Druhý test

Druhý test bude na cvičeniach v 11. týždni:

27.11. a 28.11.

Test bude prebiehať rovnakou formou ako prvý test, bude za 20 bodov. Bude z preberanej látky od začiatku semestra po 11. týždeň.

Druhý test

V prípade, že sa nemôžete zúčastniť testu (skúšky), **musíte postupovať** podľa Smernice dekana 4/2023:

1. V prípade neúčasti na vzdelávacej činnosti (ďalej neúčasti) podľa čl. 5 ods. 5 Študijného poriadku STU v Bratislave je **študent povinný najneskôr do 5 pracovných dní od jej ukončenia doručiť na Pedagogické oddelenie fakulty doklad preukazujúci dôvod neúčasti.** Pri nesplnení tejto povinnosti sa neúčasť pokladá za neospravedlnenú.

2. V prípade neúčasti počas termínu skúšky je **študent najneskôr do 5 pracovných dní od konania skúšky povinný informovať skúšajúceho o dôvode svojej neúčasti. Následne je povinný najneskôr do 5 pracovných dní od ukončenia neúčasti doručiť na Pedagogické oddelenie fakulty doklad preukazujúci dôvod neúčasti.** V prípade nesplnenia týchto povinností študent stráca nárok na náhradný termín skúšky.

Projekt

Na webe predmetu B-PROG1 bude 19.11. zverejnený 10-bodový projekt.

Vašou úlohou je implementovať simulátor stolovej hry *Teleporty* podľa zadania.

Zadanie si **poriadne a pozorne** prečítajte. Na jeho vypracovanie a odovzdanie máte čas do nedele 8.12., 23:59. Odovzdáva sa zdrojový kód Vášho riešenia (.py súbor/súbory) do Akademického informačného systému do príslušného miesta odovzdania.

Neskorší dátum odovzdania **nie je možný.**

V prípade akýchkoľvek nejasností a otázok k projektu sa obráťte na svojich cvičiacich.

PROG1: Prednáška 10

Zoznamy (lists)

*Časť druhá: Pokročilejšie **časti***

Zoznamy

Minulý týždeň sme sa bavili o zoznamoch, ako o usporiadanom dátovom type, v ktorom je možné ukladať rôzne hodnoty (heterogénny štruktúrovaný dátový typ).

Využitie zoznamov je veľmi široké, keďže umožňujú ukladať si v princípe ľubovoľný počet rôznych hodnôt (a rôznych typov hodnôt) počas behu programu, pričom neskôr je možné použiť tieto hodnoty na ďalšie spracovanie.

Ukázali sme si základné manipulácie so zoznamami (pridať prvok / odstrániť prvok). Typickou úlohou pri práci so zoznamom je zistenie, či sa v zozname nejaký prvok nachádza, koľkokrát sa tam nachádza, na akej pozícii sa tam nachádza, atď.

Dnes sa budeme baviť o zoznamoch viac, pričom si ukážeme niektoré špecifiká, ktoré platia pre prácu so zoznamami v jazyku Python.

Zoznamy

Uvažujme nasledovný program v jazyku Python:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
zoznam1 = ['a', 'b', 'c'] #zoznam 3 znakov: 'a', 'b', 'c'
zoznam2 = zoznam1         #vytvorime zoznam2, ktory bude obsahovat tie
                          #iste prvky ako zoznam1

print('zoznam1:',zoznam1) #vypiseme zoznam1 na obrazovku: ['a','b','c']
print('zoznam2:',zoznam2) #ocakavame vypis tiez ['a','b','c']

zoznam1.append('d')       #do zoznam1 pridame na koniec znak 'd'

print('zoznam1:',zoznam1) #ocakavame vypis ['a','b','c','d']
print('zoznam2:',zoznam2) #CO SA VYPISE TU?????
```

