# Druhý zápočtový test

- Bude za 20 bodov.
- Uskutoční sa na cvičeniach v 11-tom týždni: 27.11., 28.11.
- Rovnako ako prvý test bude pozostávať z 3 úloh a čas na vypracovanie bude 45 minút.
- Rovnako ako prvý test bude za PC, odpovede na papier.
- Náplňou testu bude všetko, čo dovtedy preberieme, t.j. to, čo bolo na prvom teste + reťazce a zoznamy.

PROG1: Prednáška 9

# **Zoznamy** (lists)

*Časť prvá:* ***Základy***

# Zoznamy

Minulú prednášku sme sa bavili o reťazcoch – reťazec bol **štruktúrovaný dátový typ**, t.j. bola to hodnota, ktorá zároveň pozostávala z nejakého počtu jednoduchších elementov – znakov. Reťazec bol teda tvorený usporiadanou postupnosťou jednoduchých znakov.

Slovo **usporiadanou** znamená, že každý jednoduchý znak mal svoju pozíciu – **index.**

Navyše platí, že reťazec je tzv. **homogénny** štruktúrovaný dátový typ, t.j. jednotlivé hodnoty, z ktorých pozostáva – v tomto prípade znaky – sú rovnakého „typu“, teda všetko to boli jednoduché znaky.

Dnes sa budeme baviť o ďalšom **štruktúrovanom dátovom type** – **zoznam (angl. list).**

**Zoznam** je predstaviteľom **heterogénneho** štruktúrovaného dátového typu, t.j. zoznam bude znovu pozostávať z nejakej postupnosti hodnôt – nazveme ich **prvky zoznamu –** ktoré však môžu byť **ľubovoľného** typu. Zároveň platí, že zoznam je tvorený **usporiadanou** postupnosťou prvkov, teda **každý prvok zoznamu má svoj index**, podobne ako v prípade reťazcov.

# Vytváranie zoznamov

Zoznam môžeme vytvoriť napríklad priamo v zdrojovom kóde, vymenovaním jeho prvkov. Na vytváranie zoznamu používame hranaté zátvorky, každý prvok oddelíme čiarkou.

Podobne ako pri reťazcoch, aj pri zoznamoch môžeme vytvárať premenné, pomocou ktorých vieme so zoznamami pracovať jednoducho tak, že do nich priradíme požadovaný zoznam prvkov.

Následne vieme pomocou týchto premenných napríklad vypísať celý zoznam na obrazovku, pozri ďalší slajd.

# Vytváranie zoznamov

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
cele_cisla = [10, 20, 30] #zoznam obsahujuci 3 prvky - cisla 10, 20, 30
desatinne_cisla = [3.14, 2.718, 1.412] #zoznam obsahujuci 3 prvky - desatinne cisla
zoznam_retazcov = ["toto", "su", "styri", "retazce"] #zoznam 4 retazcov

#zoznamy su heterogenne, t.j. ich prvky mozu byt lubovolnych typov
#priklad zoznamu ktory obsahuje cele cislo, desatinne cislo, retazec a dalsi zoznam
nahodne_vymysleny_zoznam = [1, 2.5, "ahoj", [1,2]]

print(cele_cisla) #vypis zoznamu na obrazovku
print(desatinne_cisla) #vypis zoznamu na obrazovku
print(zoznam_retazcov) #vypis zoznamu na obrazovku
print(nahodne_vymysleny_zoznam) #vypis zoznamu na obrazovku
```

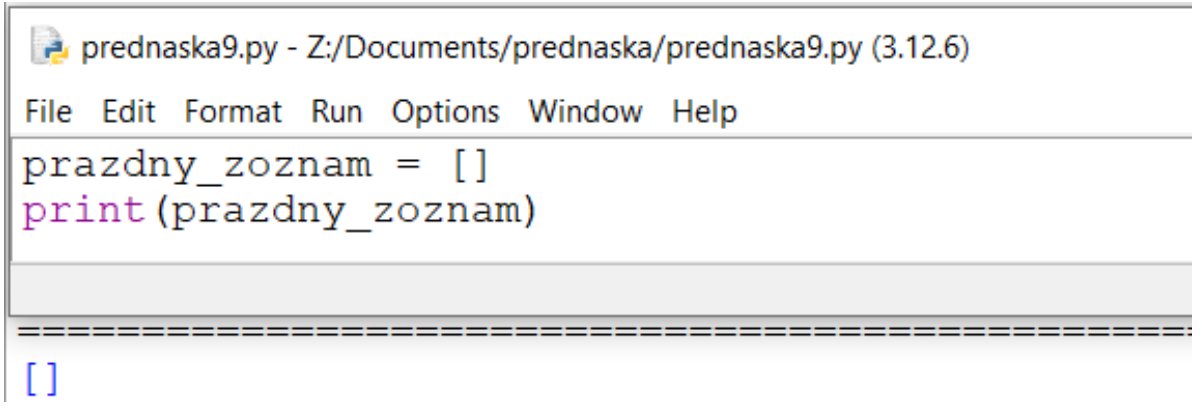
Ln: 11 Col: 21

