FAKULTA
INFORMATIKY
Masarykova univerzita

IB111 Základy programovania

Cvičenie 5 – reťazce a zoznamy

Matej Troják

(Poďakovanie: Valdemar Švábenský)

1. Reťazec (string)
2. Zoznam (list)
3. Priraďovanie v Pythone
4. Úlohy na zoznamy
5. Úlohy na reťazce

Reťazec (string)

- Postupnosť znakov (od Pythonu 3 – symboly Unicode) uzavretých:
  - V apostrofoch: `my_string ='Brian'`
  - V úvodzovkách: `my_string ="Brian"`
- preferencia podľa názoru, ale konzistentne
- apostrof v reťazci – *escaping*

```
>>> print('I‘ m Brian!')
SyntaxError: invalid syntax
>>> print('I\‘m Brian!')
I‘m Brian!
>>> print("I‘m Brian!")
I‘m Brian!
```

Dĺžka reťazca

- Funkcia `len()`:

```
>>> my_string = 'Python'
>>> len(example)
6
```

- Prázdny reťazec = prázdne apostrofy / úvodzovky:

```
>>> empty = ''
>>> type(empty)
<class 'str'> # It really is a string
>>> len(empty)
0
```

Prístup k znakom reťazca

- Jednotlivé znaky reťazca sú indexované celými číslami
- Indexuje sa od 0 po `len(string)-1`!

```
>>> my_string = 'Python'
```

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

- Syntax sprístupnenia znakov: `string[index]`

```
>>> first_letter = my_string[0]
>>> print(first_letter)
'P'
>>> my_string[3]
'h'
```

Rozsahy indexov

```
>>> my_string = 'Python'
```

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

```
>>> my_string[2:4]      # start + end index
'th'
>>> my_string[2:]       # start only, default end
'thon'
>>> my_string[:4]       # end only, default start
'Pyth'
```

Rozsahy indexov (2)

```
>>> my_string = 'Python'
```

Znak	P	y	t	h	o	n
Index	-6	-5	-4	-3	-2	-1

```
>>> my_string[-4:-1]    # start and end
'tho'
>>> my_string[:-2]      # end only
'Pyth'
>>> my_string[-4:]      # start only
'thon'
```

- Pozor na kombinácie

```
>>> my_string[5:-1]     # position -1 == 5
''
```

Rozsahy indexov (3)

- Možnosť špecifikovať krok pri prechode reťazcom
- Syntax: `string[start:end:step]`

```
>>> my_string = 'Python'
>>> my_string[::2]
'Pto'
```

- Tajná správa – ako ju správne vypísať len pomocou indexov?

```
>>> message = '!XeXgXaXsXsXeXmX XtXeXrXcXeXsX XeXhXtX XmXaX XI'
```

Prechádzanie reťazcov

- Cyklus `for`

```
>>> string = 'spam!'
>>> for character in string:
        print(character)
s
p
a
m
!
```

Operátory na reťazcoch

- "Sčítanie"

```
>>> 'King' + 'Arthur'
'KingArthur'
>>> 'King ' + 'Arthur'
'King Arthur'
>>> 'King ' + 'Arthur ' + str(1) + 'st'
'King Arthur 1st'
```

- Spájanie rezov

```
>>> king = 'King Arthur 4th'
>>> king[:12] + '5' + king[13:]
'King Arthur 5th'
```

- "Násobenie"

```
>>> 'king' * 3
'kingkingking'
```

Operátory in, not in

- Test na výskyt znaku / podreťazca v reťazci

```
>>> 'A' in 'awesome'
False
>>> 'dying' in 'studying'
True
>>> 'end' not in 'friend'
False
>>> '' in 'anything'
True
```

Užitočné funkcie na reťazcoch

- `lower()` – vráti reťazec v lower case
- `upper()` – vráti reťazec v upper case

```
>>> sentence = 'Don\'t try. Do, or do not.'
>>> sentence.upper()
"DON'T TRY. DO, OR DO NOT."
```

- `count(s)` – spočíta výskyt reťazca s v danom reťazci

```
>>> sentence.count('o')
5
```

- `find(s)` – nájde prvý index výskytu reťazca s v danom reťazci

```
>>> sentence.find('o')
1
>>> sentence.find('try')
6
```

https://docs.python.org/3/library/stdtypes.html#string-methods

Užitočné funkcie na reťazcoch (2)