```
zoznam1: ['a', 'b', 'c']
zoznam2: ['a', 'b', 'c']
zoznam1: ['a', 'b', 'c', 'd']
zoznam2: ['a', 'b', 'c', 'd']
```

Zoznamy

Vidíme, že po tom, ako sme v predošlom programe pridali prvok ‘d‘ do zoznamu *zoznam1*, pridal sa tento prvok aj do zoznamu *zoznam2*. ---> Prečo???

Celé je to spôsobené tým, **ako vznikol** *zoznam2*, konkrétne príkaz:

zoznam2 = zoznam1

v jazyku Python **neurobí to**, že by v premennej *zoznam2* bola uložená kópia toho zoznamu, ktorý je v premennej *zoznam1*, ale je tam uložený **ten istý zoznam**, ktorý je v premennej *zoznam1*, teda doslova sú to tie isté hodnoty, ktoré sa nachádzajú niekde v pamäti počítača.

Na to, aby sme si aspoň čiastočne vysvetlili, prečo to tak je, si musíme povedať najprv niečo o tom, **ako sú reprezentované dáta v jazyku Python.**

Reprezentácia dát v jazyku Python

(Zdroj: https://docs.python.org/3/reference/datamodel.html)

Všetky dáta sú v jazyku Python reprezentované pomocou tzv. objektov alebo vzťahov medzi objektami.
(Samotný koncept objektu súvisí s tzv. objektovo-orientovaným programovaním, o ktorom sa dozviete viac na predmete B-OOP, Objektovo-orientované programovanie).

Každý objekt (reprezentácia dát v Pythone) má:

a) identitu – v podstate súvisí s adresou, kde v pamäti sú dáta uložené
b) typ – dáta môžu byť rôzneho typu (celé číslo, desatinné číslo, reťazec, None, bool)...
c) hodnotu – napríklad konkrétna hodnota čísla, reťazca, True/False, …

a) Ak chceme zistiť **identitu** hodnoty(objektu), môžeme použiť funkciu **id()**
b) Na zistenie typu hodnoty môžeme použiť funkciu **type()**

Reprezentácia dát v jazyku Python

Napríklad pri práci s celými číslami:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
cislo = 10          #vytvori v pamati objekt cele cislo s hodnotou 10. Zaroven vytvori prememnu
                    #"cislo", pomocou ktorej budeme s objektom pracovat.

print(id(cislo))    #id() vrati id "objektu", t.j. cislo suvisiace s adresou objektu v pamati

print(type(cislo)) #type() vrati typ "objektu"

print(cislo)        #samotna premenna len "ukazuje" na prislusny objekt, pomocou nej
                    #s nim vieme pracovat a napriklad vypisat jeho hodnotu na obrazovku
```

Ln: 2

```
140714479323864
<class 'int'>
10
```

Reprezentácia dát v jazyku Python

Napríklad pri práci s desatinnými číslami:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
cislo = 1.5          #vytvori v pamati objekt desatinne cislo s hodnotou 1.5. Zaroven vytvori prememnu
                     #"cislo", pomocou ktorej budeme s objektom pracovat.

print(id(cislo))     #id() vrati id "objektu", t.j. cislo suvisiace s adresou objektu v pamati

print(type(cislo)) #type() vrati typ "objektu"

print(cislo)         #samotna premenna len "ukazuje" na prislusny objekt, pomocou nej
                     #s nim vieme pracovat a napriklad vypisat jeho hodnotu na obrazovku
```

Ln: 1 Co

```
2379267856368
<class 'float'>
1.5
```

Reprezentácia dát v jazyku Python

Napríklad pri práci s reťazcami:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
ret = "ahoj"        #vytvori v pamati objekt retazec s hodnotou "ahoj". Zaroven vytvori prememnu
                    #ret, pomocou ktorej budeme s objektom pracovat.

print(id(ret))      #id() vrati id "objektu", t.j. cislo suvisiace s adresou objektu v pamati

print(type(ret))    #type() vrati typ "objektu"

print(ret)          #samotna premenna len "ukazuje" na prislusny objekt, pomocou nej
                    #s nim vieme pracovat a napriklad vypisat jeho hodnotu na obrazovku
```

```
2447932770368
<class 'str'>
ahoj
```

Reprezentácia dát v jazyku Python

Napríklad pri práci so zoznamami:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t = [1,2,3]       #vytvori v pamati objekt zoznam s hodnotou [1,2,3]. Zaroven vytvori prememnu
                  #t, pomocou ktorej budeme s objektom pracovat.