```
[10, 20, 30]
[3.14, 2.718, 1.412]
['toto', 'su', 'styri', 'retazce']
[1, 2.5, 'ahoj', [1, 2]]
```

# Vytváranie zoznamov

Ako je vidieť v predošlom príklade, prvkom zoznamu môže byť **ďalší zoznam!**
V predošlom príklade bol posledným prvkom zoznamu *nahodne_vymysleny_zoznam* zoznam 2 čísiel [1,2]. Takýto zoznam, ktorý je zároveň prvkom iného zoznamu, nazývame **vnorený.**

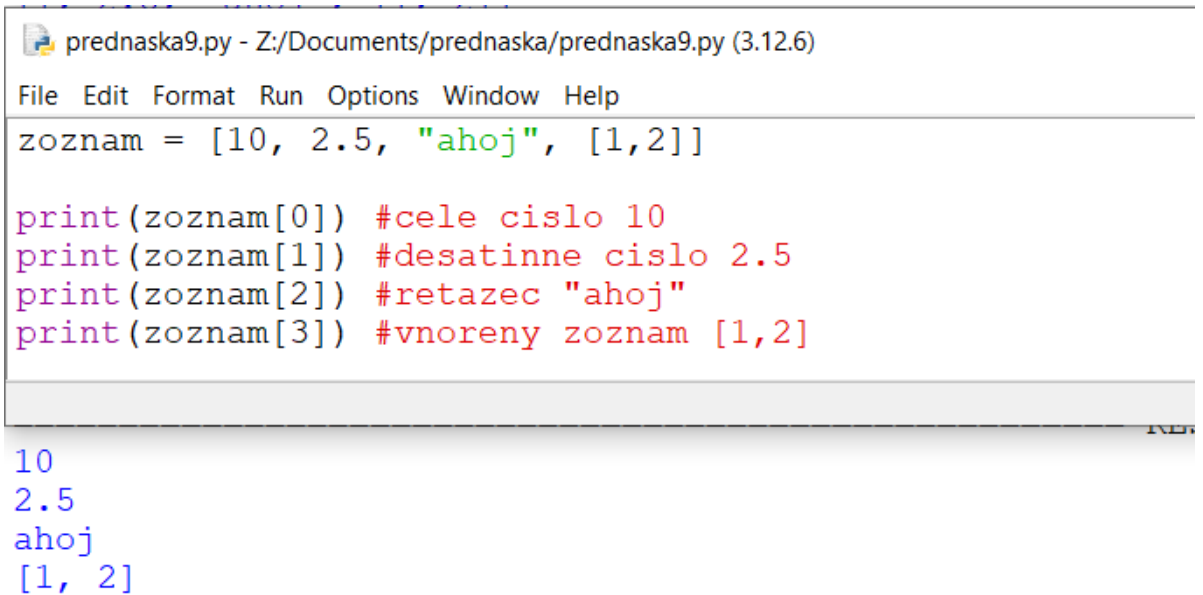
Ak potrebujeme, prázdny zoznam vytvoríme pomocou prázdnych hranatých zátvoriek:



# Prístup k prvkom zoznamu

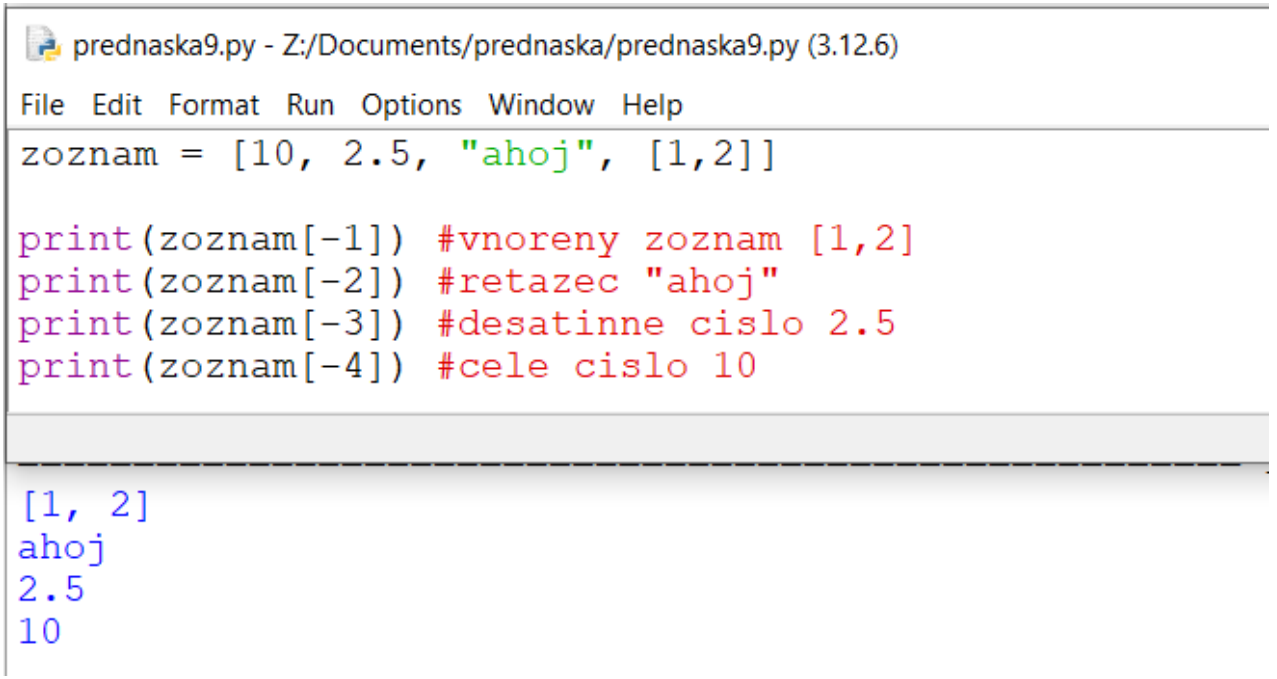
Podobne ako pri reťazcoch, aj pri zoznamoch pristupujeme k ich prvkom pomocou **indexov**.

Podobne ako pri reťazcoch, aj pri zoznamoch sú prvky indexované od **nuly**, teda prvý prvok má index 0, druhý prvok má index 1, atď.



# Prístup k prvkom zoznamu

Podobne ako pri reťazcoch, aj pri zoznamoch môžeme použiť indexáciu so zápornými indexami, t.j. index -1 má posledný prvok zoznamu, index -2 má predposledný prvok, atď.



# Prístup k prvkom zoznamu

Podobne ako pri reťazcoch, ak sa pokúsime o prístup k prvku na indexe, ktorý neexistuje, Python vypíše chybu:

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
zoznam = [10, 2.5, "ahoj", [1,2]]

print(zoznam[0]) #cele cislo 10
print(zoznam[1]) #desatinne cislo 2.5
print(zoznam[2]) #retazec "ahoj"
print(zoznam[3]) #vnoreny zoznam [1,2]
print(zoznam[4]) #prvok s takymto indexom neexistuje, Python vypise chybu IndexError
```

Ln: 3 Col: 14

```
10
2.5
ahoj
[1, 2]
Traceback (most recent call last):
  File "Z:/Documents/prednaska/prednaska9.py", line 7, in <module>
    print(zoznam[4]) #prvok s takymto indexom neexistuje, Python vypise chybu IndexError
IndexError: list index out of range
```

