05_Kolekce

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

[1]:
```
numbers = [1, 2, 3, 4, 5]
numbers
```

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

[2]:
```
numbers = []
numbers
```

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

[3]:
```
letters = list('hello')
letters
```

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

[4]:
```
numbers = list(range(5))
numbers
```

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

- Některé funkce vrací seznam

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

[5]:
```
['Prasátko', 'Peppa']
```

- Na pořadí záleží

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

[6]:
```
True
```

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

[7]:
```
False
```

Indexování

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

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

[8]:
```
3
```

[9]:
```
numbers[-1]
```

[9]:
```
5
```

[10]:
```
numbers[1:4]
```

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

[11]:
```
numbers[:-2]
```

[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

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

[12]:
```
4
```

[13]:
```
5 in numbers
```

[13]:
```
True
```

[14]:
```
numbers.count(2)
```

[14]:
```
2
```

[15]:
```
numbers.index(2)
```

[15]:
```
1
```

- Sřetězení

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

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

- Opakování

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

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

- Matematické operace `sum`, `min`, `max`

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

[18]: 10

[19]:
```
min(numbers)
```

[19]: 1

[20]:
```
max(numbers)
```

[20]: 5

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

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

[21]: 10

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

[22]: 'e'

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

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

[23]: False

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

[24]: True

Modifikace seznamů

[25]:
```
numbers = [1, 2, 3, 4, 5]
numbers
```

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

[26]:
```
numbers[2] = 8
```

[27]:
```
numbers
```

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

Přidávání prvků

- Metoda `.append(item)` přidá nový prvek `item` na konec seznamu
- Metoda `.insert(i, item)` přidá prvek `item` na pozici `i` (následující prvky se posouvají o 1 doprava)

[28]:
```
numbers = [1, 2, 3]
numbers
```

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

[29]:
```
numbers.append(4)
```

[30]:
```
numbers
```

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

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

[32]:
```
numbers
```

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

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

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

[34]:
```
numbers
```

[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)