print(id(t))      #id() vrati id "objektu", t.j. cislo suvisiace s adresou objektu v pamati

print(type(t))    #type() vrati typ "objektu"

print(t)          #samotna premenna len "ukazuje" na prislusny objekt, pomocou nej
                  #s nim vieme pracovat a napriklad vypisat jeho hodnotu na obrazovku
```

```
1346731790592
<class 'list'>
[1, 2, 3]
```

Reprezentácia dát v jazyku Python

Každá hodnota v jazyku Python je teda reprezentovaná pomocou tzv. objektu, ktorý má:

a) svoje id (=svoju adresu v pamäti)
b) svoj typ
c) svoju konkrétnu hodnotu

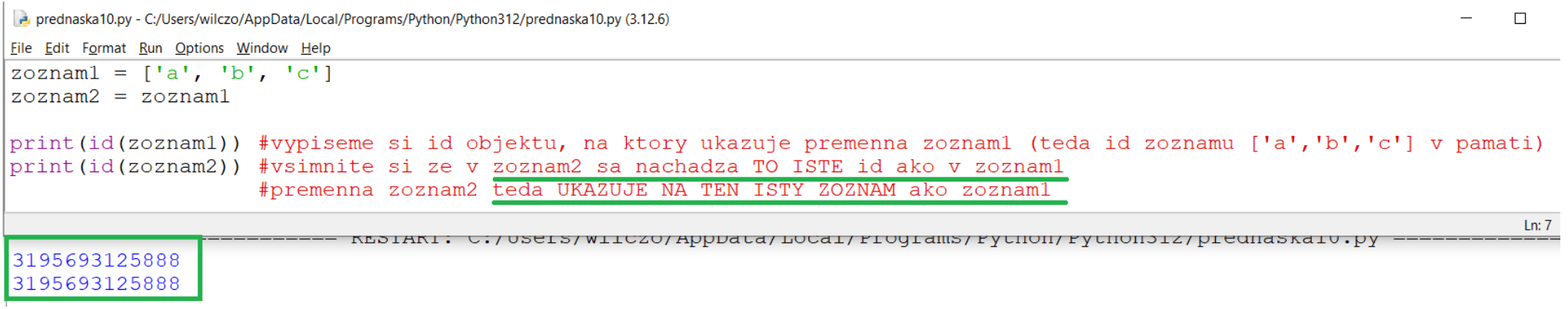
Premenné sú potom **len ukazovatele** (niekedy ich pri objektoch nazývame aj **referencie**) na danú hodnotu v pamäti, pomocou ktorých s danou hodnotou pracujeme - vypisujeme jej hodnotu na obrazovku, v prípade *mutable* (meniteľných) objektov ako reťazce meníme obsahy objektu a podobne…

Vráťme sa teraz k nášmu príkladu 2 zoznamov zo začiatku prednášky.

Reprezentácia dát v jazyku Python

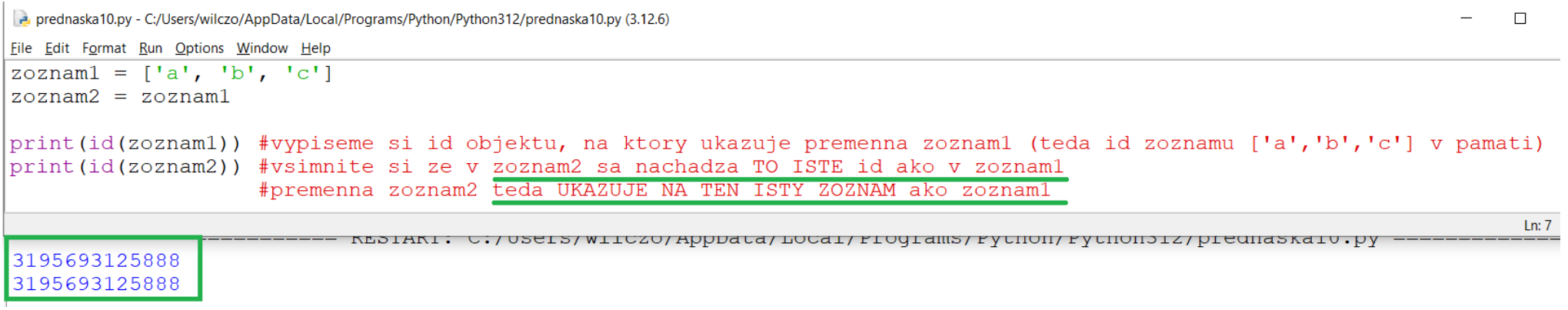
Znovu vytvorme zoznam [‘a‘, ‘b‘, ‘c‘] , priraďme ho do premennej *zoznam1* (povedali sme si, že toto priradenie vlastne vytvorí v pamäti objekt *zoznam* s hodnotami [‘a‘, ‘b‘, ‘c‘] a premenná *zoznam1* len tvorí referenciu na tento objekt.

Ak teraz vykonáme príkaz *zoznam2* = *zoznam1*, do premennej *zoznam2* sa vloží **tá istá referencia (odkaz)** na **ten istý objekt**, ako je v premennej *zoznam1*.



Reprezentácia dát v jazyku Python

Preto ak pomocou premennej *zoznam1* zmeníme hodnotu objektu v pamäti – napríklad tam pridáme prvok ‘d‘ na koniec zoznamu - keďže premenná *zoznam2* ukazuje na **ten istý objekt**, volanie *print(zoznam1)* a *print(zoznam2)* vypíšu dvakrát **ten istý zoznam!**



Reprezentácia dát v jazyku Python

Preto, keď pomocou premennej (ktorá ako teraz už vieme tvorí len akýsi odkaz na hodnotu zoznamu v pamäti), pridáme do zoznamu nový prvok, zmenu vidíme aj cez všetky ostatné premenné, ktoré ukazujú na ten istý objekt:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
zoznam1 = ['a', 'b', 'c']
zoznam2 = zoznam1

print(id(zoznam1)) #vypiseme si id objektu, na ktory ukazuje premenna zoznam1 (teda id zoznamu ['a','b','c'] v pamati)
print(id(zoznam2)) #vsimnite si ze v zoznam2 sa nachadza TO ISTE id ako v zoznam1
                   #premenna zoznam2 teda UKAZUJE NA TEN ISTY ZOZNAM ako zoznam1

zoznam1.append('d') #pridame znak 'd' do zoznamu v pamati, na ktory ukazuje premenna zoznam1 (a taktiez zoznam2)

print(id(zoznam1)) #vidime, ze po pridani prvku sa NEZMENILO id zoznamu (zmenila sa len HODNOTA daneho objektu)
print(id(zoznam2)) #to iste pre zoznam2
print(zoznam1)     #ze sa zmenila hodnota zoznamu vidime tak, ze vo vypise uz vidime aj pridany prvok 'd'
print(zoznam2)     #kedze zoznam2 ukazuje na ten isty objekt v pamati, aj tu vidime vo vypise 'd'
```