# Prístup k prvkom zoznamu

Podobne pri použití záporných indexov:

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
zoznam = [10, 2.5, "ahoj", [1,2]]

print(zoznam[-1]) #vnoreny zoznam [1,2]
print(zoznam[-2]) #retazec "ahoj"
print(zoznam[-3]) #desatinne cislo 2.5
print(zoznam[-4]) #cele cislo 10
print(zoznam[-5]) #prvok s takymto indexom neexistuje, Python vypise chybu IndexError
```

Ln: 5 Col: 38

```
[1, 2]
ahoj
2.5
10
Traceback (most recent call last):
  File "Z:/Documents/prednaska/prednaska9.py", line 7, in <module>
    print(zoznam[-5]) #prvok s takymto indexom neexistuje, Python vypise chybu IndexError
IndexError: list index out of range
```

# Zmena prvkov zoznamu

Na rozdiel od reťazcov, prvky zoznamu sú **meniteľné** (zoznamy sú „*mutable“)*

Pomocou prístupu k prvkom zoznamov cez indexy je možné meniť prvky v zozname na príslušnom indexe!

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t = [10, 2.5, "ahoj", [1,2]]
print(t)     #vypise cely zoznam ulozeny v premennej t
print(t[0]) #vypise prvy prvok zoznamu v premennej t

t[0] = -1.5 #zmenime prvy prvok zoznamu na desatinne cislo -1.5
print(t)     #vypise cely zoznam ulozeny v premennej t
print(t[0]) #vypise prvy prvok zoznamu v premennej t
```

```
[10, 2.5, 'ahoj', [1, 2]]
10
[-1.5, 2.5, 'ahoj', [1, 2]]
-1.5
```

# Operátor **in**

Podobne ako v reťazcoch, aj pri zoznamoch môžeme zistiť, či sa v zozname *zoznam* nachádza nejaký *prvok* pomocou booleovského operátora **in**, t.j. výraz:

*prvok* **in** *zoznam*

nadobúda hodnotu *True* ak sa *prvok* nachádza v zozname *zoznam* a nadobúda hodnotu *False* ak sa *prvok* nenachádza v zozname *zoznam*.

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t = [10, 2.5, "ahoj", [1,2]]

print (10 in t)     #True pretoze cislo 10 sa nachadza v zozname t

cislo = 2.5
print (cislo in t) #True pretoze hodnota premennej cislo, t.j. 2.5, sa nachadza v zozname t

print("ahoj" in t) #True pretoze zoznam t obsahuje retazec "ahoj"
print("aho" in t)  #False pretoze zoznam t neobsahuje retazec "aho"
```

```
True
True
True
False
```

# Operátor **in**

V prípade vnorených zoznamov sa ako prvok zoznamu chápe celý vnorený zoznam!

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

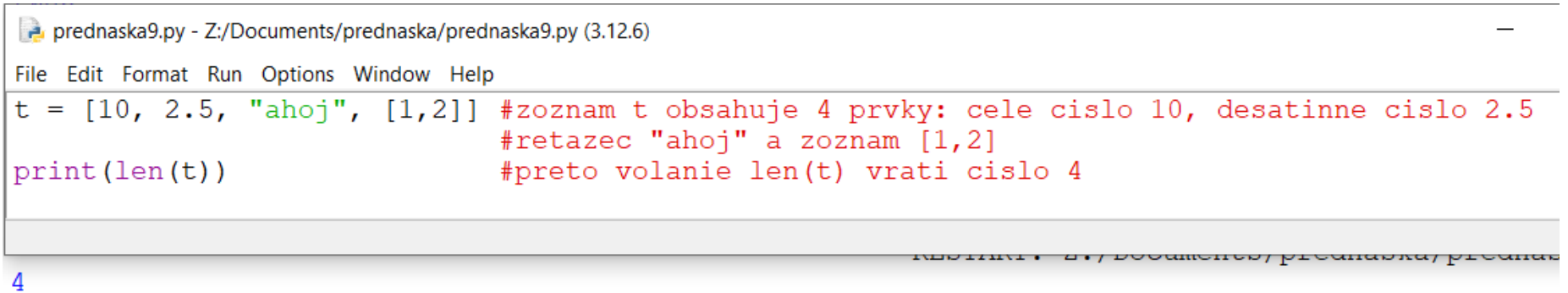
File Edit Format Run Options Window Help

```
t = [10, 2.5, "ahoj", [1,2]]

print ([1,2] in t)      #True pretoze zoznam [1,2] je prvkom zoznamu t

minizoznam = [1,2]
print(minizoznam in t) #True pretoze zoznam [1,2] je prvkom zoznamu t

minizoznam = [2,1]
print(minizoznam in t) #False pretoze zoznam [2,1] NIE je prvkom zoznamu t
                       #teda na poradi prvkov v zozname ZALEZI

print(1 in t)          #False kedze cislo 1 nie je prvkom zoznamu t
                       #hoci 1 je prvkom VNORENEHO zoznamu [1,2], avsak operator in
                       #sa nevnara do vnorenych zoznamov, ale vnima ich ako celok, t.j.
                       #v tomto pripade sa ako prvok zoznamu t uvazuje cely zoznam [1,2]
```

