C2184 Úvod do programování v Pythonu

5. Kolekce

Kolekce

Jsou objekty, které obsahují strukturovaně více hodnot.

Mezi základní typy kolekcí patří:

- ***n*-tice** (`tuple`) – soubor *n* hodnot s daným pořadím, (1, 5, 1)
- **seznam** (`list`) – upravovatelný soubor hodnot s daným pořadím, [1, 5, 1]
- **množina** (`set`) – upravovatelný soubor unikátních hodnot bez pořadí, {1, 5}
- **slovník** (`dict`) – upravovatelný soubor pojmenovaných hodnot, {'x': 1, 'y': 5}

Všechny kolekce jsou iterovatelné objekty.

Seznam (`list`)

- **Upravovatelný** soubor hodnot **s daným pořadím**
- Vytváříme je:
  - pomocí `[]` z jednotlivých prvků
  - pomocí funkce `list` z jiného iterovatelného objektu
- Seznam lze modifikovat (narozdíl od řetězce): měnit, přidávat, odebírat prvky
- Používáme, když máme více obdobných objektů, např. seznam čísel, seznam studentů
- Nejčastěji používaný typ kolekce

In [1]:
```
seznam = [1, 2, 3, 4, 5]
seznam
```

Out[1]:
```
[1, 2, 3, 4, 5]
```

In [2]:
```
seznam = []
seznam
```

Out[2]:
```
[]
```

In [3]:
```
pismena = list('hello')
pismena
```

Out[3]:
```
['h', 'e', 'l', 'l', 'o']
```

In [4]:
```
cisla = list(range(5))
cisla
```

Out[4]:
```
[0, 1, 2, 3, 4]
```

- Některé funkce vrací seznam

In [5]:
```
'Prasátko Peppa'.split()
```

Out[5]:
```
['Prasátko', 'Peppa']
```

- Na pořadí záleží

In [6]:
```
[1, 2, 3] == [1, 2, 3]
```

Out[6]:
```
True
```

In [7]:
```
[1, 2, 3] == [1, 3, 2]
```

Out[7]:
```
False
```

Indexování

- Jako u řetězců, počítáme zleva od 0, zprava od -1

In [8]:
```
seznam = [1, 2, 3, 4, 5]
seznam[2]
```

Out[8]:
```
3
```

In [9]:
```
seznam[-1]
```

Out[9]:
```
5
```

In [10]:
```
seznam[1:4]
```

Out[10]:
```
[2, 3, 4]
```

In [11]:
```
seznam[:-2]
```

Out[11]:
```
[1, 2, 3]
```

Operace se seznamy

Podobné jako u řetězců

- `len` – délka
- `in, not in` – přítomnost prvku
- `count` – počet výskytů prvku
- `index` – první výskyt prvku

In [12]:
```
cisla = [1, 2, 5, 2]
len(cisla)
```

Out[12]:
```
4
```

In [13]:
```
5 in cisla
```

Out[13]:
```
True
```

In [14]:
```
cisla.count(2)
```

Out[14]:
```
2
```

In [15]:
```
cisla.index(2)
```

Out[15]:
```
1
```

- Sřetězení

In [16]:
```
[1, 2] + [5, 2, 8]
```

Out[16]:
```
[1, 2, 5, 2, 8]
```

- Opakování

In [17]:
```
[1, 2] * 3
```

Out[17]:
```
[1, 2, 1, 2, 1, 2]
```

- Matematické operace `sum, min, max`

In [18]:
```
cisla = [1, 2, 5, 2]
sum(cisla)
```

Out[18]: 10

In [19]:
```
min(cisla)
```

Out[19]: 1

In [20]:
```
max(cisla)
```

Out[20]: 5

- (Tyto operace fungují na všech iterovatelných objektech, pokud to typově dává smysl.)

