
print('Python')
```

 Python


```
print("Python")
```

 Python

```
print(type('Python'))
```

 <class 'str'>

```
print(type("Python"))
```

 <class 'str'>


```
print('Python' == "Python")
```

 True

```
print(''Python'')
```


 Python

```
print('''
Python
je super
programovací jazyk
''')
```



```
Python
je super
programovací jazyk
```

```
print('
Python
je super
programovací jazyk
')
```



```
File "C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\2845944465.py",
line 1
  print('
    ^
SyntaxError: unterminated string literal (detected at line 1)
```

```
print('Python")
```

```

File "C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\632333053.py",
line 1
    print('Python")
      ^
SyntaxError: unterminated string literal (detected at line 1)

```

```
print(''Python'')
```

```
print(len(''
Python
je super
programovací jazyk
''))
)
```

```
print(len(''Radim''))
```

```
print(len(''
Radim
Radim
''))
)
```

Pomocí uvozovek můžeme zapisovat současně písmena, speciální znaky a také čísla:

```
print('Radim Jedlicka')
```

```
⇒ Radim Jedlicka
```

```
print('123456789')
```

```
⇒ 123456789
```

```
print(type('123456'))
```

```
⇒ <class 'str'>
```

```
print("!@#$$%^&*")
```

```
⇒ !@#$$%^&*
```

✓ Použití uvozovek

Občas budeme potřebovat zápis doplnit o uvozovky, jako součást textu. Zápis se může stát trochu obtížnější:

```
print('It's Friday')
```

```
File "C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\2085839381.py",
line 1
    print('It's Friday')
      ^
SyntaxError: unterminated string literal (detected at line 1)
```

Můžeme použít opačné uvozovky, než máme na začátku a konci stringu:

```
print("It's Friday")
```

```
It's Friday
```

```
print('It"s" Friday')
```

```
It"s" Friday
```

```
print("It's "kind of" Friday")
```

```
File "C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\1948690863.py",
line 1
    print("It's "kind of" Friday")
      ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

Můžeme použít tzv. *escape characters*. Jde o symbol zpětného lomítka. Ten interpretu Pythonu oznamuje, že cokoliv, co bude za lomítkem následovat je speciální symbol:

```
print("It's \"kind\" of Friday")
```

```
It's "kind" of Friday
```

```
print("It's \"kind\" of \n\tFriday")
```

Použití **speciálních symbolů** souvisejících se zpětným lomítkem je víc. Jsou to tzv. **escape characters**. V tabulce níž najdeš soupis těch nejčastějších:

Speciální znak	Význam
\'	Apostrof
\	Zpětné lomítko
\n	Nový řádek
\r	<i>Return carriage</i>
\t	Tabulátor
\b	<i>Backspace</i>
\f	<i>Form feed</i> (konec stránky)

Speciální symboly uvidíš, pokud si nyní vyzkoušíš zápis s třemi uvozovkami na začátku a na konci:

```
print("""  
Ahoj,  
vsem!  
""")
```

```
"" "  
Ahoj,  
vsem!  
"" "
```

Už při zápisu v interpretu si všimni, že ti dovolí pokračovat na **dalších řádcích** (pomocí třech teček ...).

To je současně i podstatou třech uvozovek na začátku a na konci. Umožní ti zapsat hodnotu typu str na více řádků.

✓ Převádění datových (některých) typů (přetypování)

V některých situacích budeme muset pracovat s různými datovými typy a v takových situacích se velice často chybí:

```
print(2 + "2")
```



```
-----  
TypeError                                Traceback (most recent call last)  
~\AppData\Local\Temp\ipykernel_13336\61943562.py in <module>  
----> 1 print(2 + "2")  
  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
print("2" + "2")
```



```
22
```

```
print('a' + 'a')
```



```
aa
```

```
print(2 + 2)
```



```
4
```

```
print(2 + 2.0)
```



```
4.0
```

Naštěstí je možné **některé** datové typy převádět:

```
print("2")
```



```
2
```

```
print(type('2'))
```



```
<class 'str'>
```

```
print(int('2'))
```



```
2
```

```
print(type(int('2')))
```



```
<class 'int'>
```

Pomocí více zabudovaných funkcí, které zapíšu za sebe, můžeme jednak datový typ převést a současně jej ověřit:

```
print(type(int("2")))
```

Přetypovat můžeme s pomocí funkcí `str`, `int`, `float`, ... Ale některé znaky není možné převádět:

```
print(int("@"))
```



```
-----
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13336\4067393531.py in <module>
----> 1 print(int("@"))

ValueError: invalid literal for int() with base 10: '@'
```

✓ Spojování (concatenation)

