

C2184 Úvod do programování v Pythonu (2021)

5. Kolekce

Kolekce

Jsou objekty, které obsahují strukturovaně více hodnot.

Mezi základní typy kolekcí patří:

- **seznam** (list) – upravovatelný soubor hodnot s daným pořadím, [1, 5, 1]
- **n-tice** (tuple) – neupravovatelný 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] == [3, 2, 1]
```

[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
```

- Sřetěžení

```
[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
```

Metoda `join`

- Opak `split`
- Spojuje prvky pomocí separatoru
- Funguje na seznamy i jiné iterovatelné objekty, ale prvky musí být typu `str`

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

```
[25]: '1 - 2 - 3'
```

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

```
[26]: '123'
```

Modifikace seznamů

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

```
[27]: [1, 2, 3, 4, 5]
```

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

```
[29]: numbers
```

```
[29]: [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)

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

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

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

```
[32]: numbers
```

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

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

```
[34]: numbers
```

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

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

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

```
[36]: numbers
```

```
[36]: [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)

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

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

```
[38]: 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)

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

```
[40]: numbers
```

```
[40]: [1, 3, 4, 5]
```

- Metoda `.clear()` odstraní všechny prvky

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

```
[42]: numbers
```

```
[42]: []
```

Změna směru

- Metoda `.reverse()`

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

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

```
[45]: numbers
```

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

Řazení

- Metoda `.sort()`

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

```
[47]: numbers
```

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

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

```
[49]: numbers
```

```
[49]: [3, 2, 1, 0, 0]
```

- Řazení řetězců – podle abecedy (pozor, nefunguje pro smíšenou velikost písmen a diakritiku)

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

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

```
[51]: animals.sort(key=len) # řadíme podle délky  
animals
```

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

Příklady použití seznamu

- V řetězci najděte všechny číslice a spočítejte jejich součin.

```
[52]: string = 'j52s1f4hdas8jlk21f'  
  
numbers = []  
for char in string:  
    if char.isdigit():  
        numbers.append(int(char))  
print(numbers)  
  
product = 1  
for number in numbers:  
    product *= number
```

```
print(product)
```

```
[5, 2, 1, 4, 8, 2, 1]  
640
```

- V větě nahradte čtvrté slovo slovem SPAM.

```
[53]: sentence = 'We interrupt this program to annoy you and make things,  
↳generally more irritating.'  
words = sentence.split()  
words[3] = 'SPAM'  
for word in words:  
    print(word, end=' ')
```

```
We interrupt this SPAM to annoy you and make things generally more,  
↳irritating.
```

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)`

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

```
[54]: ('Vlhká', 5, 'Brno')
```

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

```
[55]: (1.0, 2.5)
```

```
[56]: animals
```

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

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

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


Indexování n-tic

- Jako u řetězců a seznamů

```
[58]: address[0]
```

```
[58]: 'Vlhká'
```

```
[59]: address[1]
```

```
[59]: 5
```

```
[60]: address[2]
```

```
[60]: 'Brno'
```

```
[61]: address[:2]
```

```
[61]: ('Vlhká', 5)
```

“Nultice” (*empty tuple*)

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

```
[62]: ()
```

“Jednice” (*singleton*)

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

```
[63]: (5,)
```

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

```
[64]: 5
```

- Pozor na rozdíl mezi “jednicí” a samotným prvkem

```
[65]: (5,) == 5
```

```
[65]: False
```

Operace s n-ticemi

- Jako u seznamů

```
[66]: address
```

```
[66]: ('Vlhká', 5, 'Brno')
```

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

```
[67]: 3
```

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

```
[68]: 0
```

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

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

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

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

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

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

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

- Každý prvek je obsažen nejvíc jednou

```
[72]: {1, 1, 2, 2, 2, 3}
```

[72]: {1, 2, 3}

- Pořadí není dané

[73]: {1, 2, 3} == {3, 2, 1}

[73]: True

- Nelze indexovat

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