In [21]:
```
sum(range(5))
```

Out[21]: 10

In [22]:
```
min('hello')  # Minimum u řetězců = první v abecedě
```

Out[22]: 'e'

- Logické funkce `all` (všechny prvky pravdivé) a `any` (aspoň jeden prvek pravdivý)

In [23]:
```
all([True, True, True, False])
```

Out[23]:
```
False
```

In [24]:
```
any([True, True, True, False])
```

Out[24]:
```
True
```

Modifikace seznamů

In [25]:
```
seznam
```

Out[25]:
```
[1, 2, 3, 4, 5]
```

In [26]:
```
seznam[2] = 8
```

In [27]:
```
seznam
```

Out[27]:
```
[1, 2, 8, 4, 5]
```

Přidávání prvků

- Metoda `append(prvek)` přidá nový prvek na konec
- Metoda `insert(index, prvek)` přidá prvek na daný index (následující prvky se posouvají o 1 doprava)

In [28]:
```
seznam = [1, 2, 3]
seznam
```

Out[28]:
```
[1, 2, 3]
```

In [29]:
```
seznam.append(4)
```

In [30]:
```
seznam
```

Out[30]:
```
[1, 2, 3, 4]
```

In [31]:
```
seznam.insert(2, 10)
```

In [32]:
```
seznam
```

Out[32]:
```
[1, 2, 10, 3, 4]
```

- Metoda `extend(iterable)` přidá na konec seznamu všechny prvky z jiného iterovatelného objektu

In [33]:
```
seznam.extend([1, 2, 3])
```

In [34]:
```
seznam
```

Out[34]:
```
[1, 2, 10, 3, 4, 1, 2, 3]
```

Odebírání prvků

- Metoda `pop()` odebere poslední prvek a vrátí ho jako výsledek
- Metoda `pop(i)` odebere *i*-tý prvek a vrátí ho jako výsledek (následující prvky se posouvají o 1 doleva)

In [35]:
```
seznam
```

Out[35]:
```
[1, 2, 10, 3, 4, 1, 2, 3]
```

In [36]:
```
x = seznam.pop()  # Odstraň a vrať poslední prvek
print(seznam, x)
```

```
[1, 2, 10, 3, 4, 1, 2] 3
```

In [37]:
```
x = seznam.pop(2)  # Odstraň a vrať druhý prvek
print(seznam, x)
```

```
[1, 2, 3, 4, 1, 2] 10
```

- Metoda `remove(prvek)` odebere první výskyt prvku `prvek` (následující prvky se posouvají o 1 doleva)

In [38]:
```
seznam
```

Out[38]:
```
[1, 2, 3, 4, 1, 2]
```

In [39]:
```
seznam.remove(2)  # Odstraň prvek 2
```

In [40]:
```
seznam
```

Out[40]:
```
[1, 3, 4, 1, 2]
```

Změna směru

In [41]:
```
seznam
```

Out[41]:
```
[1, 3, 4, 1, 2]
```

In [42]:
```
seznam.reverse()
```

In [43]:
```
seznam
```

Out[43]:
```
[2, 1, 4, 3, 1]
```

Řazení

- Metoda `sort`

In [44]:
```
seznam = [1, 4, 2, -3, 5, 0]
seznam.sort()
seznam
```

Out[44]:
```
[-3, 0, 1, 2, 4, 5]
```

In [45]:
```
seznam.sort(reverse=True)
seznam
```

Out[45]:
```
[5, 4, 2, 1, 0, -3]
```

In [46]:
```
seznam = ['pes', 'kočka', 'slon']
seznam.sort()
seznam
```

Out[46]:
```
['kočka', 'pes', 'slon']
```

In [47]:
```
seznam.sort(key=len)
seznam
```

Out[47]:
```
['pes', 'slon', 'kočka']
```

n-tice (`tuple`)

- **Neupravovatelný** soubor *n* hodnot s daným pořadím
- Vytváříme je:
  - pomocí `( )` z jednotlivých prvků
  - pomocí funkce `tuple` z jiného iterovatelného objektu
- *n*-tici nelze modifikovat
- Používáme, když jednotlivé prvky mají svůj specifický význam a není nutné přidávat/odebírat prvky, např. adresa = (ulice, číslo, město)

