

Lokálne vs. globálne premenné



- **Lokálna premenná:**
 - Dostupná len v tele funkcie, kde bola deklarovaná
 - Nemožno volať mimo tela funkcie
 - Lokálnu premennú zmeníme na globálnu použitím **global** pred názvom premennej
- **Globálna premenná:**
 - Dostupná kdekoľvek v tele Python súboru

```
c = 10

def handle_variable():
    c = 15
    a = 20
    # z premennej c vytvoríme globálnu premennú použitím global
    # global c

    # premenná a je lokálna premenná
    # Python vráti chybu
    print(a)

    # premenná c je globálna premenná
    # Python vráti 10
    handle_variable()
    print(c)
```

Dátové štruktúry

- Vďaka dátovým štruktúram vieme efektívne ukladať dáta a urobiť náš algoritmus **rýchlejší**
- Základ dátových štruktúr predstavuje ukladať dáta efektívnym spôsobom – pri operáciach insert, remove
- Štruktúra dát má veľký dopad na celkový čas behu programu

„Zlí programátori majú starosti ohľadom kódu, dobrí programátori majú starosti ohľadom dátových štruktúr a ich vzťahov“ Linus Torvalds (vývojár Linuxu)



Polia (Arrays)



- **Polia** sú dátové štruktúry, kde jednotlivé položky poľa sú identifikované tzv. **indexom**, začínajúcí 0
- Patria medzi rýchle dátové štruktúry
- Položky poľa sú umiestnené vpravo, vedľa seba navzájom, v RAM pamäti, čo je dôvod, prečo sú prístupné cez index
- Do položky poľa môžeme uložiť čokoľvek: int, str, Array, Object, ...
- **Polia** podľa veľkosti:
 - **statické** (nemenia veľkosť)
 - **dynamické** (menia veľkosť dynamicky)

• Kedy pracovať s poliami v Pythone:

- Chceme prístup k položkám poľa cez známy index
- Chceme manipulovať s poslednými položkami poľa
- Poznáme počet položiek, ktoré chceme vkladať do poľa (nie je dynamická štruktúra poľa)
- Ukladať do poľa rôzne dátové typy

Pole my_array[]

Index	Hodnota
0	34
1	-5
2	another_array[]
3	„string“
4	Object

Dvojrozmerné pole nums[][]

	0	1	2	3
0				
1				
2				
3				
				nums[2][3]

Zoznamy (Lists)

- Vo väčšine prípadov pole chápeme:

```
my_array = [1, 5, 7, 9, 11]
```

Tento zápis nazývame v Pythone **List**, nie je typický array

- Ak chcem použiť array, potom musíme použiť NumPy arrays
- **list** funkcia - vytvorí array, `my_list = list(1, 2, 3, 4)`
- Výpis z listu `print(my_array[0])`
- Položky zoznamu (lists) sú indexované, prvá položka má index [0], druhá položka má index [1], atď.
- Očakáva sa rovnaká veľkosť dát v jednotlivých položkách zoznamu
- list (zoznam) má vlastnosti: **zoradenosť**, **zmeniteľnosť**, **umožňujúca duplicitné hodnoty**
- **Zoradený** list znamená – položky majú definované poradie a toto poradie sa nezmení. Ak do zoznamu pridávame nové položky, tie sa umiestnia na koniec zoznamu
- **Zmeniteľný** (mutable) znamená, po vytvorení môžeme položky v zozname meniť, pridávať a odstraňovať
- **Umožňujúce duplicity** znamená: zoznam môže obsahovať rovnaké hodnoty položiek
- **Zoznamy** (lists) sú **výhodné** použiť, ak pracujeme s poslednou položkou zoznamu (`insert`, `update`, `remove`) – rýchlosť operácie je väčšia

**** NumPy implementuje skutočné jednorozmerné (aj viacrozmerné) polia, ktoré sú až 50-krát rýchlejšie ako tradičné zoznamy v Pythone.**



TM

Zoznamy (Lists)

```
# Zoznam - list
my_list = ["Jonathan Toews", "Brent Seabrook", "29", 50, 46.29, [25, 39, "Scores"]]

# Vrátenie dĺžky zoznamu
len(my_list)

# Vymazanie položky zoznamu
del my_list[0]

# remove vymaže prvý výskyt znaku
my_list.remove("29")
print(my_list)

# count vráti počet výskytu daného prvku
print(my_list.count("Brent Seabrook"))

# index vráti pozíciu prvého výskytu daného prvku
print(my_list.index("Brent Seabrook"))

# Pridanie položky zoznamu
my_list.append("Marian Hossa")

# Insert vloží druhý prvok v zátvorke na pozíciu 4
my_list.insert(4, "Erik Gustafsson")

# Funkcia extend rozbali pole a pridá na koniec listu
my_list.extend(["Joe Quenneville", 200, 300])

# Funkcia reverse obráti poriadok v zozname
my_list.reverse()
print(my_list)
```