```
True
True
False
False
```

# Dĺžka zoznamu - funkcia **len()**

Podobne ako pri reťazcoch, funkciu **len()** je možné použiť aj na zistenie veľkosti zoznamu. POZOR! Funkcia **len**() vráti **počet prvkov zoznamu!** Ak je teda prvkom zoznamu napríklad vnorený zoznam, ten je považovaný za 1 prvok!



# Prechádzanie cez zoznam

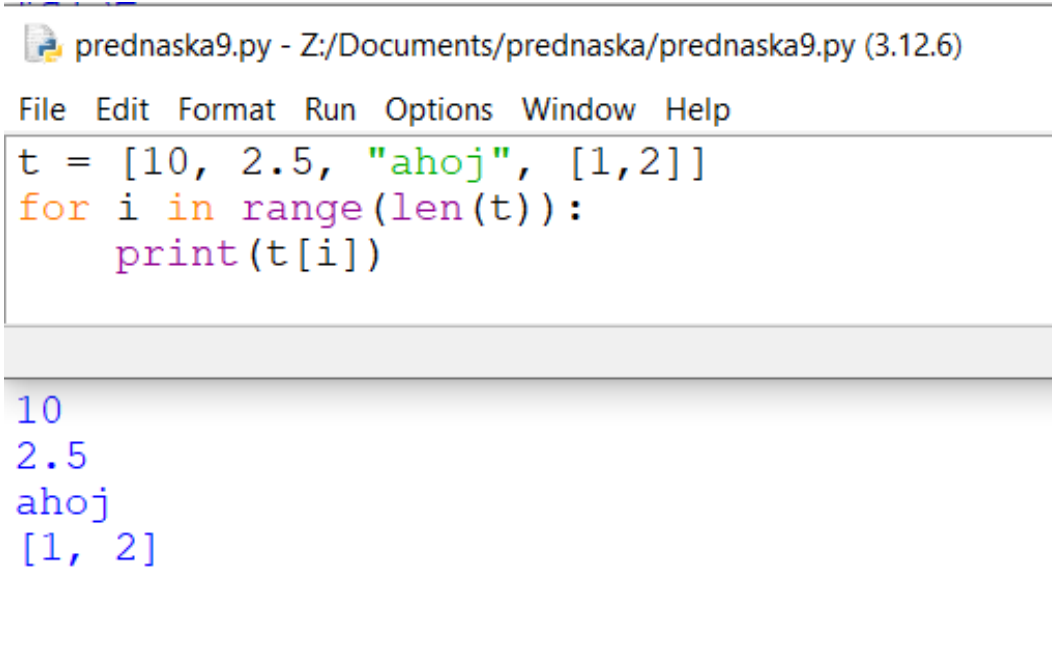
Podobne ako pri reťazcoch, existujú 2 hlavné prístupy pri prechádzaní cez zoznam, t.j. iteratívnom prístupe ku všetkým jeho prvkom:

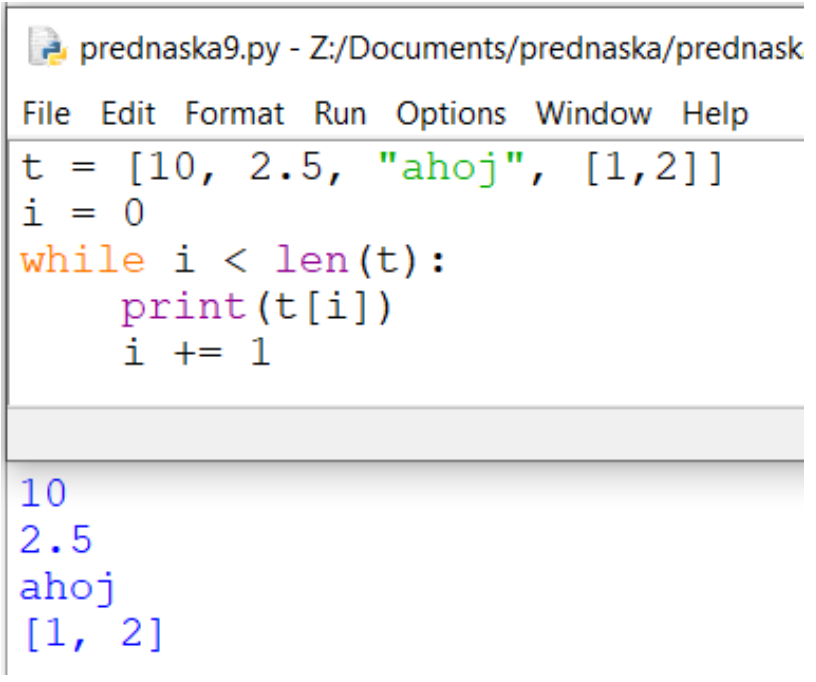
a) prechod cez **indexy** prvkov zoznamu

b) prechod **priamo cez prvky** zoznamu

# Prechádzanie prvkov zoznamu pomocou indexov

Pomocou cyklu (**while** alebo **for**) si môžeme elegantne postupne generovať indexy pre prístup k jednotlivým prvkom zoznamu. V oboch uvedených prípadoch vytvoríme premennú *i* , ktorú budeme používať ako index pri prístupe k prvkom, všimnite si, že bude postupne nadobúdať hodnoty od 0 po **len**(t)-1, vrátane. Vľavo príklad pomocou **for**-cyklu, vpravo **while-**cyklus.

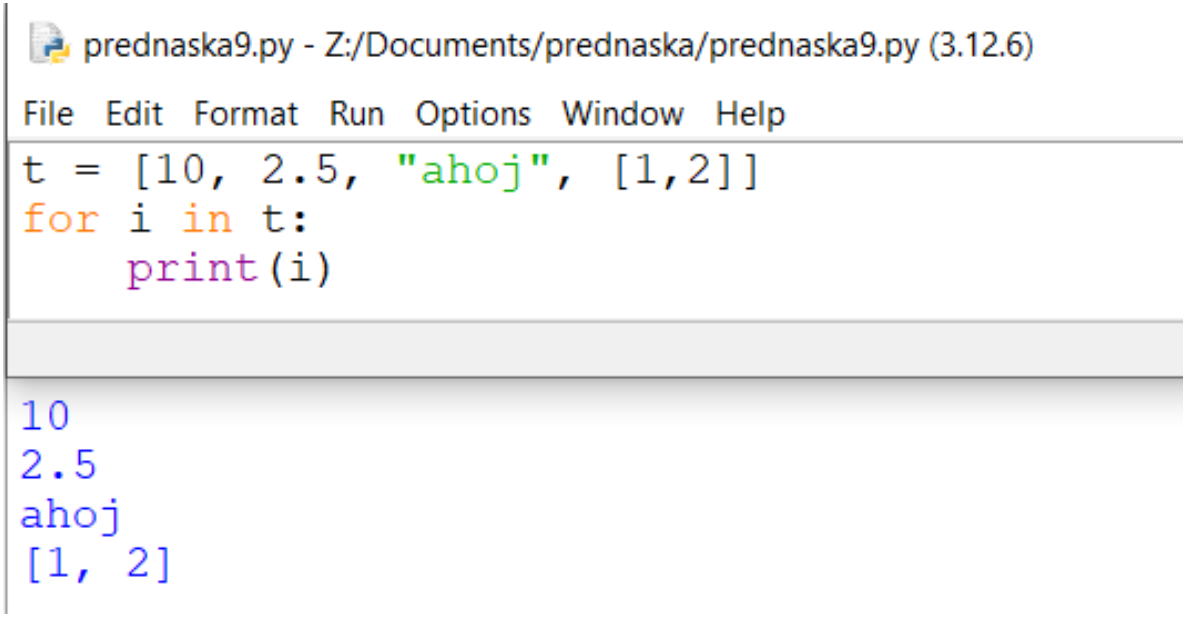


# Prechádzanie prvkov zoznamu priamo

Podobne ako pri reťazcoch, Python poskytuje možnosť prechádzať cez elementy zoznamu **priamo**. V tomto prípade to znamená, že sa vytvorí premenná, ktorá bude **priamo** nadobúdať hodnoty jednotlivých **prvkov zoznamu**.

V jazyku Python má uvedený prístup nasledovnú syntax pomocou príkazu **for**:



# Operátor + pre zoznamy

Tak ako je možné použiť operátor + pre reťazce v zmysle zreťazenia reťazcov, rovnako je možné použiť operátor + aj pre spojenie 2 zoznamov.

Ak *t1* a *t2* sú 2 zoznamy, potom *t1* + *t2* **vráti** zoznam, ktorý vznikne tak, že sa vytvorí nový zoznam, ktorý tvoria najprv prvky zoznamu *t1* a potom prvky zoznamu *t2*.

Teda zoznamy *t1* a *t2* ostanú nezmenené.

# Operátor + pre zoznamy

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [10, 2.5, "ahoj", [1,2]]
t2 = ['a', [2,3], -2.67, [], 5]
t3 = t1 + t2

print("Pocet prvkov t1: ", len(t1))
print("Zoznam t1: ", t1)
print("Pocet prvkov t2: ", len(t2))
print("Zoznam t2: ", t2)
print("Pocet prvkov t3: ", len(t3))
print("Zoznam t3: ", t3)
```

```
Pocet prvkov t1:  4
Zoznam t1:  [10, 2.5, 'ahoj', [1, 2]]
Pocet prvkov t2:  5
Zoznam t2:  ['a', [2, 3], -2.67, [], 5]
Pocet prvkov t3:  9
Zoznam t3:  [10, 2.5, 'ahoj', [1, 2], 'a', [2, 3], -2.67, [], 5]
```