In [48]:
```
adresa = ('Vlhká', 5, 'Brno')
adresa
```

Out[48]:
```
('Vlhká', 5, 'Brno')
```

In [1]:
```
souradnice = (1.0, 2.5)
souradnice
```

Out[1]:
```
(1.0, 2.5)
```

In [50]:
```
seznam
```

Out[50]:
```
['pes', 'slon', 'kočka']
```

In [51]:
```
ntice = tuple(seznam)
ntice
```

Out[51]:
```
('pes', 'slon', 'kočka')
```

Indexování *n*-tic

- Jako u řetězců a seznamů

In [52]:
```
adresa = ('Vlhká', 5, 'Brno')
adresa[0]
```

Out[52]:
```
'Vlhká'
```

In [53]:
```
adresa[1]
```

Out[53]:
```
5
```

In [54]:
```
adresa[2]
```

Out[54]:
```
'Brno'
```

In [55]:
```
adresa[:2]
```

Out[55]:
```
('Vlhká', 5)
```

- "Nultice" (*empty tuple*)

In [56]:
```
nultice = ()
nultice
```

Out[56]:
```
()
```

- "Jednice" (*singleton*)

In [57]:
```
jednice = (5,)
jednice
```

Out[57]:
```
(5,)
```

In [58]:
```
cislo = (5)
cislo
```

Out[58]:
```
5
```

- Pozor na rozdíl mezi "jednicí" a samotným prvkem

In [59]:
```
(5,) == 5
```

Out[59]:
```
False
```

Operace s *n*-ticemi

- Jako u seznamů

In [60]:
```
cisla = (1, 2, 3)
len(cisla)
```

Out[60]:
```
3
```

In [61]:
```
sum(cisla)
```

Out[61]:
```
6
```

In [62]:
```
cisla.count(8)
```

Out[62]:
```
0
```

In [63]:
```
cisla + cisla[::-1]
```

Out[63]:
```
(1, 2, 3, 3, 2, 1)
```

In [64]:
```
cisla * 3
```

Out[64]:
```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Množina (`set`)

- Upravovatelný soubor **unikátních** hodnot **bez daného pořadí**
- Vytváříme je:
  - pomocí `{}` z jednotlivých prvků
  - pomocí funkce `set` z jiného iterovatelného objektu
- Množinu lze modifikovat
- Každý prvek obsažen nejvíc jednou
- Nelze indexovat

In [65]:
```
mnozina = {1, 2, 3}
mnozina
```

Out[65]:
```
{1, 2, 3}
```

In [66]:
```
mnozina = set('hello')
mnozina
```

Out[66]:
```
{'e', 'h', 'l', 'o'}
```

- Každý prvek je obsažen nejvíc jednou

In [67]:
```
seznam = [1, 1, 2, 3, 3]
set(seznam)
```

Out[67]:
```
{1, 2, 3}
```

- Pořadí není dané

In [68]:
```
{1, 2, 3} == {3, 2, 1}
```

Out[68]:
```
True
```

- Nelze indexovat

In [69]:
```
mnozina = {1, 2, 3}
mnozina[0]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-69-a0d66b3a8e7f> in <module>()
      1 mnozina = {1, 2, 3}
----> 2 mnozina[0]