Zoznamy (Lists)



```
# Upravovať vieme aj pomocou zloženého indexu
my_list=[0,1,2,3,4,5]
my_list[1:3]
# Nahradíme prvky s indexom 1 a 2
my_list[1:3] = [20, 25]
print(my_list)
my_list[1:3] = [30, 40, 50]
print(my_list)
# Odstránime prvok s indexom 1 a 2
my_list[1:3] = []
print(my_list[1:3])

# Vráti poslednú položku zoznamu
print(my_list[-1])
# my_list[-2] vráti predposlednú (-2) položku zoznamu
print(my_list[-2])
```

```
# Funkcia pop vráti posledný prvok z listu a zmaže ho
my_list=[1,2,3,4,5]
my_list.pop()
print(my_list)

# Na nasčítanie nových prvkov musíme výsledok operácie priradiť
# do premennej, ináč sa pôvodná premenná nezmení
my_list = [0,1,2,3,4,5]
my_list + [30, 40]
print(my_list)
# priradenie do premennej
my_list_new = my_list + [50, 70]
print(my_list_new)
```

List comprehension

- Umožňuje vytvoriť nový zoznam (list) pozostávajúci z existujúcich prvkov hodnôt iného zoznamu

new_list = [variable for item in my_list if condition is true]



```
my_list = [1, 7, 10, 29, 14, -6, 58]
new_list = []

# Vrátí zoznam párných čísel
for number in my_list:
    if number % 2 == 0:
        new_list.append(number)

print(new_list)

# Rovnaký zápis - list comprehension
new_list = [number for number in my_list if number % 2 == 0]
print(new_list)

# Vrátí zoznam mien hráčov, ktorých dĺžka je viac ako 12 znakov
my_list = ["Jonathan Toews", "Brent Seabrook", "Patrick Kane", "Joe"]
new_list = [name for name in my_list if len(name) > 12]
print(new_list)
```


Tuples

- Tuples sú podobné zoznamom (lists), ako ďalší dátový typ na ukladanie dát
- Položky v tuple nemôžu byť zmenené (tuple je immutable objekt), tzn. pridávať, mazať, meniť **nie je povolené**
- Sú rýchlejšie ako zoznamy (napr. pri vyhľadávaní)
- Ak sa veľkosť tuples nemení (dáta nepribúdajú), použiť tuples namiesto lists (menej alokovanej pamäte RAM)
- Využívajú sa hlavne v slovníkoch (dictionaries)
- Prístup k položkám cez index
- Tuple má vlastnosti: **zoradenosť, nezmeniteľnosť** (immutable), **umožňujúci duplicitné hodnoty**
- Tuple charakterizuje okrúhla zátvorka pozostávajúca min. z 2 položiek
 - `my_tuple = (1,)` – toto je tuple
 - `my_tuple = ("Chicago")` – toto **nie je** tuple, ale string



```
# Toto je Tuple
my_tuple = ("Jonathan Toews", "Patrick Kane", "Joe Quenneville", 40, 22.6, -1)

# Toto nefunguje, Tuple je immutable dátový typ
my_tuple[2] = "Erik Gustafsson"

# Výpis Tuple
print(my_tuple[2])

# Ina definícia Tuple
my_tuple = 1,2,3,4,5

# Tuple ako návratová hodnota z funkcie
def return_my_tuple(var1, var2):
    return var1, var1
return_my_tuple("Chicago", "Blackhawks")

# Spojenie (join) Tuple
tuple1 = ("Erik", "Joe", "Brandon")
tuple2 = ("Gustafsson", "Quenneville", "Saad")

tuple3 = tuple1 + tuple2
print(tuple3)
```


Dictionaries (slovníky)

- Slovníky sú abstraktné dátové typy
- dáta v slovníkoch predstavujú jednorozmerné pole
- Skladajú sa z páru **key->value**, každý kľúč sa nachádza najviac 1-krát v celom slovníku
- Každá položka vie byť identifikovaná indexom, začínajúca indexom 0, 1, 2,...
- Umožňujú ukladať rôzne dátové typy
- Sú mutable dátové typy
- Slovník má vlastnosti: **neusporiadanosť, zmeniteľnosť a neumožňuje duplicity**
- **Neusporiadanosť**: položky slovníku nemajú definované poradie, vieme na položku odkazovať indexom (Python ver. 3.6 a staršie, ver. 3.7 a novšie položky sú usporiadané)
- **Zmeniteľnosť**: položky v slovníku vieme meniť, pridávať a odstraňovať
- **Neumožňujúce duplicity**: slovník nemôže obsahovať položky s rovnakým kľúčom, **key**
- Zápis:
 - `my_dictionary = {key: value, key2: value, ...}` alebo
 - `my_dictionary = dict(key = value, key2 = value, ...)`



