Seznamy

Již známe následující typy dat:

jednoduché:
- číselné ***int*** a ***float***
- logický ***bool***

posloupnost:
- sekvence znaků ***str***
- sekvence řádků - otevřený textový soubor
- posloupnost čísel - pomocí **range()**

Nyní se seznámíme s datovou strukturou ***seznam:***

V Pythonu se tento typ nazývá ***list()***, je to vlastně posloupnost hodnot libovolného typu (v angličtině se také používá termín kolekce) a říkáme, že typ seznamu se skládá z prvků. Kromě názvu ***seznam*** můžeme použít také název ***tabulka*** nebo ***pole*** (pole většinou pro seznamy hodnot stejného typu).

Seznamy vytváříme zápisem prvků v hranatých závorkách[]:

```
>>>prazdny = []                    # prázdny seznam
>>>zviera = ['pes', 'Dunco', 2011, 35.7, 'hneda']
>>>type(zviera)
<class 'list'>
```

Operace se seznami

Základní operace se seznamem fungují téměř přesně tak, jak je používáme s řetězci znaků:

indexování s hranatými závorkami ***[ ]*** - je přesně stejné jako u řetězců:
(index je celé číslo od 0 do počtu prvků seznamu - 1, nebo je to záporné číslo)

```
>>>zviera[0]
  'pes'
```

zřetězení pomocí znaku ***plus +*** znamená, že vytvoříme nový větší seznam, který obsahuje nejprve prvky prvního seznamu a poté všechny prvky druhého seznamu, například:

```
>>> [1] + [2] + [3, 4] + [] + [5]
   [1, 2, 3, 4, 5]
```

vícenásobné zřetězení pomocí znaku hvězdička * označuje, že seznam je vzájemně zřetězen za zadaný počet opakování, například:

```
>>> jazyky = ['Python', 'Pascal', 'C++', 'Java', 'C#']
>>> vela = 3 * jazyky
>>> vela
   ['Python', 'Pascal', 'C++', 'Java', 'C#', 'Python', 'Pascal', 'C++', 'Java', 'C#', 'Python',
    'Pascal', 'C++', 'Java', 'C#']
```

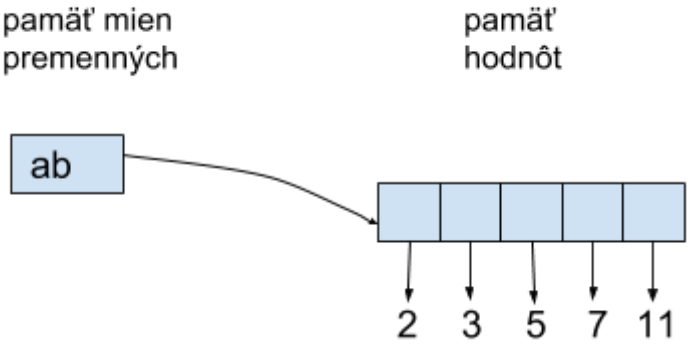
zjišťování přítomnosti prvku pomocí příkazu ***in*** prověřuje, zda seznam obsahuje hledaný prvek, například:

```
>>> teploty = [10, 13, 15, 18, 17, 12, 12]
>>> 18 in teploty
True
>>> 'Y' not in 'Python'
True
```

Struktura seznamu

ab = [2, 3, 5, 7, 11]

Do paměti názvů (globální jmenný prostor) je přidán jeden identifikátor proměnné *ab* a také odkaz na seznam pěti prvků. Tento seznam si můžeme představit jako pět polí vedle sebe v paměti hodnot, z nichž každý má odkaz na odpovídající hodnotu:



Je dobře si uvědomit, že v současné době máme v paměti 6 proměnných, z nichž jedna je *ab* (je typu list) a dalších pět je *ab[0], ab[1], ab[2], ab[3], ab[4]* a (všechny jsou typu *int*).