TypeError: 'set' object does not support indexing
```

- Prázdná množina se zapisuje `set()`, protože `{}` je rezervováno pro prázdný slovník

In [70]:
```
type(set())
```

Out[70]:
```
set
```

In [71]:
```
type({})
```

Out[71]:
```
dict
```

Množinové operace

- `len` – počet prvků
- `in, not in` – přítomnost prvku (efektivnější než u seznamu, nemusí se kontrolovat všechny prvky)

In [72]:
```
A = {1, 2, 3}
```

In [73]:
```
len(A)
```

Out[73]: 3

In [74]:
```
2 in A
```

Out[74]: True

In [75]:
```
5 in A
```

Out[75]: False

In [76]:
```
A = {1, 2, 3}
B = {2, 4, 6}
```

In [77]:
```
A & B  # Průnik A ∩ B
```

Out[77]:
```
{2}
```

In [78]:
```
A | B  # Sjednocení A ∪ B
```

Out[78]:
```
{1, 2, 3, 4, 6}
```

In [79]:
```
A - B  # Rozdíl množin A - B (prvky A kromě prvků B)
```

Out[79]:
```
{1, 3}
```

In [80]:
```
A <= B  # A je podmnožinou B (A ⊆ B)
```

Out[80]:
```
False
```

In [81]:
```
A < B  # A je vlastní podmnožinou B (A ⊂ B)
```

Out[81]:
```
False
```

Přidávání prvků

- Metoda add

In [82]:
```
A = set()
A.add(5)
A.add(6)
A
```

Out[82]: {5, 6}

In [83]:
```
A.add(5)
A
```

Out[83]: {5, 6}

Odebírání prvků

- Metoda `discard(x)` odebere prvek x (neudělá nic, pokud tam x není)
- Metoda `remove(x)` odebere prvek x (skončí chybou, pokud tam x není)

In [84]:
```
A = set(range(5))
A
```

Out[84]:
```
{0, 1, 2, 3, 4}
```

In [85]:
```
A.discard(2)
```

In [86]:
```
A
```

Out[86]:
```
{0, 1, 3, 4}
```

- Metoda `pop()` odebere jeden libovolný prvek a vrátí ho jako výsledek

In [87]:
```
A
```

Out[87]:
```
{0, 1, 3, 4}
```

In [88]:
```
x = A.pop()
x
```

Out[88]:
```
0
```

In [89]:
```
A
```

Out[89]:
```
{1, 3, 4}
```

- Metoda `clear()` odebere všechny prvky

In [90]:
```
A.clear()
```

In [91]:
```
A
```

Out[91]:
```
set()
```

Slovník (`dict`, *dictionary*)

- Upravovatelný soubor dvojic **klíč:hodnota**, **bez daného pořadí**
- Vytváříme je:
  - pomocí `{}` z jednotlivých dvojic klíč:hodnota
  - pomocí funkce `dict` z jiného iterovatelného objektu
- Slovník lze modifikovat
- Každý klíč obsažen nejvíc jednou
- Na pořadí nezáleží
- Indexujeme podle klíčů (ne podle pořadí)

In [92]:
```
slovnik = {'klic1': 'hodnota1', 'klic2': 'hodnota2'}
slovnik
```

Out[92]:
```
{'klic1': 'hodnota1', 'klic2': 'hodnota2'}
```

In [93]:
```
slovnik = {}  # Prázdný slovník
slovnik
```

Out[93]:
```
{}
```

In [1]:
```
seznam_dvojic = [('jack', 4098), ('sape', 4139), ('guido', 4127)]
slovnik = dict(seznam_dvojic)
slovnik
```

Out[1]:
```
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

Indexování slovníku

- Pomocí klíče

In [2]:
```
slovnik
```

Out[2]:
```
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

In [3]:
```
slovnik['jack']
```

Out[3]:
```
4098
```

In [4]:
```
slovnik['bob']
```

```
---------------------------------------------------------
KeyError                Traceback (most recent call last)
<ipython-input-4-9505a0e8ff92> in <module>()
----> 1 slovnik['bob']

KeyError: 'bob'
```

- Metoda `get` je jako `[ ]`, ale řeší i chybějící klíče

In [5]:
```
print(slovnik.get('guido'))
```

```
4127
```

In [6]:
```
print(slovnik.get('bob'))
```

```
None
```

In [7]:
```
print(slovnik.get('bob', 'defaultni_hodnota'))
```

```
defaultni_hodnota
```

Testování přítomnosti klíče

- U slovníku se pomocí `in` dotazujeme na klíče (ne na hodnoty)

In [8]:
```
slovnik
```

Out[8]:
```
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

In [9]:
```
'jack' in slovnik
```

Out[9]:
```
True
```

In [10]:
```
4127 in slovnik
```

Out[10]:
```
False
```

In [11]:
```
4127 in slovnik.values()
```

Out[11]:
```
True
```

Přidání nebo úprava stávajících hodnot

- Pokud klíč není ve slovníku, přidá se a přiřadí se mu hodnota
- Pokud klíč je ve slovníku, stará hodnota se zahodí a přiřadí se nová