Ln: 13

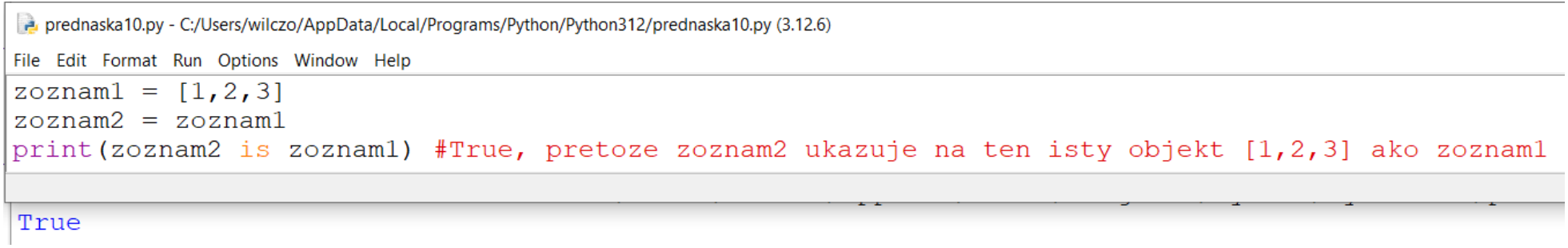
```
2223850803776
2223850803776
2223850803776
2223850803776
['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd']
```

Operátor **is**

Ak chceme zistiť, či 2 premenné ukazujú na ten istý objekt, môžeme použiť operátor **is**:

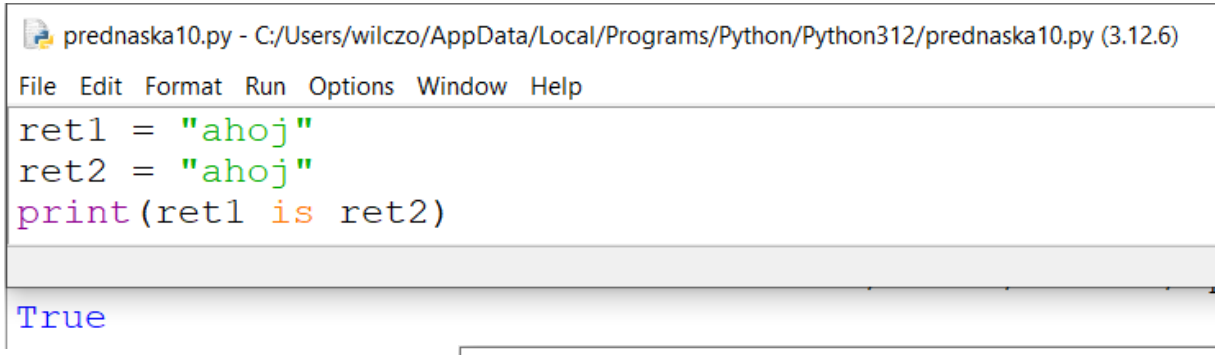
premenna1 **is** *premenna2*

je v Pythone booleovský výraz, ktorý nadobudne hodnotu True / False podľa toho, či premenná *premenna1* ukazuje na ten istý objekt ako *premenna2*:

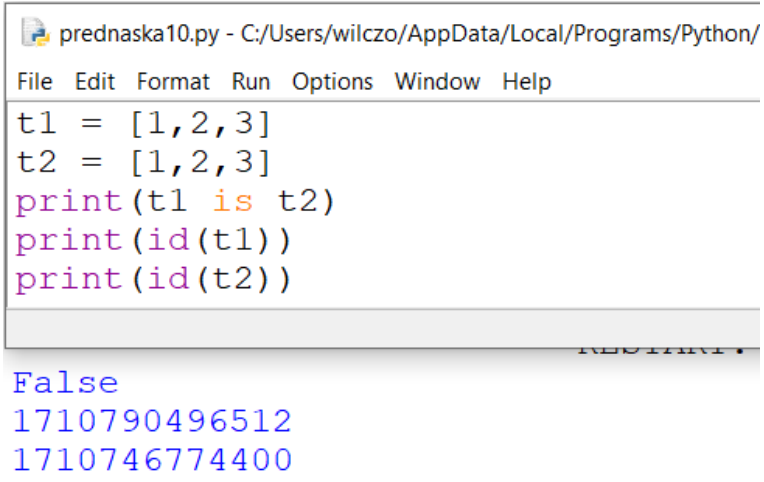


Reprezentácia dát v jazyku Python

Napríklad pri reťazcoch dokonca platí, že ak vytvoríme viacero premenných, ktoré ukazujú na ten istý reťazec zadaný „natvrdo“ v kóde, tak vlastne ukazujú na **ten istý objekt:**



Pri zoznamoch vo však **neplatí!**



Reprezentácia dát v jazyku Python

Všimnite si teda, že v jazyku Python môžu nastať tieto situácie:

a) 2 rôzne premenné môžu ukazovať na **ten istý zoznam** (z hľadiska objektu v pamäti)
b) 2 rôzne premenné môžu ukazovať na 2 rôzne zoznamy (2 rôzne objekty v pamäti) s **tými istými hodnotami!**

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1,2,3]
t2 = t1
print("t1:",t1)
print("t2:",t2)
print("t2 is t1:",t2 is t1) #True, pretoze t1 aj t2 ukazuju na ten isty objekt v pamati