[35]:
```
x = numbers.pop()
print(x)
print(numbers)
```

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

[36]:
```
x = numbers.pop(2)
print(x)
print(numbers)
```

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

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

[37]:
```
numbers = [1, 2, 3, 4, 5]
numbers.remove(2)
```

[38]:
```
numbers
```

[38]:
```
[1, 3, 4, 5]
```

- Metoda `.clear()` odstraní všechny prvky

[39]:
```
numbers.clear()
```

[40]:
```
numbers
```

[40]:
```
[]
```

Změna směru

- Metoda `.reverse()`

[41]:
```
numbers = [1, 2, 3, 0, 0]
```

[42]:
```
numbers.reverse()
```

[43]:
```
numbers
```

[43]:
```
[0, 0, 3, 2, 1]
```

Řazení

- Metoda `.sort()`

[44]:
```
numbers.sort()
```

[45]:
```
numbers
```

[45]:
```
[0, 0, 1, 2, 3]
```

[46]:
```
numbers.sort(reverse=True)
```

[47]:
```
numbers
```

[47]:
```
[3, 2, 1, 0, 0]
```

[48]:
```
animals = ['pes', 'kočka', 'slon']
animals.sort()
animals
```

[48]:
```
['kočka', 'pes', 'slon']
```

[49]:
```
animals.sort(key=len)
animals
```

[49]:
```
['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. `address = (street, number, city)`

[50]:
```
address = ('Vlhká', 5, 'Brno')
address
```

[50]:
```
('Vlhká', 5, 'Brno')
```

[51]:
```
coordinates = (1.0, 2.5)
coordinates
```

[51]:
```
(1.0, 2.5)
```

[52]:
```
animals
```

[52]:
```
['pes', 'slon', 'kočka']
```

[53]:
```
animals_tuple = tuple(animals)
animals_tuple
```

[53]:
```
('pes', 'slon', 'kočka')
```

Indexování n-tic

- Jako u řetězců a seznamů

[54]:
```
address[0]
```

[54]:
```
'Vlhká'
```

[55]:
```
address[1]
```

[55]:
```
5
```

[56]:
```
address[2]
```

[56]:
```
'Brno'
```

[57]:
```
address[:2]
```

[57]:
```
('Vlhká', 5)
```

"Nultice" (*empty tuple*)

[58]:
```
empty = ()
empty
```

[58]:
```
()
```

"Jednice" (*singleton*)

[59]:
```
singleton = (5,)  # Zápis n-tice s jedním prvkem
singleton
```

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

[60]:
```
number = (5)  # Pouze zápis čísla v závorce
number
```

[60]:
```
5
```

- Pozor na rozdíl mezi "jednicí" a samotným prvkem

[61]:
```
(5,) == 5
```

[61]:
```
False
```

Operace s n-ticemi

- Jako u seznamů

[62]:
```
address
```

[62]:
```
('Vlhká', 5, 'Brno')
```

[63]:
```
len(address)
```

[63]:
```
3
```

[64]:
```
address.count(8)
```

[64]:
```
0
```

[65]:
```
address + address[::-1]
```

[65]:
```
('Vlhká', 5, 'Brno', 'Brno', 5, 'Vlhká')
```

[66]:
```
address * 3
```

[66]:
```
('Vlhká', 5, 'Brno', 'Vlhká', 5, 'Brno', 'Vlhká', 5, 'Brno')
```

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
- Nemá dané pořadí, tudíž nelze ji indexovat

[67]:
```
colors = {'red', 'yellow', 'brown'}
colors
```

[67]:
```
{'brown', 'red', 'yellow'}
```

[68]:
```
letters = set('hello')
letters
```

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

- Každý prvek je obsažen nejvíc jednou

```
[69]: {1, 1, 2, 2, 2, 3}
```

```
[69]: {1, 2, 3}
```

- Pořadí není dané

```
[70]: {1, 2, 3} == {3, 2, 1}
```

```
[70]: True
```

- Nelze indexovat

```
[71]: numbers = {1, 2, 3}
      numbers[0]
```

```
␣
↪---------------------------------------------------------------------------

      TypeError                                 Traceback (most␣
↪recent call last)

      <ipython-input-71-40b3d4d6b413> in <module>
        1 numbers = {1, 2, 3}
  ----> 2 numbers[0]


      TypeError: 'set' object is not subscriptable
```

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

```
[72]: type(set())
```

```
[72]: set
```

```
[73]: type({})
```

```
[73]: dict
```

Množinové operace

- `len()` – počet prvků

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

[74]:
```
A = {1, 2, 3}
```

[75]:
```
len(A)
```

[75]: 3

[76]:
```
2 in A
```

[76]: True

[77]:
```
5 in A
```

[77]: False

- `&`, `set.intersection()` – průnik $A \cap B$
- `|`, `set.union()` – sjednocení $A \cup B$
- `-`, `set.difference()` – množinový rozdíl $A \setminus B$
- `^`, `set.symmetric_difference()` – symetrický rozdíl $A \triangle B$
- `<=` – inkluze $A \subseteq B$
- `<` – ostrá inkluze $A \subset B$

[78]:
```
A = {1, 2, 3}
B = {2, 4, 6}
```

[79]:
```
A & B  # Průnik A ∩ B
```

[79]: {2}

[80]:
```
A | B  # Sjednocení A ∪ B
```

[80]: {1, 2, 3, 4, 6}

[81]:
```
A - B  # Rozdíl množin A - B (prvky A kromě prvků B)
```

[81]: {1, 3}

[82]:
```
A ^ B  # Symetrický rozdíl A Δ B (prvky A kromě prvků B plus prvky B␣
↪kromě prvků A)
```

[82]: {1, 3, 4, 6}

[83]:
```
A <= B  # A je podmnožinou B (A ⊆ B, tj. B obsahuje všechno z A)
```

[83]:
```
False
```

[84]:
```
A < B  # A je vlastní podmnožinou B (A ⊂ B, tj. B obsahuje všechno z
↪A a ještě něco navíc)
```

[84]:
```
False
```

Přidávání prvků

- Metoda `.add(x)` přidá prvek x

[85]:
```
A = set()
A.add(5)
A.add(6)
A
```

[85]:
```
{5, 6}
```

[86]:
```
A.add(5)
A
```

[86]:
```
{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)