In [12]:
```
slovnik
```

Out[12]:
```
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

In [13]:
```
slovnik['bob'] = 1234
slovnik
```

Out[13]:
```
{'bob': 1234, 'guido': 4127, 'jack': 4098, 'sape': 4139}
```

In [14]:
```
slovnik['guido'] = 666
slovnik
```

Out[14]:
```
{'bob': 1234, 'guido': 666, 'jack': 4098, 'sape': 4139}
```

- Metoda `update` přijímá jako parametr další slovník, kterým upraví stávající

In [15]:
```
slovnik
```

Out[15]:
```
{'bob': 1234, 'guido': 666, 'jack': 4098, 'sape': 4139}
```

In [16]:
```
slovnik.update({'guido': 1111, 'alice': 9876})
slovnik
```

Out[16]:
```
{'alice': 9876, 'bob': 1234, 'guido': 1111, 'jack': 4098, 'sape': 4139}
```

Odebírání hodnot

- Pomocí metody `pop` jako u seznamů

In [17]:
```
slovnik
```

Out[17]:
```
{'alice': 9876, 'bob': 1234, 'guido': 1111, 'jack': 4098, 'sape': 4139}
```

In [18]:
```
x = slovnik.pop('jack')
print(x, slovnik)
```

```
4098 {'sape': 4139, 'guido': 1111, 'bob': 1234, 'alice': 9876}
```

- Metoda `clear` vyprázdní celý slovník

In [19]:
```
slovnik.clear()
slovnik
```

Out[19]:
```
{}
```

Homogenní a heterogenní kolekce, kolekce v kolekci

- *Homogenní kolekce* – hodnoty stejného typu, `[1, 2, 3]`
- *Heterogenní kolekce* – hodnoty smíšeného typu, `[1, 'ahoj', True]`
- Kolekce mohou v sobě obsahovat další kolekce
  - Výjimka: Klíče v slovníku a prvky množiny mohou být pouze nemodifikovatelné objekty.

In [112]:
```
[1, '2', 3.0, [4, 5], (6, 7)]
```

Out[112]:
```
[1, '2', 3.0, [4, 5], (6, 7)]
```

In [113]:
```
{'a': [(1,),(1,2)], 'b': [(2,3),(1,)], 'c': [(9,),(1,)], 'd':[]}
```

Out[113]:
```
{'a': [(1,), (1, 2)], 'b': [(2, 3), (1,)], 'c': [(9,), (1,)], 'd': []}
```

Shrnutí

- Seznam, `list, [,,,]`
  - Více obdobných objektů v daném pořadí
  - Chceme je přidávat/měnit/odebírat
- *n*-tice, `tuple, (,,,)`
  - Více hodnot se specifickým významem
  - Nechceme měnit
- Množina, `set, {,,,}`
  - Více obdobných objektů, na pořadí nám nezáleží
  - Chceme efektivně přidávat/odebírat/testovat přítomnost prvků
- Slovník, `dict, {:,:,:,:}`
  - Chceme si hodnoty asociovat s nějakým klíčem
  - Chceme efektivně získat hodnotu pro daný klíč

Další typy kolekcí

Další specializované typy kolekcí nabízí modul `collections`:

- `namedtuple`
- `defaultdict`
- `OrderedDict`
- `Counter`
- `frozenset`
- `deque`
- …

Volba typu kolekce

Jaké typy kolekcí byste použili v těchto případech? Jaké budou typy hodnot (klíčů)?

- Každý den měříme teplotu vzduchu, chceme uložit naměřené hodnoty za celý měsíc.
- Měříme teplotu vzduchu 25.9. v Brně, Praze, Plzni a Ostravě.
- Ve třídě je několik studentů, u každého si chceme pamatovat jméno, příjmení a UČO.
- Vedeme si seznam zemí, které jsme navštívili.
- Čteme knihu a počítáme, kolikrát byla zmíněna každá z postav.
- Fakulta má několik ústavů, u každého si potřebujeme pamatovat jména zaměstnanců.

- Každý den měříme teplotu vzduchu, chceme uložit naměřené hodnoty za celý měsíc.
  - Seznam reálných čísel
- Měříme teplotu vzduchu 25.9. v Brně, Praze, Plzni a Ostravě.
  - Slovník (klíče `str`, hodnoty `float`)
- Ve třídě je několik studentů, u každého si chceme pamatovat jméno, příjmení a UČO.
  - Seznam *n*-tic
- Vedeme si seznam zemí, které jsme navštívili.
  - Množina řetězců
- Čteme knihu a počítáme, kolikrát byla zmíněna každá z postav.
  - Slovník (klíče `str`, hodnoty `int`)
- Fakulta má několik ústavů, u každého si potřebujeme pamatovat jména zaměstnanců.
  - Slovník (klíče `str`, hodnoty seznamy řetězců)

Rozbalování (*unpacking*)

- Pomocí operátoru *
- Vytáhneme všechny prvky z iterovatelného objektu

In [114]:
```
seznam = [1, 2, 3, 4]
print(seznam)
```