- `index(s)` – ako find(s), vyhodí error ak sa v danom reťazci nenachádza

```
>>> sentence = 'Don\'t try. Do, or do not.'
>>> sentence.index('x')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
```

- `replace(a, b)` – vráti nový reťazec, ktorý v danom reťazci nahradí všetky výskyty reťazca a reťazcom b

```
>>> sentence.replace('o', '0')
"D0n't try. D0, 0r d0 n0t."
```

- `split()` – rozdelí reťazec podľa špecifikovaného oddeľovača

```
>>> sentence.split()
["Don't", 'try.', 'Do,', 'or', 'do', 'not.']
>>> sentence.split('o')
['D', "n't try. D", ', ', 'r d', ' n', 't.']
```

Užitočné funkcie na znakoch

- Znak je reťazec dĺžky 1 (teda je stále typu `str`)
- Funkcia `ord(c)`, kde c je znak, vracia celočíselný ASCII kód znaku

```
>>> ord("A")
65
>>> ord("X")
88
```

- Funkcia `chr(i)`, kde i je celé číslo, vracia znak s ASCII kódom i

```
>>> chr(65)
'A'
>>> chr(88)
'X'
```

http://www.ascii-code.com/

1. Reťazec (string)
2. Zoznam (list)
3. Priraďovanie v Pythone
4. Úlohy na zoznamy
5. Úlohy na reťazce

Zoznam (list)

- Štruktúrovaný dátový typ pre uchovávanie viacerých položiek v danom poradí ľubovoľného typu a počtu
- Definícia: `list_name =[item_1, item_2, ... ]`

```
>>> numbers = ['one', 'two', 'three', 'four', 'five']
```

- Sprístupnenie položiek: cez index

```
>>> numbers[2]
'three'
```

- Zmena položky: cez index (zoznam je *meniteľný*)!

```
>>> numbers[3] = 'styri'
>>> numbers
['one', 'two', 'three', 'styri', 'five']
```

Rezy zoznamov

- Ako u reťazcov:
  - Počiatočný aj koncový index
  - Len počiatočný / len koncový index
  - Záporné indexy
  - Krok

```
>>> numbers = ['one', 'two', 'three', 'four', 'five']
>>> numbers[2:]
['three', 'four', 'five']
>>> numbers[:-1]
['one', 'two', 'three', 'four']
>>> numbers[1:3]
['two', 'three']
```

Prechádzanie zoznamu

- Pomocou cyklu `for`

```
lst = [List of some kind]
for item in lst:
    # Do something for every item
```

```
for prime in [2, 3, 5, 7]:
    print(prime)
```

```
lst = [List of some kind]
for index in range(len(lst)):
    # Do something for each item lst[index]
```

```
lst = ["x", "y", "z"]
for index in range(len(lst)): # [0, 1, 2]
    print(lst[index])
```

Aritmetické operátory na zoznamoch

- Ako u reťazcov (+, *)

```
>>> list_a = [1, 2, 3, 4]
>>> list_b = ['a', 'b', 'c', 'd']
>>> list_a + list_b
[1, 2, 3, 4, 'a', 'b', 'c', 'd']
>>> list_a * 2
[1, 2, 3, 4, 1, 2, 3, 4]
```

Operátory in, not in

- Ako u reťazcov
- Test na výskyt položky v zozname

```
>>> numbers = ['one', 'two', 'three', 'four', 'five']
>>> 'fi' + 've' in numbers
True
>>> 'six' + 've' in numbers
False
>>> 'six'  in numbers
False
```

Funkcia range()

- Funkcia `range()` vracia *iterátor*
  - štruktúra na efektívne prechádzanie
- Volanie `list(range())` generuje zoznam

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(2, 5))
[2, 3, 4]
>>> list(range(2, 10, 3))
[2, 5, 8]
```

1. Reťazec (string)
2. Zoznam (list)
3. Priraďovanie v Pythone
4. Úlohy na zoznamy
5. Úlohy na reťazce

(Ne)meniteľné dátové typy

- Nemeniteľné typy: **FBITS**
  - **F**loat, **B**ool, **I**nt, **T**uple, **S**tr
  - Hodnoty nie je možné zmeniť
  - Ak je argument funkcie tohto typu, jeho hodnota sa skopíruje do funkcie
  - Pôvodný parameter je nezmenený
- Meniteľné typy: **SOLD**
  - **S**et, **O**bject, **L**ist, **D**ict
  - Hodnoty je možné zmeniť
  - Ak je argument funkcie tohto typu, zdieľa sa s funkciou
  - Môže dôjsť k tzv. vedľajším efektom (zmene argumentu)

`http://stackoverflow.com/questions/8056130/immutable-vs-mutable-types`

Operátor is

- Test na identitu objektov (rovnaká inštancia), nie na rovnosť prvkov