[87]:
```
A = set(range(5))
A
```

[87]:
```
{0, 1, 2, 3, 4}
```

[88]:
```
A.discard(2)
```

[89]:
```
A
```

[89]:
```
{0, 1, 3, 4}
```

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

[90]:
```
A
```

[90]:
```
{0, 1, 3, 4}
```

[91]:
```
x = A.pop()
print(x)
print(A)
```

```
0
{1, 3, 4}
```

- Metoda `.clear()` odebere všechny prvky

[92]:
```
A.clear()
```

[93]:
```
A
```

[93]:
```
set()
```

Slovník (`dict`, *dictionary*)

- Upravovatelný soubor dvojic **klíč:hodnota**
- 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)

[94]:
```
en2cz = {'apple': 'jablko', 'carrot': 'mrkev'}
en2cz
```

[94]:
```
{'apple': 'jablko', 'carrot': 'mrkev'}
```

[95]:
```
my_vocabulary = {}  # Prázdný slovník
my_vocabulary
```

[95]:
```
{}
```

[96]:
```
tuples = [('jack', 4098), ('sape', 4139), ('guido', 4127)]
phonebook = dict(tuples)
phonebook
```

[96]:
```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```

Indexování slovníku

- Pomocí klíče

[97]:
```
phonebook
```

[97]:
```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```

[98]:
```
phonebook['jack']
```

[98]:
```
4098
```

[99]:
```
phonebook['bob']
```

```
␣
↪---------------------------------------------------------------------------␣

      KeyError                                  Traceback (most␣
↪recent call last)

      <ipython-input-99-90de1601f191> in <module>
  ----> 1 phonebook['bob']


      KeyError: 'bob'
```

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

[100]:
```
print(phonebook.get('guido'))
```

```
4127
```

[101]:
```
print(phonebook.get('bob'))
```

```
None
```

[102]:
```
print(phonebook.get('bob', 'number_not_found'))
```

```
number_not_found
```

Testování přítomnosti klíče

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

[103]:
```
'jack' in phonebook
```

[103]:
```
True
```

[104]:
```
4127 in phonebook
```

[104]:
```
False
```

[105]:
```
4127 in phonebook.values()
```

[105]:
```
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

[106]:
```
phonebook
```

[106]:
```
{'jack': 4098, 'sape': 4139, 'guido': 4127}
```

[107]:
```
phonebook['bob'] = 1234
phonebook
```

[107]:
```
{'jack': 4098, 'sape': 4139, 'guido': 4127, 'bob': 1234}
```

[108]:
```
phonebook['guido'] = 666
phonebook
```

[108]:
```
{'jack': 4098, 'sape': 4139, 'guido': 666, 'bob': 1234}
```

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

[109]:
```
phonebook.update({'guido': 1111, 'alice': 9876})
phonebook
```

[109]:
```
{'jack': 4098, 'sape': 4139, 'guido': 1111, 'bob': 1234, 'alice':␣
↪9876}
```

Odebírání hodnot

- Pomocí metody `.pop()` jako u seznamů

[110]:
```
jacks_phone = phonebook.pop('jack')
print(jacks_phone)
print(phonebook)
```

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

- Metoda `.clear()` vyprázdní celý slovník

[111]:
```
phonebook.clear()
phonebook
```

[111]:
```
{}
```

Pořadí v slovníku

- Nezáleží na pořadí

[112]:
```
{1: 10, 2: 20} == {2: 20, 1: 10}
```

[112]:
```
True
```

- Slovník si pamatuje, v jakém pořadí byly klíče přidány (platí pouze od Pythonu 3.6)

[113]:
```
print({1: 10, 2: 20})
print({2: 20, 1: 10})
```