t3 = [1,2,3]
print("t1:",t1)
print("t3:",t3)
print("t3 is t1:",t3 is t1) #False, pretoze t1 a t3 ukazuju na ROZNE objekty v pamati
                            #hoci hodnoty v oboch zoznamoch su TIE ISTE
```

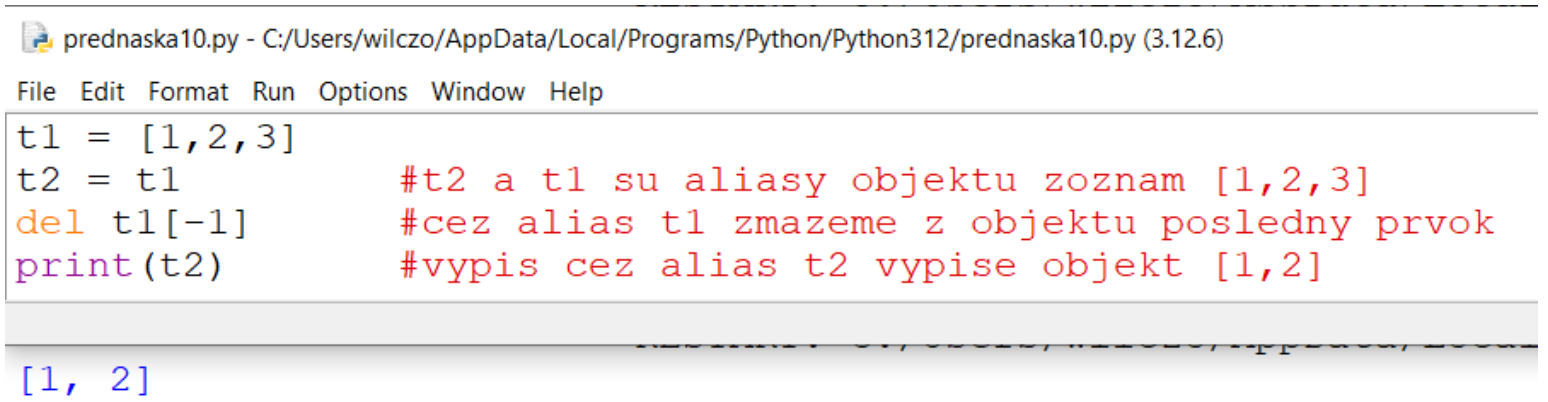
```
t1: [1, 2, 3]
t2: [1, 2, 3]
t2 is t1: True
t1: [1, 2, 3]
t3: [1, 2, 3]
t3 is t1: False
```

Aliasing

Ak 2 premenné ukazujú na ten istý objekt v pamäti, teda pre 2 premenné vráti operátor **is** hodnotu True, tak hovoríme, že dochádza k tzv. **aliasing-u**.

Teda obe premenné predstavujú tzv. **alias** pre daný objekt.

Ak je objekt s viacerými aliasmi meniteľný (mutable) – napríklad zoznam, tak je ho možné meniť cez ľubovoľný **alias** (ľubovoľnú premennú, ktorá naň ukazuje) a vykonaná zmena sa prejaví cez každý **alias** (napríklad pri vypísaní hodnoty objektu cez **print()**).



Porovnanie rovnosti zoznamov

Ak chceme zistiť, či sú 2 zoznamy rovnaké, používame na to operátor ==

Operátor == porovná 2 zoznamy **prvok za prvkom** a ak sú všetky prvky rovnaké, tak vráti *True*. Inak vráti *False.*

Prosím, **nepoužívajte na porovnávanie rovnosti hodnôt** operátor **is**!!!
(hoci k tomu zvádza anglický preklad „is“ do slovenčiny „je“).

Operátor **is** vám vráti *True* ak ide o **jeden a ten istý objekt**, nie o 2 objekty s **rovnakými hodnotami!!!!**

Porovnanie rovnosti zoznamov

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1,2,3]
t2 = t1
print("t1 is t2:", t1 is t2)   #vrati True pretoze ide o jeden a ten isty objekt
print("t1 == t2:", t1 == t2)   #vrati True pretoze zoznam na ktory ukazuje t1 obsahuje tie iste prvky ako zoznam t2
                               #CO JE UPLNE LOGICKE, PRETOZE OBE PREMENNE UKAZUJU NA TEN ISTY ZOZNAM, DUH!!!


t3 = [1,2,3]
print("t3 is t1:", t3 is t1)   #vrati False, pretoze t1 a t3 ukazuju na rozne objekty
print("t3 == t1:", t3 == t1)   #vrati True, pretoze oba zoznamy, na ktore ukazuju t3 a t1 obsahuju PRESNE TIE ISTE PRVKY
                               #ale z hladiska reprezentacie v pamati su to 2 rozne objekty (zoznamy)
```

Ln: 10 Co

```
t1 is t2: True
t1 == t2: True
t3 is t1: False
t3 == t1: True
```

Vytváranie kópií zoznamov

Povedali sme si teda, že ak premenná *zoznam1* ukazuje na nejaký zoznam, tak pomocou príkazu *zoznam2 = zoznam1* **nevytvoríme identickú kópiu**, ale len referenciu (odkaz) na ten istý zoznam.