# Operátor * pre zoznamy

Tak ako je možné použiť operátor * pre reťazec a číslo *k* a výsledkom je reťazec zreťazený *k*-krát sám so sebou, rovnako je možné použiť operátor * aj pre zoznam *t* a číslo *k*.

Výsledkom je nový zoznam, ktorý vznikne spojením zoznamu *t* samého so sebou *k*-krát.

Ak *t* je zoznam a *k* nezáporné celé číslo, potom *t* * *k* (alebo aj *k* * *t*) **vráti** zoznam, ktorý vznikne tak, že sa vytvorí nový zoznam, ktorý vznikne ako spojenie zoznamu *t* *k*-krát.

Teda pôvodný zoznam *t* ostane nezmenený.

# Operátor * pre zoznamy

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [10, 2.5, "ahoj", [1,2]]
t2 = t1*3
t3 = t1*0

print("Pocet prvkov zoznamu t1:",len(t1))
print("Zoznam t1: ",t1)
print("Pocet prvkov zoznamu t2:",len(t2))
print("Zoznam t2: ",t2)
print("Pocet prvkov zoznamu t3:",len(t3))
print("Zoznam t3: ",t3)
```

```
Pocet prvkov zoznamu t1: 4
Zoznam t1:  [10, 2.5, 'ahoj', [1, 2]]
Pocet prvkov zoznamu t2: 12
Zoznam t2:  [10, 2.5, 'ahoj', [1, 2], 10, 2.5, 'ahoj', [1, 2], 10, 2.5, 'ahoj', [1, 2]]
Pocet prvkov zoznamu t3: 0
Zoznam t3:  []
```

# Výseky zo zoznamu

Podobne ako pri reťazcoch, aj pri zoznamoch je možné používať výseky – t.j. slice-notáciu. Ak je teda *t* zoznam, a *n,m,k* sú celé čísla, potom *t*[*n:m:k*] **vráti** prvky zoznamu *t*:

- počnúc indexom *n*
- končiac indexom *m* (nie vrátane)
- s krokom *k*

Zároveň platí všetko to, čo sme si o výsekoch hovorili na prednáške o reťazcoch, t.j.:
- ak nie je uvedený počiatočný index, výsek sa robí od začiatku zoznamu
- ak nie je uvedený koncový index, výsek sa robí po koniec zoznamu
- ak nie je uvedený krok, výsek sa robí s krokom 1
- je možné uviesť záporný krok, v takom prípade to vráti verziu zoznamu s prvkami „odzadu“

# Výseky zo zoznamu

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [10, 2.5, "ahoj", [1,2], 'a', [], 5, [2,3,4]]

vysek = t1[1:6:2] #vysek od indexu 1 po index 6 (nie vratane) s krokom 2
                  #teda vrati zoznam s prvkami zo zoznamu t1 s indexami 1,3,5
print(vysek)

vysek = t1[2:]    #vysek od prvku s indexom 2 po koniec
print(vysek)

vysek = t1[:4]    #vysek od zaciatku po prvok s indexom 4 (nie vratane)
print(vysek)

vysek = t1[::-1]  #vysek od konca po zaciatok s krokom -1 (teda obrateny zoznam t1)
print(vysek)
```

```
[2.5, [1, 2], []]
['ahoj', [1, 2], 'a', [], 5, [2, 3, 4]]
[10, 2.5, 'ahoj', [1, 2]]
[[2, 3, 4], 5, [], 'a', [1, 2], 'ahoj', 2.5, 10]
```

# Výseky zo zoznamu

Rozdiel oproti reťazcom je ten, že pomocou slice-notácie vieme zoznamy aj **meniť**.