```
{1: 10, 2: 20}
{2: 20, 1: 10}
```

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.

[114]:
```
[1, '2', 3.0, [4, 5], (6, 7)]
```

[114]:
```
[1, '2', 3.0, [4, 5], (6, 7)]
```

[115]:
```
{'a': [(1,),(1,2)], 'b': [(2,3),(1,)], 'c': [(9,),(1,)], 'd':[]}
```

[115]:
```
{'a': [(1,), (1, 2)], 'b': [(2, 3), (1,)], 'c': [(9,), (1,)], 'd': []}
```

Shrnutí

	n-tice	Seznam	Množina	Slovník
Typ	`tuple`	`list`	`set`	`dict`
Prázdná	`()`	`[]`	`set()`	`{}`
S prvky	`(1, 2, 3)`	`[1, 2, 3]`	`{1, 2, 3}`	`{1: 10, 2: 20}`

	n-tice	Seznam	Množina	Slovník
Pořadí	✔	✔	✘	✘nezáleží/✔drží
Duplikáty	✔	✔	✘	✘klíč/✔hodnota
Modifikace	✘	✔	✔	✔
Využití	spojení hodnot se specifickým významem	obdobné objekty v pevném pořadí	obdobné objekty bez pořadí	asociace klíčů s hodnotami
Efektivní operace	`[]`	`[]`, `append`, `pop`	`in`, `add`, `discard`, `pop`	`[]`, `in`, `pop`

Další typy kolekcí

Další specializované typy kolekcí nabízí modul `collections`:

- `namedtuple`
  - jako `tuple`, jednotlivé prvky lze pojmenovat (`address.street` místo `address[0]`)
- `defaultdict`
  - jako `dict`, chybějící hodnotu umí doplnit podle zadané factory funkce
- `Counter`
  - jako `dict`, vhodný k počítání různých typů něčeho
- `frozenset`
  - jako `set`, nemodifikovatelný
- `deque`
  - jako `list`, umí efektivně pridávat/mazat z obou stran
- …

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.

Rozbalování (*unpacking*)

- Pomocí operátoru *
- Vytáhneme všechny prvky z iterovatelného objektu

[116]:
```
numbers = [1, 2, 3, 4]
print(numbers)
```

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

[117]:
```
print(*numbers)  # Stejné jako print(1, 2, 3, 4)
```

```
1 2 3 4
```

[118]:
```
[0, numbers, 5, 6]
```

[118]:
```
[0, [1, 2, 3, 4], 5, 6]
```

[119]:
```
[0, *numbers, 5, 6]  # Stejné jako [0, 1, 2, 3, 4, 5, 6]
```

[119]:
```
[0, 1, 2, 3, 4, 5, 6]
```

- Pomocí více proměnných na levé straně přiřazení

[120]:
```
a, b, c, d = numbers
print('a:', a)
print('b:', b)
print('c:', c)
print('d:', d)
```

```
a: 1
b: 2
c: 3
d: 4
```

[121]:
```
name, middle_name, surname = 'Karel Hynek Mácha'.split()
print('name:', name)
print('middle_name:', middle_name)
print('surname:', surname)
```

```
name: Karel
middle_name: Hynek
surname: Mácha
```

[122]:
```
first, second, *rest, last = range(10)
print('first:', first)
print('second:', second)
print('rest:', rest)
print('last:', last)
```

```
first: 0
second: 1
rest: [2, 3, 4, 5, 6, 7, 8]
last: 9
```

Procházení kolekcí

- Pomocí cyklu `for`

[123]:
```
for prvek in {1, 2, 3, 2}:
    print(prvek)
```

```
1
2
3
```

Procházení slovníků

- Přes klíče

[124]:
```
phonebook = {'guido': 4127, 'jack': 4098}
```

[125]:
```
for name in phonebook:
    print(name)
```

```
guido
jack
```

[126]:
```
for name in phonebook.keys():
    print(name)
```

```
guido
jack
```

- Přes hodnoty

[127]:
```
for phone in phonebook.values():
    print(phone)
```