Šablona na vytváření seznamu prvků s různými typy hodnot

Pokud potřebujeme v cyklu přiřadit do seznamu prvků různé hodnoty, musí už existovat příslušné místa (indexy). Nejlépe je takovýto seznam připravit jedním přidělením a výsenásobným zřetězením:

```
zoznam = [None] * n
for i in range(n):
        zoznam[i] = ... výpočet hodnoty
print(zoznam)
```

Procházení prvků seznamu

Procházení seznamu pomocí ***for-cycle***:

```
>>>for prvok in teploty:
print(prvok, end='')
  10, 13, 15, 18, 17, 12, 12,
```

Procházení seznamu pomocí***for-cycle s indexováním***:

```
>>> teploty =  [10, 13, 15, 18, 17, 12, 12]
>>>for i in range(7):
print(f'{i+1}. deň', teploty[i])
  1. deň 10, ...
```

Procházení seznamu pomocí ***for-cycle s indexováním s enumerate***:

```
>>>for i, prvok in enumerate(teploty):
print(f'{i+1}. deň', prvok)
1. deň 10, ...
```

Šablony pro zjišťování hodnot v seznamu

Počítání hodnot prvků (prvky, které bychom mohli například násobit nebo zřetězit):

```
sucet = 0
for prvok in zoznam:
    sucet = sucet + prvok
print(sucet)
```

Zkontrolování informace o prvcích v seznamu, například minimální prvek:

```
mn = zoznam[0]
for prvok in zoznam:
    if prvok <mn:
        mn = prvok
print(mn)
```

Výstup seznamu prvků v jednom řádku:

```
zoznam = [47, 'ab', -13, 22, 9, 25]
print('prvky zoznamu:', end=' ')
for prvok in zoznam:
    print(prvok, end=' ')
print()
```

Procházení seznamu v určitém cyklu:

```
for i in range(10):
    index = i % 4
    farba = ['red', 'blue', 'yellow', 'green'][index]
print(farba)
```

toto bychom také mohli napsat takto:

```
for i in range(10):
    print(['red', 'blue', 'yellow', 'green'][i%4])
```

Změna hodnoty prvku seznamu

Datová struktura seznamu je proměnného typu (tzv. mutable). Změnou obsahu jednoho prvku seznamu se změní pouze reference, vše ostatní zůstává bezezměny.

Můžeme změnit hodnoty prvků seznamu **přiřazením**:

```
ab[2] = 55
```

Nebo **for-cyklem:**

```
teploty = [10, 13, 15, 18, 17, 12, 12]
for i in range(len(teploty)):
    teploty[i] += 2
print(teploty)
>>>[12, 15, 17, 20, 19, 14, 14]
```

Vždy ale musíme přistupovat k prvkům seznamu pomoci indexů.

Standardní funkce se seznami

Následující funkce fungují nejen se seznamy, ale i s jakoukoliv libovolnou posloupností hodnot:

len(postupnosť) -> vrátí počet prvků se sekvence
sum(postupnosť) -> vypočítá číselný součet prvků se sekvence
max(postupnosť) -> vrátí maximální prvek sekvence (tj. jeho hodnotu)
min(postupnosť) -> vrátí minimální prvek sekvence

Funkce list()

Již máme určité zkušenosti s tím, že v Pythonu má každý základní typ definovanou svou konverzní funkci, pomocí které lze některé hodnoty různých typů převést na daný typ.
Například:

int(3.14) -> vrátí celé číslo 3
int('37') -> vrátí celé číslo 37
str(22 / 7) -> vrátí řetězec '3.142857142857143'
str(2 < 3) -> vrátí řetězec 'True'

Podobně funguje i funkce ***list(hodnota)***

Parametr funkce musí být *iterovatelná* hodnota, tj. sekvence, kterou lze procházet (iterovat), například pomocí for-cycle. Funkce ***list()*** tuto sekvenci rozebírá na prvky a z těchto prvků seskládá nový seznam.
Pokud parametr bude chybět, vygeneruje prázdný seznam.

Například:

```
>>>list(zviera)                    # kópiaexistujúcehozoznamu
   ['pes', 'Dunco', 8, 35.7, 'hneda']
>>>list(range(5, 16))
   [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>>list('Python')
   ['P', 'y', 't', 'h', 'o', 'n']
>>>list()                         # prázdnyzoznam
   []
```

Formát pro vytvoření **seznamu z textového souboru** za pomoci funkce *list()*:

```
list(open('subor.txt', encoding='utf-8'))
```

Řezy

Když jsme potřebovali změnit znak v řetězci znaků, museli jsme vždy vytvořit kopii řetězce. Použili jsme zde řezy (slice), tj. získání podřetězců. Totéž lze použít i při práci se seznamy:

```
>>> jazyky =  ['Python', 'Pascal', 'C++', 'Java', 'C#']
>>> jazyky[1:3]
   ['Pascal', 'C++']
>>> jazyky[1::2]
   ['Pascal', 'Java']
>>> jazyky[::-1]
   ['C#', 'Java', 'C++', 'Pascal', 'Python']
```

Řezy nemění obsah samotného seznamu, a proto říkáme, že jsou neměnné (immutable).