```
Traceback (most recent call last):  
  File "/tmp/ipykernel_3102/3567638886.py", line 1, in <module>  
    numbers = {1, 2, 3}  
----> 2 numbers[0]  
TypeError: 'set' object is not subscriptable
```

TypeError: 'set' object is not subscriptable

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

[75]: type(set())

[75]: set

[76]: type({})

[76]: dict

Množinové operace

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

[77]: A = {1, 2, 3}

[78]: len(A)

[78]: 3

```
[79]: 2 in A
```

[79]: True

```
[80]: 5 in A
```

[80]: 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 $A \Delta B = (A \setminus B) \cup (B \setminus A)$
- `<=` - inkluze $A \subseteq B$
- `<` - ostrá inkluze $A \subset B$

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

```
[82]: A & B # Průnik  $A \cap B$ 
```

[82]: {2}

```
[83]: A | B # Sjednocení  $A \cup B$ 
```

[83]: {1, 2, 3, 4, 6}

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

[84]: {1, 3}

```
[85]: A ^ B # Symetrický rozdíl  $A \Delta B$  (prvky  $A$  kromě prvků  $B$  plus prvky  $B$ 
      ↪ kromě prvků  $A$ )
```

[85]: {1, 3, 4, 6}

```
[86]: A <= B #  $A$  je podmnožinou  $B$  ( $A \subseteq B$ , tj.  $B$  obsahuje všechno z  $A$ )
```

[86]: False

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

[87]: False

Přidávání prvků

- Metoda `.add(x)` přidá prvek `x`

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

[88]: {5, 6}

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

[89]: {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)

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

[90]: {0, 1, 2, 3, 4}

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

```
[92]: A
```

[92]: {0, 1, 3, 4}

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

```
[93]: A
```

[93]: {0, 1, 3, 4}

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

0

{1, 3, 4}

- Metoda `.clear()` odebere všechny prvky

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

```
[96]: A
```

```
[96]: 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)

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

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

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

```
[98]: {}
```

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

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

Indexování slovníku

- Pomocí klíče

```
[100]: phonebook
```

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

```
[101]: phonebook['jack']
```

```
[101]: 4098
```

```
[102]: phonebook['bob']
```

↪ -----

KeyError Traceback (most recent call last)

/tmp/ipykernel_3102/3886755006.py in <module>
----> 1 phonebook['bob']

KeyError: 'bob'

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

```
[103]: print(phonebook.get('jack'))
```

```
4098
```

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

```
None
```

```
[105]: print(phonebook.get('bob', '???')) # nastavujeme defaultní hodnotu '?'  
↪ ???
```

```
???
```

Testování přítomnosti klíče

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

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

```
[106]: True
```

```
[107]: 4127 in phonebook
```

```
[107]: False
```

```
[108]: 4127 in phonebook.values()
```

```
[108]: 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

```
[109]: phonebook
```

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

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

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

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

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

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

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

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

Odebírání hodnot

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

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

```
4098
```

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

- Metoda `.clear()` vyprázdní celý slovník

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

```
[114]: {}
```


Pořadí v slovníku

- Nezáleží na pořadí

```
[115]: {1: 10, 2: 20} == {2: 20, 1: 10}
```

```
[115]: True
```

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

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

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

Příklad použití slovníku

- V proměnné winners máme pořadí účastníků závodu. Vypište jejich jména abecedně a k nim jejich pořadí.

```
[117]: winners = 'Bob Alice Dana Cyril Gustáv Emil Filoména'

names = winners.split()
positions = {}
for i, name in enumerate(names):
    positions[name] = i

names.sort()
for name in names:
    print(name, positions[name]+1)

# Šlo by řešit i pomocí seznamu a metody index, ale bylo by méně
↪ efektivní
```

```
Alice 2
Bob 1
Cyril 4
Dana 3
Emil 6
Filoména 7
Gustáv 5
```

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.

