KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
jan.lastovicka@upol.cz
www.inf.upol.cz/lide/jan-lastovicka
17. listopadu 12, 771 46 Olomouc

Základy programování pro IT 1

7. řetězce

verze z 11. listopadu 2024

Pro práci s textem se používají **řetězce**. Řetězec je podobně jako pole složená hodnota. Oproti polím se však řetězec skládá ze znaků. Řetězec můžeme získat tak, že jeho znaky vložíme mezi jednoduché nebo dvojité uvozovky. Například:

```
>>> "Jablko"
'Jablko'
>>> 'Jablko'
'Jablko'
>>> "žluťoučký"
'žluťoučký'
>>> "Mezi poli"
'Mezi poli'
```

Funkce `print` tiskne řetězce bez řetězec ohraňujících uvozovek. Například:

```
>>> print("Ahoj světe")
Ahoj světe
```

Ty lze k řetězci přidat vestavěnou funkcí `repr`:

```
>>> repr("Ahoj světe")
"'Ahoj světe'"
```

Tisk včetně uvozovek zařídíme následovně.

```
>>> print(repr("Ahoj světe"))
'Ahoj světe'
```

Jak již bylo zmíněno, řetězce se podobají polím. Délku řetězce (počet znaků) můžeme získat funkcí `len`:

```
>>> len("Jablko")
6
>>> len("")
0
>>> len("a")
1
```

Řetězec `""` má délku nula a nazývá se **prázdným řetězcem**. Řetězec délky jedna nazýváme **znakem**. Znak na určitém indexu získáme operátorem indexace:

```
>>> s = "Jablko"
>>> s[0]
'J'
>>> s[1]
'a'
>>> s[5]
'o'
```

Operátor rovnosti rozhoduje, zda se dva znaky rovnají. Například:

```
>>> "a" == "a"
True
>>> "a" == "b"
False
```

Operátor `==` lze také použít k rozhodování o rovnosti řetězců. Podmínka

(string1 == string2)

je splněna právě tehdy, když jsou si řetězce *string1* a *string2* rovny. Například:

```
>>> "auto" == "toau"
False
>>> "auto" == "auto"
True
>>> "auto" == "autobus"
False
```

Spojit dva řetězce lze operátorem `+`. Například:

```
>>> "troj" + "úhelník"
'trojúhelník'
```

Zatím jsme uvažovali pouze pole, která obsahují čísla. Nyní se podíváme i na pole řetězců. Pole může kromě čísel obsahovat i řetězce:

```
>>> ["Ahoj", "světe"]
['Ahoj', 'světe']
```

Otázky a úkoly k semináři

1. Napište funkci `count_char`, která vrátí počet výskytů znaku v řetězci. Například:

   ```
   >>> count_char('ř', 'řeřicha')
   2
   ```

2. Napište funkci `is_substring_from`, která očekává řetězce *`string1`*, *`string2`* a číslo *`index`*. Funkce rozhodne, zda *`string2`* obsahuje *`string1`* od indexu *`index`*. Například:

   ```
   >>>  is_substring_from("úhelník", "trojúhelník", 4)
   True
   >>>  is_substring_from("úhelník", "trojúhelník", 3)
   False
   >>>  is_substring_from("čtyř", "trojúhelník", 6)
   False
   >>>  is_substring_from("", "dva", 3)
   True
   >>>  is_substring_from("", "", 0)
   True
   ```

3. Napište funkci `is_substring`, která rozhodne, zda je v jednom řetězci obsažen druhý řetězec. Například:

   ```
   >>>  is_substring("úhelník", "trojúhelník")
   True
   >>>  is_substring( "čtyř", "trojúhelník")
   False
   >>>  is_substring("", "čtyř")
   True
   >>>  is_substring("", "")
   True
   ```

4. Napište funkci `substring_count`, která spočítá kolikrát se jeden řetězec vyskytuje v druhém jako podřetězec. Například:

   ```
   >>> substring_count("ananas", "an")
   2
   >>> substring_count("aaaa", "aa")
   3
   ```

5. Upravte funkci z předchozího úkolu tak, aby se započítaly pouze výskyty, které se nepřekrývají. Například:

```
>>> substring_count2("ananas", "an")
2
>>> substring_count2("aaaa", "aa")
2
```

6. Napište funkci `are_strings_equal`, která rozhodne, zda jsou dva řetězce stejné (obsahují stejné znaky ve stejném pořadí). Operátor `==` můžete použít pouze k porovnávání znaků a čísel. Například:

```
>>>  are_strings_equal("auto", "toau")
False
>>>  are_strings_equal("auto", "auto")
True
>>>  are_strings_equal("auto", "autobus")
False
```

7. Napište predikát `is_palindrom`, který rozhodne, zda je řetězec palindrom. Například:

```
>>> is_palindrom("madam")
True
```

8. Napište funkci `get_substring` očekávající řetězec a dvě nezáporná čísla `m` a `n`. Funkce vrátí podřetězec řetězce obsahující právě ty znaky, které jsou na indexech větších nebo rovny `m` a menších než `n`. Například:

```
>>> get_substring("makovka", 2, 5)
'kov'
```

9. Napište funkci `reverse_string`, která pro řetězec vrátí řetězec s převráceným pořadím znaků. Například:

```
>>>  reverse_string("melodie")
'eidolem'
```

10. Napište znovu funkci na rozhodování, zda je řetězec palindrom. Tentokrát použijte funkci `reverse_string` z předchozího úkolu.

11. Napište funkci `replace_char`, která očekává řetězec *`string1`*, znak *`char`* a řetězec *`string2`*. Funkce nahradí všechny výskyty znaku *`char`* v řetězci *`string1`* za *`string2`*. Například:

```
>>> replace_char("pole", "o", "ukr")
'pukrle'
```

12. Napište funkci `replace`, která očekává tři řetězce *`string1`*, *`string2`* a *`string3`*. Funkce nahradí všechny výskyty *`string2`* v *`string1`* za *`string3`*. Například:

```
>>> replace("trojúhleník", "troj", "čtyř")
'čtyřúhelník'
>>> replace("aa", "a", "bb")
'bbbb'
```

13. Napište funkci `split`, která očekává dva řetězce *`string`* a *`separator`*. Funkce vrátí pole podřetězců řetězce *`string`* rozděleného řetězcem *`separator`*. Například:

```
>>> split("borůvky, ostružiny, maliny", ", ")
['borůvky', 'ostružiny', 'maliny']
```

14. Napište naopak funkci `join`, která očekává pole řetězců *`array`* a řetězec *`connector`*. Funkce spojí řetězce v poli tak, že mezi sousední vloží řetězec *`connector`*. Například:

```
>>> join(["borůvky", "ostružiny", "maliny", ", ")
'borůvky, ostružiny, maliny'
```