Konkrétne vieme pomocou slice-notácie **nahradiť** špecifikovanú časť zoznamu pomocou prvkov iného zoznamu:

prednaska9.py - Z:/Documents/prednaska/prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [10, 2.5, "ahoj", [1,2], 'a', [], 5, [2,3,4]]
t1[1:4] = [1,2] #v zozname t1 nahradi prvky od indexu 1 po 4 (nie vratane) prvkami 1 a 2
print(t1)

t2 = [1, 2, 3, 4, 5, 6]
t2[1:] = ["ahoj"] #v zozname t2 sa vsetky prvky od indexu 1 zmazu a vlozi sa don retazec "ahoj"
print(t2)

t3 = [1, 2, 3, 4, 5, 6]
t3[:5] = ["ahoj",[5,4,3]] #v t3 sa nahradia prvky na indexoch 0,1,2,3,4 prvkami "ahoj" a [5,4,3]
print(t3)
```

Ln: 10

```
[10, 1, 2, 'a', [], 5, [2, 3, 4]]
[1, 'ahoj']
['ahoj', [5, 4, 3], 6]
```

# Metódy pre zoznamy

Podobne ako pre reťazce, aj zoznamy majú v jazyku Python niekoľko dedikovaných metód, pomocou ktorých môžeme so zoznamami vykonávať rôzne typy operácií.

Zoznam metód pre zoznamy, spolu s popisom ich volania a činnosti, môžete nájsť:
https://docs.python.org/3/tutorial/datastructures.html

# Metódy pre zoznamy – výber zopár metód

Metóda *list*.**append**(*x*) – pridá prvok *x* do zoznamu *list* ako nový prvok na **jeho koniec**

```
t = [1,2,3]
t.append("a")
print(t)
```

```
[1, 2, 3, 'a']
```

Metóda *list*.**extend**(*x*) – argument *x* musí byť zoznam (resp. iný iterovateľný objekt). Metóda vezme prvky zoznamu *x* a doplní ich na **koniec** zoznamu *list*.

```
t = [1,2,3]
t.extend(["a","b"])
print(t)
```

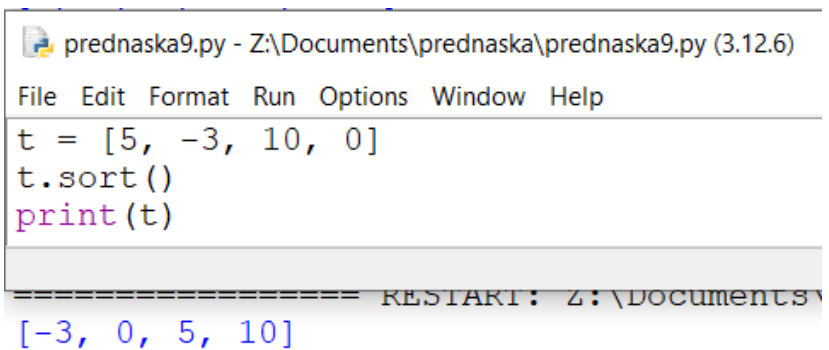
```
[1, 2, 3, 'a', 'b']
```

# Metódy pre zoznamy – výber zopár metód

Metóda *list*.**insert**(*i, x*) – vloží prvok *x* do zoznamu *list* na index *i*. Prvok, ktorý bol pôvodne na indexe *i* sa nezmaže, ale sa posunie na index *i+1*, prvok z indexu *i+1* sa posunie na index *i+2*, atď.

```
zoznam1 = [5, -10, 0, 3]
zoznam1.insert(0,1)
print(zoznam1)

[1, 5, -10, 0, 3]
```

Metóda *list*.**sort**() – utriedi prvky zoznamu *list* od najmenšieho po najväčší.



# Metódy pre zoznamy vs metódy pre reťazce

Všimnite si **rozdiel** medzi metódami pre reťazce a metódami pre zoznamy!

Metódy pre reťazce spravidla **vracajú nový reťazec** – napríklad metóda **upper()** vráti verziu reťazca, kde každé malé písmeno je nahradené veľkým písmenom. Teda pôvodný reťazec zostane zachovaný!

```
ret1 = "ahoj"
ret2 = ret1.upper() #upper VRATI novy retazec ako navratovu hodnotu!
print(ret1)         #retazec ret1 ostane nezmeneny, t.j. "ahoj"
print(ret2)         #v premennej ret2 je novy retazec "AHOJ"
```

```
ahoj
AHOJ
```

# Metódy pre zoznamy vs metódy pre reťazce

Všimnite si **rozdiel** medzi metódami pre reťazce a metódami pre zoznamy!

Metódy pre zoznamy spravidla **upravujú pôvodný zoznam** – napríklad metóda **sort()** zoradí pôvodný zoznam a **nič nevracia!!!**

```
zoznam1 = [5, -10, 0, 3]
zoznam2 = zoznam1.sort() #sort zoradi POVODNY zoznam, avsak vracia None!
print(zoznam1)           #zoznam1 sa zmeni na zoradeny od najmensieho po najvacsie
print(zoznam2)           #v premennej zoznam2 je hodnota None pretoze ju vracia metoda sort()!
```

Ln: 2 Co

```
[-10, 0, 3, 5]
None
```

**Preto treba pozorne čítať dokumentáciu k metódam, kde je vždy uvedené, čo metóda vracia a čo robí!**

# Mazanie prvkov zo zoznamu

Odstránenie prvku na danom indexe cez metódu **pop()**

- metóda *list.***pop()** - zo zoznamu *list* odstráni **posledný** prvok a zároveň **vráti** jeho hodnotu ako svoju návratovú hodnotu.
- metóda *list*.**pop(** *i* **)** - ak metóde **pop()** poskytneme ako argument celé číslo, odstráni sa zo zoznamu *list* prvok **na indexe** *i*.

prednaska9.py - Z:\Documents\prednaska\prednaska9.py (3.12.6)

File Edit Format Run Options Window Help