Přiřazování do řezu

Když například vybereme podseznam pomocí řezu ***zoznam[od:do:krok]***, taková operace s původním seznamem nic neudělá, jen vytvoří zcela nový seznam. Můžeme však také změnit obsah seznamu tak, že změníme pouze jeho část. Sekce seznamu tedy může být na levé straně přiřazovacího příkazu a pak na pravé straně přiřazovacího příkazu musí být nějaká posloupnost (nemusí to být seznam). Přiřazovací příkaz tuto posloupnost projde, vytvoří z ní seznam a ten přiřadí na místo udaného řezu:

```
abc = list('Python')
abc[2:2] = ['dve', 'slova']          # rez dĺžky 0 sa nahradí dvomi prvkami
abc
>>>['P', 'y', 'dve', 'slova', 't', 'h', 'o', 'n']
```

A protože tím modifikujeme původní seznam, nezíváme tuto operaci měnitelnou (mutable).

Porovnání seznamů

Seznamy můžeme porovnávat mezi sebou (pro rovnost, nebo menší/větší). Funguje to na stejném principu jako porovnávání řetězců znaků. Postupně se procházejí jednotlivé prvky jednoho a druhého seznamu dokud jsou stejné. Je-li jeden ze prvků menší, považuje se jeho seznam za menší. Při porovnávání prvků je výsledek porovnání těchto dvou různých hodnot je výsledkem porovnání celého seznamu.
Každé dva porovnávané prvky musí být Python schopen porovnat: pro rovnost je to bez problémů, ale relační operace <> nebude fungovat například pro porovnávání čísel a řetězců

Například:

```
>>> [1, 2, 5, 3, 4] > [1, 2, 4, 8, 1000]
True
>>> [1, 'ahoj'] == ['ahoj', 1]
False
>>> [1, 'ahoj'] < ['ahoj', 1]
  ...
TypeError: '<' not supportedbetweeninstancesof 'int' and 'str'
```

Seznam jako parametr funkce

Průměr jsme dříve počítali pomocí *for-cyklu* a proměnných pro *počet* a *součet*.
Pro seznam můžeme pomocí standardních funkcí sum() a len() zjistit průměr takto:

```
def priemer(zoznam):
  return sum(zoznam) / len(zoznam)
```

Další funkce detekuje počet výskytů určité hodnoty v seznamu:

```
def pocet(zoznam, hodnota):
vysl = 0
for prvok in zoznam:
        if prvok == hodnota:
                vysl += 1
 return vysl
```

Toto pak u jakékoliv iterovatelné struktury můžeme zapsat pomocí metody ***count()***:

```
>>> 'bla-bla-bla'.count('l')
  3
```

Metody

Metodami voláme funkce které pracují s nějakou hodnotou, za kterou dáváme tečku a samotné jméno funkce s případnými parametry.

U seznamů to vypadá takto:

seznam.funkcia(parametre)

Pro seznamy existují dvě **immuteble** metody (neupravují seznam, pouze z něho čtou)***:***

metoda count() **-** vrátí počet výskytů dané hodnoty v seznamu.

zoznam.count(hodnota)

metoda index() **-** vrátí index prvního výskytu dané hodnoty v seznamu

zoznam.index(hodnota)

Při použití této metody musíme být opatrní, aby nám program nespadl, když hodnota není v seznamu. Pokud tomu chceme předejít, můžeme použít například následující zápis:

```
farby = ['red', 'blue', 'red', 'blue', 'yellow']
if 'green' in farby:
        index = farby.index('green')
else:
        print('"green" sa v zozname nenachádza')
>>>'green' sa v zozname nenachádza
```

Všechny další metody jsou **muteble**, to znamená, že budou modifikovat samotný seznam.