```
4127
4098
```

- Přes klíče a hodnoty

[128]:
```
for name, phone in phonebook.items():
    print(f'{name}: {phone}')
```

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

Chytré procházení kolekcí

- Funkce `enumerate` očísluje prvky

[129]:
```
names = ['Bob', 'Alice', 'Cyril']
for i, name in enumerate(names):
    print(f'{i}. {name}')
```

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

[130]:
```
for i, name in enumerate(names, 1):
    print(f'{i}. {name}')
```

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

- Funkce `reversed` prochází od konce

[131]:
```
for name in reversed(names):
    print(name)
```

```
Cyril
Alice
Bob
```

- Funkce `sorted` prochází vytváří nový seřazený seznam

[132]:
```
for name in sorted(names):
    print(name)
```

```
Alice
Bob
Cyril
```

- Funkce `zip` prochází více objektů najednou

[133]:
```
for name, letter in zip(names, 'XYZW'):
    print(name, letter)
```

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

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

[134]:
```
iterator = reversed(names)
for name in iterator:
    print(name)
```

```
Cyril
Alice
Bob
```

[135]:
```
for names in iterator:
    print(names)
```

Generátorové výrazy (*generator expression*)

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

[136]:
```
numbers = [1, 2, 3, 4, 5, 6]
```

[137]:
```
[str(x) for x in numbers]
```

[137]:
```
['1', '2', '3', '4', '5', '6']
```

[138]:
```
{i: i**2 for i in numbers}
```

[138]:
```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

- Filtrujeme přidáním `if`

[139]:
```
{x for x in numbers if x % 2 == 0}
```

[139]:
```
{2, 4, 6}
```

- Typ výsledné kolekce je dán typem závorek:
  - `[… for … in …]` – *list comprehension*
  - `{… for … in …}` – *set comprehension*
  - `{…: … for … in …}` – *dictionary comprehension*
  - Výjimka: `(… for … in …)` vytváří iterátor, nikoliv *n*-tici

[140]:
```
(x for x in numbers if x >= 3)  # iterátor
```

[140]:
```
<generator object <genexpr> at 0x7f63d04fbb30>
```

[141]:
```
tuple(x for x in numbers if x >= 3)  # n-tice
```

[141]:
```
(3, 4, 5, 6)
```

Metoda `join`

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

[142]:
```
' - '.join(['1', '2', '3'])
```

[142]:
```
'1 - 2 - 3'
```

Vytvoření nového objektu vs modifikace objektu

- Vytvoření nového objektu:

[143]:
```
old = 'Spam spam spam.'
new = old.replace('spam', 'ham')
print('Old:', old)
print('New:', new)
```

```
Old: Spam spam spam.
New: Spam ham ham.
```

[144]:
```
old = [1, 8, 5, 2]
new = sorted(old)
print('Old:', old)
print('New:', new)
```

```
Old: [1, 8, 5, 2]
New: [1, 2, 5, 8]
```

- Modifikace objektu:

[145]:
```
old = [1, 8, 5, 2]
new = old.sort()
print('Old:', old)
print('New:', new)
```

```
Old: [1, 2, 5, 8]
New: None
```

[146]:
```
old = [1, 8, 5, 2]
new = old.append(0)
print('Old:', old)
print('New:', new)
```

```
Old: [1, 8, 5, 2, 0]
New: 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:

[147]:
```
a = [1, 2, 3]
b = [1, 2, 3]
```

[148]:
```
a == b
```

[148]:
```
True
```

[149]:
```
a is b
```

[149]:
```
False
```

[150]:
```
a.append(4)
print('a:', a)
print('b:', b)
```

```
a: [1, 2, 3, 4]
b: [1, 2, 3]
```

- Ten samý objekt:

[151]:
```
a = [1, 2, 3]
b = a
```

[152]:
```
a == b
```

[152]:
```
True
```

[153]:
```
a is b
```

[153]:
```
True
```

[154]:
```
a.append(4)
print('a:', a)
print('b:', b)
```

```
a: [1, 2, 3, 4]
b: [1, 2, 3, 4]
```

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

[155]:
```
a = [6, 8, 3, 1]
b = a.copy()
a.sort()
print('a:', a)
print('b:', b)
```

```
a: [1, 3, 6, 8]
b: [6, 8, 3, 1]
```