```
zoznam1 = [5, -10, 0, 3]
prvok = zoznam1.pop(1)    #odstrani prvok na indexe 1, teda cislo -10 a vrati ho ako navratovu hodnotu
print(prvok)
print(zoznam1)

zoznam2 = ["a", "b", "c"]
prvok = zoznam2.pop()    #pop bez argumentu odstrani posledny prvok a vrati ho ako navratovu hodnotu
print(prvok)
print(zoznam2)
```

Ln: 9

```
-10
[5, 0, 3]
c
['a', 'b']
```

# Mazanie prvkov zo zoznamu

Odstránenie prvku na danom indexe cez príkaz **del**

- pomocou príkazu **del** *t*[*i*] odstráni zo zoznamu *t* prvok na indexe *i*
- *pomocou príkazu* ***del*** *t*[*n:m:k*] odstráni zo zoznamu *t* prvky v zmysle slice-notácie, t.j. od indexu *n* po index *m* (nie vrátane) s krokom *k*

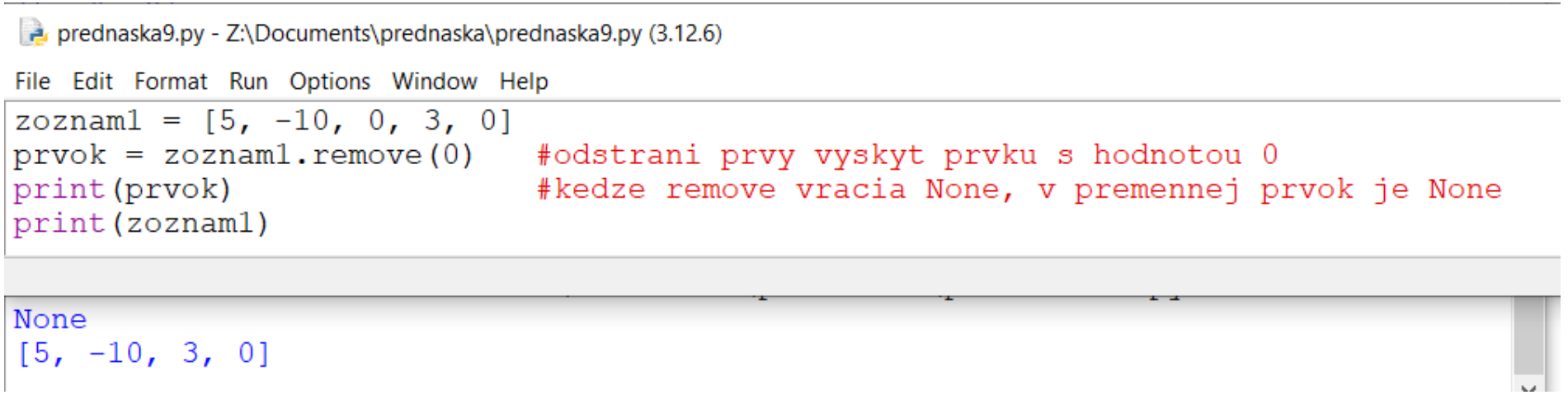
```
zoznam1 = [5, -10, 0, 3, 0]
del zoznam1[1]
print(zoznam1)

zoznam2 = [10,20,30,40,50,60,70,80,90]
del zoznam2[1:6:2] #odstrani prvky na indexoch 1, 3, 5
print(zoznam2)
```