metóda append() **-** přidá na konec seznamu nový prvek - seznam se tak rozšíří o 1. Funkce nic nevrací, takže nemá smysl přiřazovat její volání do nějaké proměnné.

zoznam.append(hodnota)

šablona pro vytvoření seznamu pomocí příkazu append

Pokud potřebujeme vytvořit seznam různých hodnot, aniž bychom znali jeho výslednou délku, můžeme metodu *append* použít tímto způsobem:

```
zoznam = []                     # najprv je zoznam prázdny
for i in range(n):              # alebofor-cyklus preinúpostupnosť
        if ... nejaka_podmienka:
                nova_hodnota = ...
                zoznam.append(nova_hodnota)
print(zoznam)
```

metoda insert() **-** přidá nový prvek do umístění seznamu - tím se seznam zvýší o 1. Funkce nic nevrací, takže nemá smysl přiřazovat její volání do nějaké proměnné.

zoznam.insert(index, hodnota)

Uvědomte si, že ***zoznam.insert(len(zoznam), hodnota)*** vždy přidává na konec seznamu, to znamená, že dělá totéž jako ***zoznam.append(hodnota)***

metoda pop() **-** odstraní poslední prvek z konce seznamu - seznam se tak zkrátí o 1. **Funkce vrátí hodnotu odebraného prvku**. Pokud byl seznam prázdný, funkce nic nevrátí, ale spadne na chybu.

zoznam.pop()

metoda pop() s indexem **-** odstraní příslušný prvek (daný indexem) ze seznamu - seznam se tak zkrátí o 1. **Funkce vrátí hodnotu odebraného prvku**. Pokud byl seznam prázdný, funkce nic nevrátí, ale spadne na chybu.

zoznam.pop(index)

metoda remove() **-** odstraní ze seznamu první výskyt prvku s danou hodnotou - seznam se tak zkrátí o 1. Funkce nic nevrací. Pokud hodnota není v seznamu, funkce spadá na chybu.

```
zoznam.remove(hodnota)
```

metoda sort() **-** přeskupí prvky seznamu tak, aby byly uspořádány vzestupně - seznam tímto způsobem nemění svou délku. Funkce nic nevrací. Pokud prvky v seznamu nelze vzájemně porovnat (například existují čísla a řetězce), funkce spadá na chybu.

```
zoznam.sort()
```

Běžnou chybou začínajícího uživatele je přiřazení výsledku této metody (tj.None) do proměnné, například:

```
>>>abc = ['raz', 'dva', 'tri', 'styri']
>>>abc = abc.sort()                    # vždyvrátiNone
>>>print(abc)
None
```

Seznam jako výsledek funkce

První funkce **vytvoří seznam stejných hodnot**:

```
def urob_zoznam(n, hodnota=0):
    return [hodnota] * n
```

Můžeme jí například použít následujícím způsobem:

```
>>> pole1 = urob_zoznam(30)
>>> pole1
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Jiná funkce **vytvoří nový seznam z dané posloupnosti**:

```
def zoznam(postupnost):
    vysl = []
    for prvok in postupnost:
        vysl.append(prvok)
    return vysl
```

Uvědomte si, že to dělá téměř totéž jako volání funkce *list()*

```
>>>py = zoznam('Python')
>>>py
  ['P', 'y', 't', 'h', 'o', 'n']
```

Funkce pridaj() na základě jednoho seznamu **vytvoří nový seznam, na jehož konec přidá nový prvek**. Původní seznam zůstane nezměněn:

```
def pridaj(zoznam, hodnota):
    return zoznam + [hodnota]
```

Je tedy **neměnná**, protože nemění hodnotu žádného dříve existujícího seznamu.

Pokud bychom tuto funkci zapsali takto:

```
def pridaj1(zoznam, hodnota):
    zoznam.append(hodnota)
    return zoznam
```

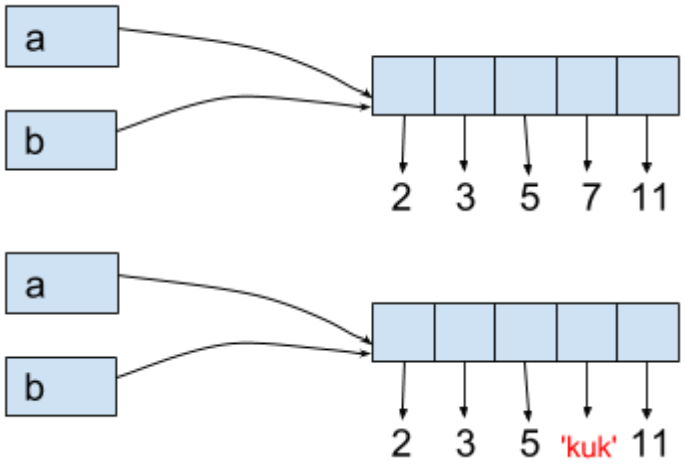
Voláním této funkce bychom získali velmi podobné výsledky, ale také změnil hodnotu prvního parametru - seznamu typů proměnných. Takto je tato funkce **proměnlivá**, protože změní typ parametru.

Dvě proměnné odkazují na stejný seznam

Přiřazení seznamu do proměnné znamená, že jsme ve skutečnosti přiřadili do proměnné odkaz na seznam, ten ale může mít více odkazů v naší paměti, například:

```
>>> a = [2, 3, 5, 7, 11]
>>> b = a
>>> b[3] = 'kuk'
>>> a
  [2, 3, 5, 'kuk', 11]
```

Změnili jsme obsah proměnné *b* (změnili jsme její prvek s indexem 3), ale to také změnilo obsah proměnné *a*:



Shrnutí:

Vložení do seznamu
Viděli jsme více různých způsobů, jak můžeme do seznamu přidat jednu hodnotu.
Vložení nějaké hodnoty před prvek s indexem i:

pomocí *řezu* (mutable):
zoznam[i:i] = [hodnota]

pomocí metody *insert()* (mutable):
zoznam.insert(i, hodnota)

když i == len(zoznam), přidáváme na konec (za poslední prvek), můžeme použít metodu *append()* (mutable):
zoznam.append(hodnota)

toho lze dosáhnout i takto (mutable):
zoznam += [hodnota]

Ve svých programech můžete použít způsob, který je pro vás nejlepší, ale zápis řezu je nejméně čitelný a používá se nejmíň.

Vyhození všech prvků ze seznamu
nejjednodušší způsob (neměnný), který můžeme použít pouze v případě, **když nepotřebujeme zachovat odkaz na původní seznam**, protože tímto rušíme odkaz na původní seznam a vytváříme nový. :
zoznam = []

Jiné metody vymazání seznamu, které uchovávají odkaz na seznam:

všechny prvky seznamu jsou postupně vyřazeny pomocí while cyklu
(jedná se o zbytečně velmi neefektivní řešení)

```
while zoznam:
    zoznam.pop()
```

přiřazením ke řezu
(je hůře čitelné a méně srozumitelné řešení)

```
zoznam[:] = []
```

metoda clear()
(je pravděpodobně nejčitatelnější způsob zápisu.)

```
zoznam.clear()
```

Vyřazení ze seznamu

Existuje také několik způsobů, jak vyhodit prvek ze seznamu.
Pokud přiřadíme prvek k indexu i, můžeme zapsat:

pomocí řezu:

```
zoznam[i:i+1] = []
```

pomocí příkazu del:

```
del zoznam[i]
```

pomocí metody pop(), která také vrátí vyhazovanou hodnotu:

```
hodnota = zoznam.pop(i)
```

velmi neefektivní použití metody *remove(),* která jako parametr neočekává index, ale vyhazovanou hodnotu:

```
zoznam.remove(zoznam[i])
```

tato metoda je velmi neefektivní (zbytečně hledá prvek v seznamu) a navíc může někdy vyhodit hodnotu, která odpovídá indexu a nachází se dříve než hledaná hodnota.

zřejmě funguje také:

```
zoznam = zoznam[:i] + zoznam[i+1:]
```

toto přidružení nezmění původní seznam, ale vytvoří nový seznam bez prvku s tímto indexem

Vytvoření kopie seznamu

Pokud potřebujeme vytvořit kopii celého seznamu, lze to provést pomocí cyklu:

```
kopia = []
for prvok in zoznam:
    kopia.append(prvok)
```

můžeme také použít řez:

```
kopia = zoznam[:]
```

nebo funkci list():

```
kopia = list(zoznam)
```