```
>>> x = [1, 2, 3]
>>> y = [1, 2, 3]
>>> x == y          # Equal elements
True
>>> x is y          # But different objects
False
```

```
>>> a = [1, 2, 3]
>>> b = a
>>> a is b          # The same objects
True
```

Kopírovanie zoznamov – vytvorenie aliasu

```
>>> badguys1 = ["Joker", "Bane"]
>>> badguys2 = badguys1                    # New reference
>>> print(badguys1)
['Joker', 'Bane']
>>> print(badguys2)
['Joker', 'Bane']
>>> badguys1 is badguys2
True
```

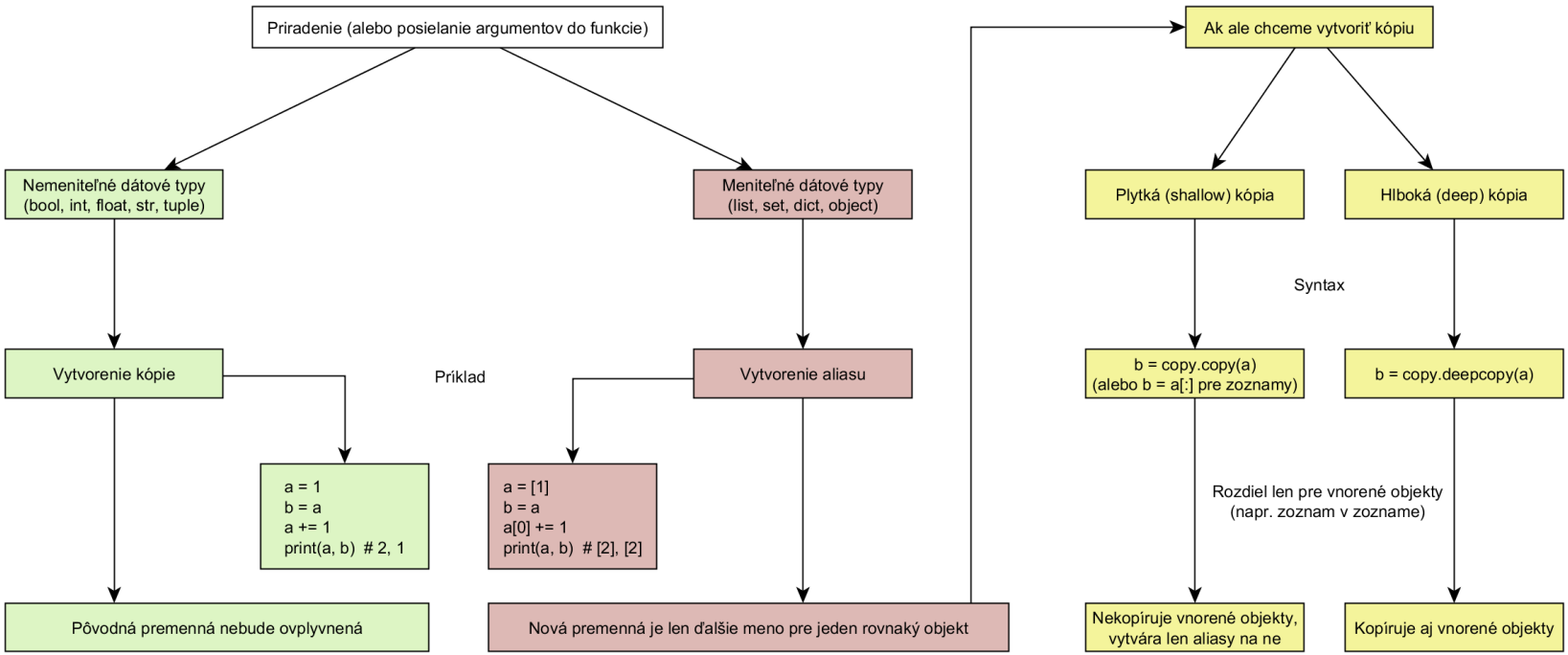
```
>>> badguys1[0] = "Scarecrow"              # Warning !!!
>>> print(badguys1)
['Scarecrow', 'Bane']
>>> print(badguys2)
['Scarecrow', 'Bane']
>>> badguys1 is badguys2
True
```

Kopírovanie zoznamov – plytká kópia