```
[5, 0, 3, 0]
[10, 30, 50, 70, 80, 90]
```

# Mazanie prvkov zo zoznamu

Odstránenie prvku s danou hodnotou cez metódu **remove()**

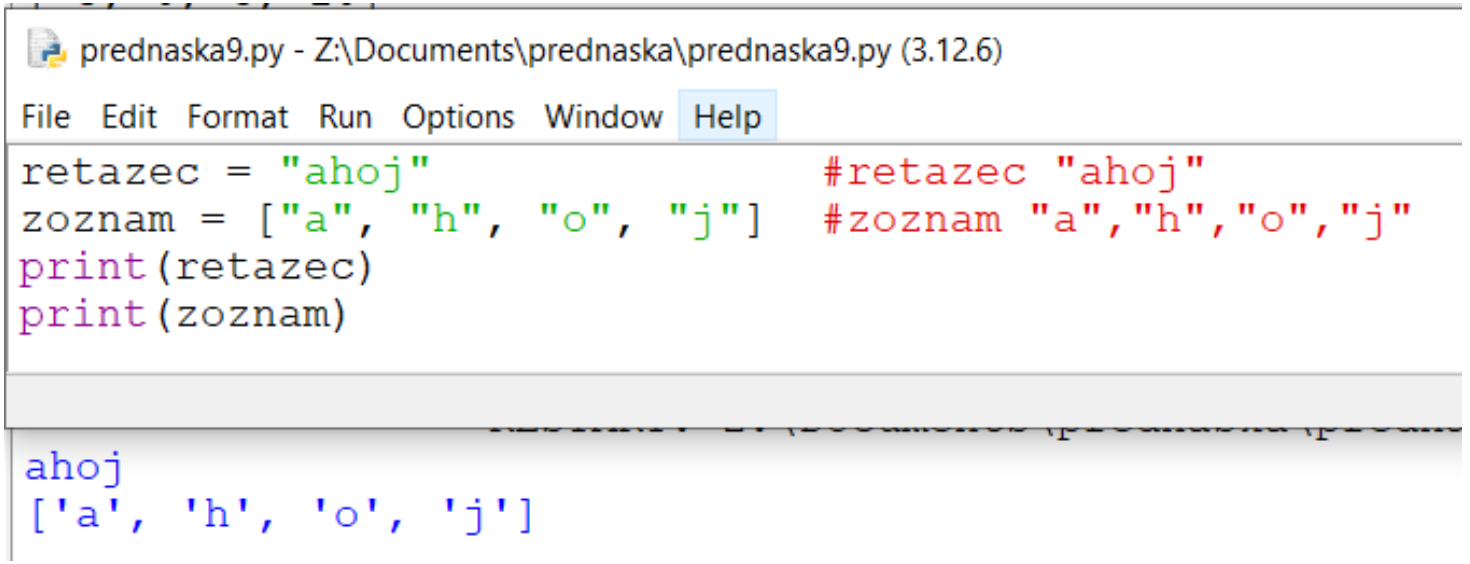
- metóda *list.***remove(***x***)** - zo zoznamu *list* odstráni prvý výskyt prvku s hodnotou *x*. Táto metóda nič nevracia (resp. vracia None).



# Zoznamy versus reťazce

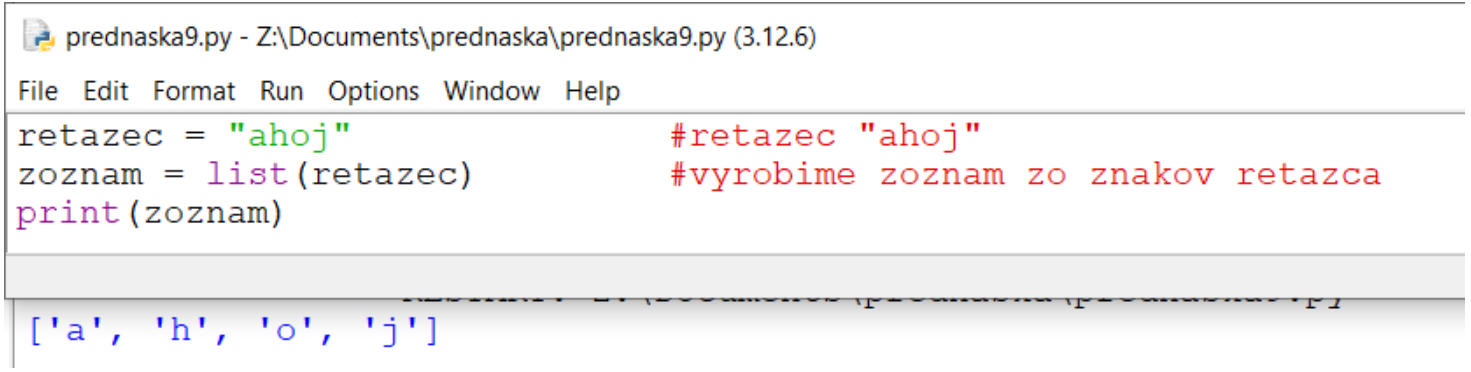
Ako sme povedali, zoznamy sú heterogénna usporiadaná postupnosť prvkov, reťazce sú homogénna usporiadaná postupnosť znakov.

Avšak zoznam znakov **nie je to isté** ako reťazec!



# Zoznamy versus reťazce

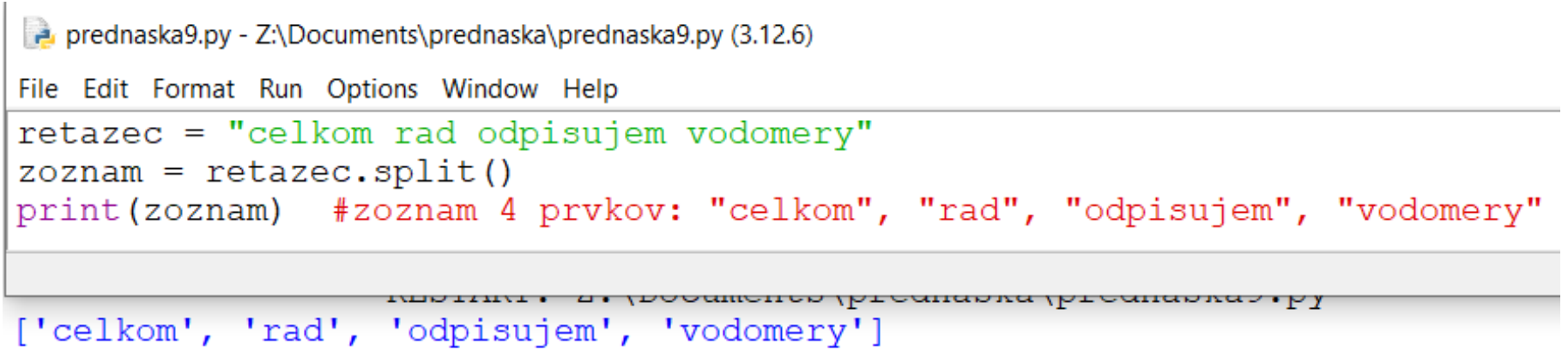
Ak chceme reťazec previesť na zoznam príslušných znakov, môžeme použiť funkciu **list()**, ktorá vezme iterovateľný objekt (napríklad reťazec) a vráti ho ako zoznam príslušných prvkov – v prípade reťazcov teda volanie **list***(retazec)* vráti zoznam znakov reťazca *retazec*



# Zoznamy versus reťazce

Reťazec *str* je možné rozdeliť pomocou určeného znaku (*delimiter)* na podreťazce pomocou metódy *str.***split(***delimiter***).** Volanie **vráti** zoznam príslušných podreťazcov, pričom pôvodný reťazec *str* ostane nezmenený.

V prípade, že sa *delimiter* neuvedie, urobí sa rozdelenie pomocou tzv. whitespace-znakov (patrí sem napríklad medzera).



# Zoznamy versus reťazce

Ak chceme zo zoznamu urobiť reťazec, potom môžeme použiť reťazcovú metódu **join().** Ak *iterable* je zoznam reťazcov, potom metóda *str.***join***(iterable)* vezme reťazce zo zoznamu *iterable* a zreťazí ich do jedného reťazca, pričom ich pospája pomocou reťazca *str*.

```
oddelovac = "a"
zoznam = ["b","n","n"]
vysledok = oddelovac.join(zoznam) #spoji "b" "n" "n" pomocou oddelovaca "a", teda "b"+"a"+"n"+"a"+"n"
print(vysledok)
```

Ln: 3 Col

```
banan
```

Ak by sme teda ako reťazec *str* použili prázdny reťazec “”, dostanem priamo reťazec pozostávajúci zo zreťazenia prvkov v zozname, ktorý je argumentom metódy **join**():

```
oddelovac = ""
zoznam = ["b","n","n"]
vysledok = oddelovac.join(zoznam) #spoji "b" "n" "n" pomocou oddelovaca "", teda "b"+""+"n"+""+"n"
print(vysledok)
```

```
bnn
```