Dictionaries (slovníky)



```
my_dict = {
    'name': 'Tomas',
    'surname': 'Tatar',
    'number': 21,
    'EU_player': True,
    'teams': {'AHL': 'Grand Rapids', 'NHL': ['Detroit', 'Las Vegas', 'Montreal']},
    'career': {
        'start': 'Dubnica nad Vahom',
        'NHL_start': 'Detroit',
        'now': 'Montreal'
    }
}

# Kopírovanie slovníkov
new_dict = my_dict.copy()
print(new_dict)

# Výpis položiek key->value zo slovníka
for key, value in my_dict.items():
    print('Key is: ' + key + " and value is: " + str(value))

# Výpis položiek
print(my_dict['name'] + ' ' + my_dict['surname'])
print(my_dict['career']['now'])
print(my_dict['teams']['NHL'][-1])

# Výpis listu vnútornej slovníka
for item in my_dict['teams']['NHL']:
    print(item)
```

```
# Prístup na základe kľúču, ktorý nevráti chybu, ak neexistuje
# Vieme definovať hodnotu, ktorú vráti, ak kľúč neexistuje
my_dict.get('born')
my_dict.get('not_existing_key', 'Key born not found')

# keys vráti ako list kľúčov
my_dict.keys()

# values vráti ako list hodnôt
# odporúčané popužiť ako list(my_dict.keys())
my_dict.values()

# items vráti všetko key->value slovníka ako list tuples
list(my_dict.items())

# Vráti hodnotu kľúča a kľúč vymaže zo slovníka
my_dict.pop("number")

# vymaže posledné key->value zo slovníka a vráti ho ako tuple
my_dict.popitem()

# Vymazanie položky zo slovníka
del my_dict['EU_player']

# Pridanie položky do slovníka
my_dict['position'] = 'left wing'
```

Sets (zostavy)

- Sets sú implementované pomocou slovníkov (využívajú pole)
- Slúžia na ukladanie rôznych dát, objektov
- Sú mutable
- **Vlastnosti:** neusporiadanosť, neindexovanosť, nezmeniteľnosť, nepovoľuje duplicity
- **Neusporiadanosť:** položky nemajú definované poradie (náhodne usporiadané)
- **Nezmeniteľnosť:** samotné položky nevieme meniť, obsah položiek musí byť immutable dátový typ, ale sets samo o sebe vieme zmeniť
- **Neindexovanosť:** nevieme pristupovať k položkám, nepoznáme pozíciu
- **Neumožňujúce duplicity:** sets **neobsahujú** položky s rovnakou hodnotou
- Sets sú opozitom zoznamov (lists), pri prehľadávaní položiek sú lists o niečo rýchlejšie než sets, na opačnú stranu sets sú omnoho rýchlejšie pri nájdení konkrétnej položky v sekvencii položiek
- sets sa využívajú na testovanie členstva prvku v sekvencii, odstránenie duplicít zo sekvencie, na výpočet matematických operácií, ako je prienik, zjednotenie, rozdiel
- Zápis:
 - `my_set = {value1, value2, value3, ...}` alebo
 - `my_set = set((value1, value2, value3, ...))`



Sets (zostavy)



```
my_set = {"Tomas", "Tatar", 21, True, "Montreal"}

# Nefunguje, vráti chybu
# print(my_set[0])

# Výpis set: {True, 'Tatar', 21, 'Tomas', 'Montreal'}
# Poradie popložiek nie je zachované !
print(my_set)

# Kontrola výskytu položky
if "Tatar" in my_set:
    print("Found one")

for item in my_set:
    print(item)

# Pridanie položky
my_set.add("left wing")

# Pridanie viacerých položiek
my_set.update([29, "Detroit", False])
```

```
# Dĺžka set
len(my_set)

# remove() vs discard()
# remove() vráti chybu, ak položka neexistuje
# discard() nevráti chybu, ak položka neexistuje
my_set.discard("born")

# pop funkcia vráti poslednú NÁHODNÚ položku a vymaže ju
print(my_set.pop())

# union spojí 2 sets do jedného, odstráni DUPLICITY
set1 = {1, 10, "Joe"}
set2 = {1, "Joe", "Patrick", "Kane", 29}
result = set1.union(set2)
print(result)

# Rozdiel dvoch sets vráti 10
set1 = {1, 10, "Joe"}
set2 = {1, "Joe", "Patrick", "Kane", 29}
result = set1.difference(set2)
print(result)
```