List (seznam)

```
numbers = [10, 20, 5, 10]
point = ["Point 1", 10, 20]
zeros = [0] * 10

squares[1]
# 20

squares[-3:]  # vrací nový seznam
# [20, 5, 10]
```

- Může obsahovat různé datové typy
- Možnost indexace a "rozkrojení" (slicing)
- Jedná se o *mutable* typ
- Obvykle jsou listy předány ve formě odkazu, tzn. provedené změny se propíšou do všech "instancí"

```
rgb = ["Red", "Green", "Blue"]
rgba = rgb
rgba.append("Alph")

# rgb i rgba odkazují na stejný objekt
# rgb = ["Red", "Green", "Blue", "Alph"]
```

- Je možné je použít jako *zásobník* či *frontu*

```
from collections import deque

stack = [3, 4, 5]
>>> stack.append(6)
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6

>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")
>>> queue.append("Graham")
>>> queue.popleft()
'Eric'
>>> queue.popleft()
'John'
>>> queue
deque(['Michael', 'Terry', 'Graham'])
```

- Defaultně řada užitečných funkcí pro práci se seznamy

```
# maximální prvek
```

```python
max(numbers)

# minimální prvek
min(numbers)

# vytvoření seřazeného seznamu
sorted(numbers)

# součet prvků seznamu
sum(numbers)

# otočení pruků v seznamu
reversed(numbers)

# vytvoření seznamu z řetězce
list("123456")
# ['1', '2', '3', '4', '5', '6']
```

Dvojrozměrný seznam

```python
matrix = [
    [1, -2, 5, 20],
    [0, 2, 3, 400],
    [100, 2, 3, 4]
]

# PROBLÉM - vytvoření matice 2x3 vyplněné nulama
matrix_zeroes = [[0] * 3] * 2

# pozor, nastane však problém!
matrix_zeroes[0][0] = 1

# správně tedy, v budoucnu se dozvíme ještě elegantnější řešení
matrix_zeroes = []

for _ in range(2):
    matrix_zeroes.append([0] * 3)

matrix_zeroes[0][0] = 1
```

Tuple

```python
point = (10, 20)
point = 10, 20

len(point)
```

```
# nelze!
point[0] = 20

# tuple unpacking
x, y = point
```

- Nejedná se o *mutable* typ (čili *immutable*)

Dictionary

- Vazba *klíč* - *hodnota* > [!info] >`dict.items()` … pro celou vazbu > `dict.keys()` … klíče > `dict.values()` … hodnoty

```
point = {"x": 10, "y": 20}
point = dict([("x", 10), ("y", 20)])

# test zda ma slovník klíč "z"
"z" in point

# odstranění položky s klíčem "z"
del point["z"]

# test zda ma slovník klíč "z"
"z" in point

# odstranění položky s klíčem "z"
del point["z"]

# seřazení seznamu klíčů
sorted(point)

# délka slovníku (konkrétně klíčů)
len(point)
```

- Slovníky lze zanořovat #### Iterace přes slovníky

```
knights = {'gallahad': 'the pure', 'robin': 'the brave'}
for k, v in knights.items():
    print(k, v)

# gallahad the pure
# robin the brave

for i, v in enumerate(['tic', 'tac', 'toe']):
    print(i, v)

# 0 tic
```

```
# 1 tac
# ...
```

- Při iterování přes 2 a více ssekvencí použijeme funkci `zip()`

```
questions = ['name', 'quest', 'favorite color']
answers = ['lancelot', 'the holy grail', 'blue']
for q, a in zip(questions, answers):
    print(f'What is your {q}?  It is {a}.')

# What is your name?  It is lancelot.
# What is your quest?  It is the holy grail.
# What is your favorite color?  It is blue.
```

Set (množina)

- Jedná se o neseřazenou kolekci neduplicitních prvků (stejně jak v matematice)
- Podporuje běžné množinové operace > [!info] > *sjednocení* - `union()` nebo `|` > *průnik* - `intersection()` nebo `&` > rozdíl - `difference()` nebo `-` > *symetrická diference* (sjednocení bez průsečíku) - `symetrci_difference()` nebo `^` > > **zápis**: `a.union(b)`

```
numbers = {10, 20, 5, 10}

# vytvoření množiny ze seznamu
numbers = [10, 20, 5, 10]
numbers = set(numbers)

# počet pruků v množině
len(numbers)

# testování zda má nebo nemá množina prvek
10 in numbers
100 not in numbers

# chyba - AttributeError, `set` NEMÁ metodu append
numbers.append(10)

# přidání pruku probíhá skrze add
numbers.add(20)
# opakované přidání nemá efekt
numbers.add(20)

# odebrání pruku
numbers.remove(20)
```

```
# POZOR, vytvoří slovník!
empty = {}

# toto je správný způsob
empty = set()
type(empty)

# množina písmen z řetězce
letters = set("ahoj svete")

# nelze! seznam je mutovatelný, tedy není hashovatelný, proto nemůže být prvkem množiny
lists = set([[1, 2], [3, 4], [1, 2]])

# tuple již může
tuples = set([(1, 2), (3, 4), (1, 2)])

# nelze! slovník jako celek je mutovatelný
dicts = set([{"a": 1, "b": 2}])

# pozor zde pracujeme pouze s klíči slovníku, ty hashovatelné být musí
keys = set({"a": 1, "b": 2})

# řetězce nejsou mutovatelné, proto jsou hashovatelné a proto mohou být v množině
strings = set(["ahoj", "svete"])
```

- Mutovatelnost množiny "řeší" tzv. `frozenset()`

```
# nelze, protože množina je mutovatelná
set([set([1, 2]), set([3, 4])])

# lze, protože frozenset není mutovatelný
set([frozenset([1, 2]), frozenset([3, 4])])

# nelze - frozenset není mutovatelný
frozenset([1, 2]).add(5)
```

Cyklus `for`

- Používá se pro průchod sekvencí

```
for char in "ahoj svete":
    print(char)
# obdobně pro seznam, množinu

# použití tuple unpackingu, velice populární a čisté řešení!
for x, y in ((10, 20), (5, 2), (20, 30)):
    print(x, y)
```

```python
# zanořený unpacking
cities = [
    ('Tokyo','JP',36.933, (35.689722, 139.691667)),
    ('Delhi NCR', 'IN', 21.935, (28.613889, 77.208889)),
    ('Mexico City', 'MX', 20.142, (19.433333, -99.133333))]

for name, short_name, population, (latitude, longitude) in cities:
    print(name, short_name, population, latitude, longitude)

# PEP8 - pojmenování nepoužité hodnoty
for x, _ in ((10, 20), (5, 2), (20, 30)):
    print(x)
```

Cyklus `while`

```python
numbers = [10, 20, 5, 10]

while numbers:
    print(numbers.pop())

# nekonečná smyčka
while True:
    print('Hello')
```

Sekvence a `if`

```python
# PEP8 - využívejte faktu, že prázdné sekvence jsou vyhodnoceny jako False
# správně
if not numbers:
    pass

if numbers:
    pass

# špatně!
if not len(numbers):
    pass

if len(numbers):
    pass

# PEP8 - pořadí psaní podmínek
numbers = [1, 20, 5, 1]

# správně
```

```python
if 10 not in numbers:
    print("There is no 10!")

# špatně
if not 10 in numbers:
    print("There is no 10!")
```