```
>>> badguys1 = ["Joker", "Bane"]
>>> badguys2 = badguys1[:]          # Shallow copy
>>> print(badguys1)
['Joker', 'Bane']
>>> print(badguys2)
['Joker', 'Bane']
>>> badguys1 is badguys2
False
```

```
>>> badguys1[0] = "Scarecrow"
>>> print(badguys1)
['Scarecrow', 'Bane']
>>> print(badguys2)
['Joker', 'Bane']
>>> badguys1 is badguys2
False
```

Priradenie v Pythone – zhrnutie



1. Reťazec (string)
2. Zoznam (list)
3. Priraďovanie v Pythone
4. Úlohy na zoznamy
5. Úlohy na reťazce

Úloha 1

a) Napíšte funkciu `find_max(lst)`, ktorá vezme zoznam čísel `lst` a vráti ich maximum.

b) Napíšte funkciu `find_min(lst)`, ktorá vezme zoznam čísel `lst` a vráti ich minimum.

```
>>> my_list = [23, 42, 11, -4, 5]
>>> find_max(my_list)
42
>>> find_min(my_list)
-4
```

- Poznámka: bez použitia vstavanej funkcie `max()` a `min()`

Úloha 2

- Napíšte funkciu `find_sum(lst)`, ktorá vezme zoznam čísel `lst` a vráti ich súčet.

```
>>> my_list = [23, 42, 11, -4, 5]
>>> find_sum(my_list)
77
```

- Poznámka: bez použitia vstavanej funkcie `sum()`

Úloha 3

- Napíšte funkciu `linear_search(value, lst)`, ktorá vezme hodnotu `value` a ľubovoľný zoznam `lst` a zistí, či sa daná hodnota `value` nachádza v zozname `lst`.

```
>>> my_list = [23, 42, 11, -4, 5]
>>> linear_search(11, my_list)
True
>>> linear_search(8, my_list)
False
```

- Poznámka: nájdite iné riešenie ako `return value in lst`

1. Reťazec (string)
2. Zoznam (list)
3. Priraďovanie v Pythone
4. Úlohy na zoznamy
5. Úlohy na reťazce

Úloha 4

- Napíšte funkciu `reverse_with_X(string)`, ktorá vezme reťazec `string` a vráti ho otočený prekladaný znakom “X”

```
>>> reverse_with_X("hello")
'oXlXlXeXh'
>>> reverse_with_X("The secret message")
'eXgXaXsXsXeXmX XtXeXrXcXeXsX XeXhXT'
```

- *Bonus*: rozšírte funkciu o dva parametre – znak výplne a počet znakov

```
>>> reverse_extended("hello", "|", 5)
'o|||||l|||||l|||||e|||||h'
```

Úloha 5

- Napíšte funkciu `count_a(string)`, ktorá vráti počet písmen A/a v danom reťazci `string`.

```
>>> count_a("Ahoj ahoj")
2
>>> count_a("Python")
0
```

- *Bonus*: rozšírte funkciu o parameter, ktorý predstavuje hľadaný znak

```
>>> count_char("hello", "l")
2
```

Úloha 6

- Napíšte funkciu `value(word)`, ktorá každému písmenu v slove `word` priradí jeho poradie v abecede (od 1 do 26) a vráti súčet týchto poradí. (Slovo preveďte na veľké písmena.)

```
>>> value("Hello")
52
>>> value("HELLO")
52
>>> value("ABC")
6
```

Úloha 7

- Napíšte funkciu `caesar(string, shift)`, ktorá zašifruje reťazec `string` Caesarovou šifrou posunom o `shift` celočíselných pozícií.

```
>>> caesar("hello", 1)
'IFMMP'
>>> caesar("Python", 3)
'SBWKRQ'
```

Úloha 8

- Napíšte funkciu `diagonal(string, num_lines)`, ktorá vypíše reťazec `string` šikmo do `num_lines` riadkov.

```
>>> diagonal("abcdefgh", 2)
a  c  e  g
 b  d  f  h
>>> diagonal("abcdefgh", 3)
a   d   g
 b   e   h
  c   f
>>> diagonal("abcdefgh", 4)
a    e
 b    f
  c    g
   d    h
```