```
[1, 2, 3, 4]
```

In [115]:
```
print(*seznam)  # Stejné jako print(1, 2, 3, 4)
```

```
1 2 3 4
```

In [116]:
```
[0, seznam, 5, 6]
```

Out[116]:
```
[0, [1, 2, 3, 4], 5, 6]
```

In [117]:
```
[0, *seznam, 5, 6]  # Stejné jako [0, 1, 2, 3, 4, 5, 6]
```

Out[117]:
```
[0, 1, 2, 3, 4, 5, 6]
```

- Pomocí více proměnných na levé straně přiřazení

In [118]:
```
seznam
```

Out[118]:
```
[1, 2, 3, 4]
```

In [119]:
```
a, b, c, d = seznam
a
```

Out[119]:
```
1
```

In [120]:
```
b
```

Out[120]:
```
2
```

In [121]:
```
c
```

Out[121]:
```
3
```

In [122]:
```
d
```

Out[122]:
```
4
```

In [123]:
```
prvni, druhy, *zbytek, posledni = range(10)
prvni
```

Out[123]: 0

In [124]:
```
druhy
```

Out[124]: 1

In [125]:
```
zbytek
```

Out[125]: [2, 3, 4, 5, 6, 7, 8]

In [126]:
```
posledni
```

Out[126]: 9

Procházení kolekcí

- Pomocí cyklu `for`

In [127]:
```
for prvek in {1, 2, 3, 2}:
    print(prvek)
```

```
1
2
3
```

Procházení slovníků

- Přes klíče

In [128]:
```
slovnik = {'guido': 4127, 'jack': 4098}
```

In [129]:
```
for jmeno in slovnik:
    print(jmeno)
```

```
guido
jack
```

- Přes hodnoty

In [130]:
```
for telefon in slovnik.values():
    print(telefon)
```

```
4127
4098
```

- Přes klíče a hodnoty

In [131]:
```
for jmeno, telefon in slovnik.items():
    print(f'{jmeno}: {telefon}')
```

```
guido: 4127
jack: 4098
```

Chytré procházení kolekcí

- Funkce `enumerate` očísluje prvky

In [132]:
```
jmena = ['Bob', 'Alice', 'Cyril']
for i, jmeno in enumerate(jmena):
    print(f'{i}. {jmeno}')
```

```
0. Bob
1. Alice
2. Cyril
```

In [133]:
```
for i, jmeno in enumerate(jmena, 1):
    print(f'{i}. {jmeno}')
```

```
1. Bob
2. Alice
3. Cyril
```

- Funkce `reversed` prochází od konce

In [134]:
```
for jmeno in reversed(jmena):
    print(jmeno)
```

```
Cyril
Alice
Bob
```

- Funkce `sorted` prochází vytváří nový seřazený seznam

In [135]:
```
for jmeno in sorted(jmena):
    print(jmeno)
```