Jde o proces, kdy použijeme operátor `+`, kdy po obou stranách operátoru máme string.

```
print('Radim' + 'Jedlicka')
```



```
RadimJedlicka
```

```
print('Radim ' + 'Jedlicka')
```



```
Radim Jedlicka
```

```
print('Radim' + ' ' + 'Jedlicka')
```



```
Radim Jedlicka
```

```
print('Radim' + ' ' + 'Jedlicka')
```



```
Radim Jedlicka
```

Výsledkem je spojení obou stringů v jeden (*concatenation*)

✓ Opakování (~repetition)

Jde o proces, kdy zadaný string vypíšeme n-krát po sobě (~repetition)

```
"str" * n
```

```
print('Radim' * 3)
```

```
→ RadimRadimRadim
```

```
print('=' * 10)
```

```
→ =====
```

▼ Indexování

Znaky, ze kterých je string složený, mají **pořadí**. Toto pořadí je určené celým číslem, tzv. *index*. Pomocí tohoto indexu máme možnost vybrat konkrétní znak.

Pokud chceme získat jen část z původního stringu, můžete si ji "vyříznout" (*slicing*):

STRING:

"AHOJ, VSEM"

INDEXY OD ZAČÁTKU:

0	1	2	3	4	5	6	7	8	9
A	H	O	J	,		V	S	E	M
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

INDEXY OD KONCE:

INDEXOVÁNÍ STRINGŮ

Ihned za konkrétní hodnotu napíšeme hranatou závorku a do ní celé číslo indexu, který potřebujeme:

```
print("AUTOBUS"1)    # bez hranaté závorky nelze indexovat
```



```
File "C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\1523599280.py",
line 1
    print("AUTOBUS"1)    # bez hranaté závorky nelze indexovat
      ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

```
print("AUTOBUS"[1.0]) # bez celého čísla v hranaté závorce také ne
```

```
<>:1: SyntaxWarning: str indices must be integers or slices, not float; perhaps you r
<>:1: SyntaxWarning: str indices must be integers or slices, not float; perhaps you r
C:\Users\Radim Jedlicka\AppData\Local\Temp\ipykernel_13336\1611801611.py:1: SyntaxWar
    print("AUTOBUS"[1.0]) # bez celého čísla v hranaté závorce také ne
-----
TypeError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_13336\1611801611.py in <module>
----> 1 print("AUTOBUS"[1.0]) # bez celého čísla v hranaté závorce také ne

TypeError: string indices must be integers
```

```
print('AUTOBUS'[1])
```

```
U
```

Příklad výš můžeš přečíst jako *Ze stringu autobus mi vypiš znak na indexu 1*. Z toho vyplývá, že indexování obecně začíná celým číslem 0, tedy první znak.

```
print("AUTOBUS"[0])
```

```
A
```

```
print("AUTOBUS"[2])
```

```
T
```

```
print("AUTOBUS"[3])
```

```
O
```

```
print("AUTOBUS"[-1])
```

```
S
```

Pokud budeš potřebovat indexovat **od konce**, můžeš pracovat s negativním indexem. Poslední hodnotu získáš pomocí indexu -1, předposlední hodnotu pomocí indexu -2, atd.

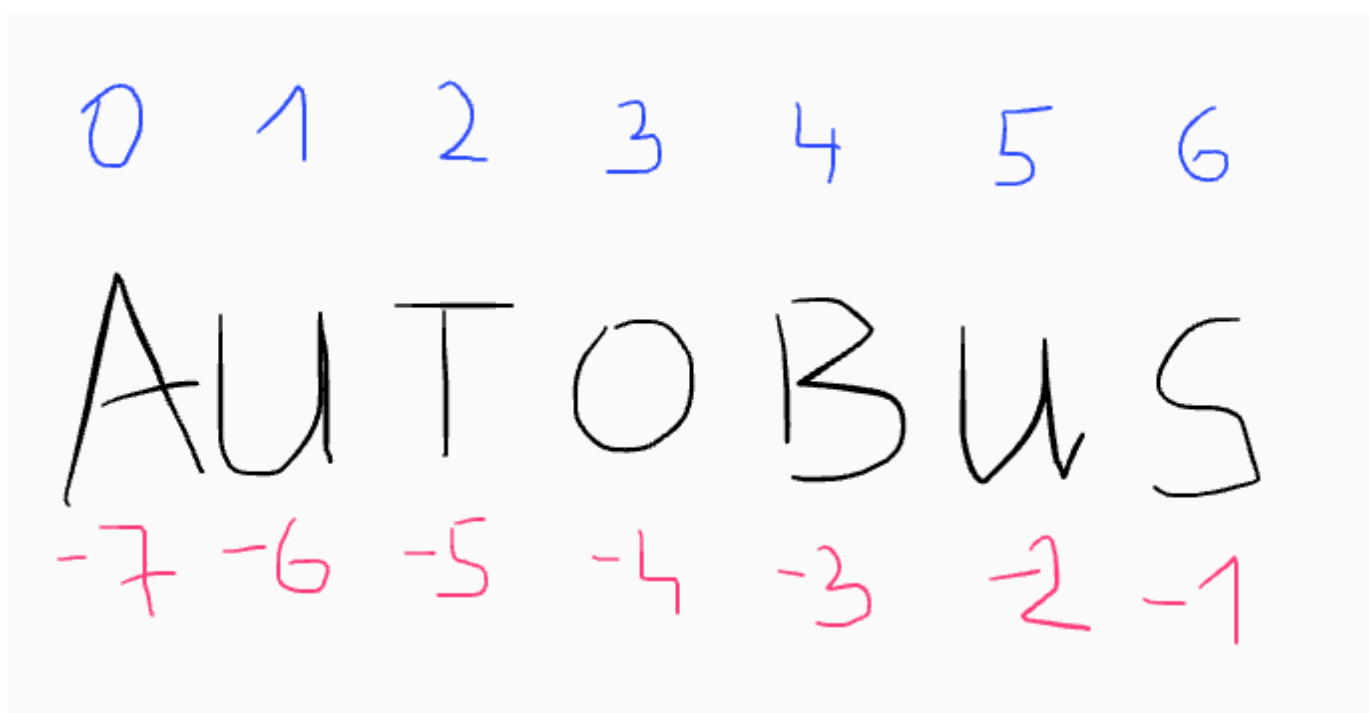
- 💡 Pro zvědavé: proč se indexuje od nuly?

✓ Slicing - část stringu

Pokud budete potřebovat pracovat pouze s částí datového typu `str`, můžete jej rozkrájet (z angl. slicing).