```
[118]: # list of tuples
addresses = [('Kamenice', 5, 62500, 'Brno'), ('Kotlářská', 2, 61137, 'Brno'), ('Botanická', 68, 30200, 'Brno')]
```

```
[119]: # dict of lists
telephones = {
    'LF': ['549 49 1305', '549 49 5346', '549 49 6855', '549 49 7179'],
    'PřF': ['549 49 3303', '549 49 6039', '549 49 3577', '549 49 6628', '549 49 5549', '549 49 5918'],
    'FI': ['549 49 1816', '549 49 1805'],
}
```

```
[120]: # dict of tuples with lists
contacts = {
    'LF': ('Kamenice', 5, 62500, 'Brno', ['549 49 1305', '549 49 5346', '549 49 6855', '549 49 7179']),
    'PřF': ('Kotlářská', 2, 61137, 'Brno', ['549 49 3303', '549 49 6039', '549 49 3577', '549 49 6628', '549 49 5549', '549 49 5918']),
    'FI': ('Botanická', 68, 30200, 'Brno', ['549 49 1816', '549 49 1805']),
}
```

Shrnutí

	Seznam	n-tice	Množina	Slovník
Typ	list	tuple	set	dict
Prázdná	[]	()	set()	{}
S prvky	[1, 2, 3]	(1, 2, 3)	{1, 2, 3}	{1: 10, 2: 20}
Pořadí	✓	✓	✗	✗nezáleží / ✓drží
Duplikáty	✓	✓	✗	✗klíč / ✓hodnota
Modifikace	✓	✗	✓	✓
Využití	obdobné objekty v pevném pořadí	spojení hodnot se specifickým významem	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 přidávat/mazat z obou stran
- ...

Volba typu kolekce

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

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

Řešení

1. Seznam reálných čísel
2. Slovník (klíče `str`, hodnoty `float`)
3. Seznam `n-tic`
4. Množina řetězců
5. Slovník (klíče `str`, hodnoty `int`)
6. Slovník (klíče `str`, hodnoty seznamy řetězců)

Rozbalování (*unpacking*)

- Pomocí operátoru *
- Vytáhneme všechny prvky z iterovatelného objektu

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

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

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

```
1 2 3 4
```

```
[123]: [0, numbers, 5, 6]
```

```
[123]: [0, [1, 2, 3, 4], 5, 6]
```

```
[124]: [0, *numbers, 5, 6]  # Stejně jako [0, 1, 2, 3, 4, 5, 6]
```

```
[124]: [0, 1, 2, 3, 4, 5, 6]
```

- Pomocí více proměnných na levé straně přiřazení

```
[125]: 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
```

```
[126]: 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
```

```
[127]: 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

```
[128]: for number in [10, 32, 9]:  
        print(number)
```

```
10  
32  
9
```

```
[129]: for number in (10, 32, 9):  
        print(number)
```

```
10  
32  
9
```

```
[130]: for number in {10, 32, 9}:  
        print(number)
```

```
32  
9  
10
```

Procházení slovníků

- Přes klíče

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

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

```
guido  
jack
```

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

guido
jack

- Přes hodnoty

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

4127
4098

- Přes klíče a hodnoty

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

guido: 4127
jack: 4098

Vytvoření nového objektu vs modifikace objektu

- Vytvoření nového objektu:

```
[136]: a = 'Spam spam spam.'  
        b = a.replace('spam', 'ham')  
        print('a:', a)  
        print('b:', b)
```

a: Spam spam spam.
b: Spam ham ham.

```
[137]: a = [1, 8, 5, 2]  
        b = a[::-1]  
        print('a:', a)  
        print('b:', b)
```

a: [1, 8, 5, 2]
b: [2, 5, 8, 1]

- Modifikace objektu:

```
[138]: a = [1, 8, 5, 2]  
        b = a.sort()  
        print('a:', a)  
        print('b:', b)
```

a: [1, 2, 5, 8]
b: None

```
[139]: a = [1, 8, 5, 2]
      b = a.append(0)
      print('a:', a)
      print('b:', b)
```

```
a: [1, 8, 5, 2, 0]
b: 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:

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

```
[141]: a == b
```

```
[141]: True
```

```
[142]: a is b
```

```
[142]: False
```

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

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

- Ten samý objekt:

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

```
[145]: a == b
```

```
[145]: True
```

```
[146]: a is b
```

```
[146]: True
```

```
[147]: 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)

```
[148]: 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]