```
Alice
Bob
Cyril
```

- Funkce `zip` prochází více objektů najednou

In [136]:
```
for jmeno, pismenko in zip(jmena, 'XYZW'):
    print(jmeno, pismenko)
```

```
Bob X
Alice Y
Cyril Z
```

- Pozor, funkce `enumerate`, `reversed`, `zip` nevytváří kolekci, pouze *iterátor* určen k jednorázovému projdění prvků

In [137]:
```
iterator = reversed(jmena)
for jmeno in iterator:
    print(jmeno)
```

```
Cyril
Alice
Bob
```

In [138]:
```
for jmeno in iterator:
    print(jmeno)
```

Generátorové výrazy (*generator expression*)

- Pomocí `...for...in...` můžeme vytvářet a filtrovat kolekce

In [139]:
```
puvodni = [1, 2, 3, 4, 5, 6]
```

In [140]:
```
[str(x) for x in puvodni]
```

Out[140]:
```
['1', '2', '3', '4', '5', '6']
```

In [141]:
```
{i: i**2 for i in puvodni}
```

Out[141]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

- Filtrujeme přidáním `if`

In [142]:
```
{x for x in puvodni if x % 2 == 0}
```

Out[142]:
```
{2, 4, 6}
```

- Typ výsledné kolekce je dán typem závorek: `[ ]` seznam, `{ }` množina nebo slovník
  - Výjimka: `( )` vytváří iterátor, nikoliv *n*-tici

In [143]:
```
(x for x in puvodni if x >= 3)  # iterátor
```

Out[143]:
```
<generator object <genexpr> at 0x7f887c3b09e8>
```

In [144]:
```
tuple(x for x in puvodni if x >= 3)  # n-tice
```

Out[144]:
```
(3, 4, 5, 6)
```

Funkce `join`

- Spojuje prvky pomocí separatorů, funguje ale pouze na prvky typu `str`

In [145]:
```
iterable = ['1', '2', '3']
' - '.join(iterable)
```

Out[145]:
```
'1 - 2 - 3'
```

Vytvoření nového objektu vs modifikace objektu

- Vytvoření nového objektu:

In [146]:
```
puvodni = 'Spam spam spam.'
novy = puvodni.replace('spam', 'ham')
print(puvodni)
print(novy)
```

```
Spam spam spam.
Spam ham ham.
```

In [147]:
```
puvodni = [1, 8, 5, 2]
novy = sorted(puvodni)
print(puvodni)
print(novy)
```

```
[1, 8, 5, 2]
[1, 2, 5, 8]
```

- Modifikace objektu:

In [148]:
```
puvodni = [1, 8, 5, 2]
novy = puvodni.sort()
print(puvodni)
print(novy)
```

```
[1, 2, 5, 8]
None
```

In [149]:
```
puvodni = [1, 8, 5, 2]
novy = puvodni.append(0)
print(puvodni)
print(novy)
```

```
[1, 8, 5, 2, 0]
None
```

Stejné objekty vs ten samý objekt

- Operátory `==, !=` testují, jestli jsou dva objekty stejné
- Operátory `is, is not` testují, jestli se jedná o ten samý objekt
- Dva stejné objekty:

In [150]:
```
a = [1, 2, 3]
b = [1, 2, 3]
a == b
```

Out[150]:
```
True
```

In [151]:
```
a is b
```

Out[151]:
```
False
```

In [152]:
```
a.append(4)
print(a, b)
```

```
[1, 2, 3, 4] [1, 2, 3]
```

- Ten samý objekt:

In [153]:
```
a = [1, 2, 3]
b = a
a == b
```

Out[153]:
```
True
```

In [154]:
```
a is b
```

Out[154]:
```
True
```

In [155]:
```
a.append(4)
print(a, b)
```

```
[1, 2, 3, 4] [1, 2, 3, 4]
```

- Všechny modifikovatelné kolekce můžeme duplikovat pomocí metody `copy` (vytváříme tak nový objekt)

In [156]:
```
a = [6, 8, 3, 1]
b = a.copy()
b.sort()
print(a, b)
```

```
[6, 8, 3, 1] [1, 3, 6, 8]
```