```
print("AUTOBUS"[0:4])
```

⇒ AUTO



Stejně jako u **indexování** použijeme hranatou závorku. Tentokrát ji doplníme **dvojtečkou** a **dalším celým číslem**. První hodnota je potom **počáteční index**, druhá hodnota je **konečný index**.

Opatrně **druhý index** je braný **nepřímo**, takže abys získal znak z indexu 6, musíš napsat číslo 7:

```
print("AUTOBUS"[0:6])
```

⇒ AUTOBU

```
print("AUTOBUS"[1:7])
```

⇒ UTOBUS

Dále je možné **zápis zkrátit**. Pokud si budeš zápisem s pomocí indexů jistý, můžeš vyzkoušet následující:

1. "autobus"[:2] - vynecháš první index a začneš dvojtečkou (původně [0:2])
2. "autobus"[2:] - vynecháš druhý index a končíš dvojtečkou (původně [2:7])

```
print("AUTOBUS"[:2])      # první dvě písmena
```

⇒ AU

```
print("AUTOBUS"[:2])      # první dvě písmena, zápis jedním číslem
```

⇒ AU

```
print("AUTOBUS"[2:7])     # od třetího písmena do konce
```

⇒ TOBUS

```
print("AUTOBUS"[2:])
```

⇒ TOBUS

```
print("AUTOBUS"[2:10])
```

⇒ TOBUS

✓ Striding

Nakonec operace známá jako **přeskakování** (~*striding*), umožňuje získat každý n-tý údaj ze stringu.

Doplníme **třetí celočíselnou hodnotu** do hranaté závorky, oddělenou dvojtečkou:

```
print("AUTOBUS"[0:7:2])
```

⇒ ATBS

Hodnota 2 zapsaná v hranaté závorce výš říká, že vezme nejprve index 0, a potom každý druhý index (tedy indexy 2, 4, 6).

```
print("AUTOBUS"[1:7:2])
```

↩️ UOU

Upravíme hodnoty v hranatých závorkách, ale platí pořád stejné pravidlo. Vezmeme nejprve index 1 (tedy "u"), a potom každý druhý index (tentokrát indexy 3 , 5).

Opět je možné zápis **zkrátit**. Vynech hodnoty indexů pro **počátek** a **konec**. Zapišeš pouze dvě dvojtečky a poslední hodnotu pro **přeskakování**:

```
print("AUTOBUS"[::2])
```

↩️ ATBS

```
print("AUTOBUS"[::3])
```

↩️ AOS

Dokonce můžeš použít **zápornou hodnotu** pro přeskakování pozpátku:

```
print("AUTOBUS"[::-1]) # obracene poradi, zacne 's', jeden znak za druhym
```

↩️ SUBOTUA

```
print("AUTOBUS"[::-2]) # obracene poradi, zacne 's', pote kazdy druhy znak
```

↩️ SBTA

```
print("AUTOBUS"[::-3]) # obracene poradi, zacne 's', pote kazdy treti znak
```

↩️ SOA