V praxi však často potrebujeme vytvoriť **nezávislú kópiu** pôvodného zoznamu, teda nový objekt, ktorý má **tú istú hodnotu** (tie isté prvky) ako pôvodný zoznam, avšak **iné id().**

Pre tieto účely vieme využiť 2 typy kópií:

a) tzv. plytká kópia (*shallow copy)*
b) tzv. hlboká kópia *(deep copy)*

Vytváranie plytkej kópie zoznamu

Plytkú kópiu zoznamu vytvoríme (minimálne) dvomi rôznymi spôsobmi:

a) pomocou slice-notácie *list*[:] ktorá vráti plytkú kópiu zoznamu *list*, v ktorej sú ako prvky **referencie** na všetky prvky zoznamu *list*.
b) pomocou metódy *list***.copy()**, ktorá vráti plytkú kópiu zoznamu *list*, v ktorej sú ako prvky **referencie** na všetky prvky zoznamu *list*.

Výhodou plytkej kópie je, že vrátený zoznam je **nový** objekt, t.j. objekt s iným **id()** než pôvodný zoznam, z ktorého bola kópia robená.

V prípade, že chceme vytvoriť kópiu zoznamu, ktorý obsahuje ako prvky *nemeniteľné* objekty, napríklad číselné objekty (celé čísla, desatinné čísla) či znaky/reťazce, je vytvorenie plytkej kópie postačujúce.

Vytváranie plytkej kópie zoznamu

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1, 2, 3]
t2 = t1[:]          #pomocou slice notacie vznikne plytka kopia povodneho zoznamu
                    #ako novy objekt s vlastnym id
print("t1 = ",t1)  #vypis prvkov t1
print("t2 = ",t2)  #vidime, ze prvky t2 su tie iste ako t1
print(id(t1))       #id objektu na ktory ukazuje t1
print(id(t2))       #id objektu na ktory ukazuje t2, vidime, ze ide o objekt s inym id ako t1

t1.append(4)        #do zoznamu t1 vlozime cislo 4
print(t1)           #vidime, ze 4 bola vlozena do zoznamu na ktory ukazuje t1
print(t2)           #avsak NIE do zoznamu na ktory ukazuje t2
```

```
t1 =  [1, 2, 3]
t2 =  [1, 2, 3]
2697147666944
2697104141184
[1, 2, 3, 4]
[1, 2, 3]
```

Vytváranie plytkej kópie zoznamu

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1, 2, 3]
t2 = t1.copy()      #pomocou metody .copy() vznikne plytka kopia povodneho zoznamu
                    #ako novy objekt s vlastnym id
print("t1 = ",t1)  #vypis prvkov t1
print("t2 = ",t2)  #vidime, ze prvky t2 su tie iste ako t1
print(id(t1))       #id objektu na ktory ukazuje t1
print(id(t2))       #id objektu na ktory ukazuje t2, vidime, ze ide o objekt s inym id ako t1

t1.append(4)        #do zoznamu t1 vlozime cislo 4
print(t1)           #vidime, ze 4 bola vlozena do zoznamu na ktory ukazuje t1
print(t2)           #avsak NIE do zoznamu na ktory ukazuje t2
```

```
t1 =  [1, 2, 3]
t2 =  [1, 2, 3]
2374084022528
2374040365952
[1, 2, 3, 4]
[1, 2, 3]
```

Problémy s plytkou kópiou

Ak zoznam, ktorého plytkú kópiu vytvárame, obsahuje ako prvky ďalšie zoznamy (prípadne iné meniteľné objekty), potom vzniká problém – plytká kópia totiž ako prvky do kópie zoznamu vkladá len **referencie (odkazy)** na prvky pôvodného zoznamu.

Vieme sa o tom presvedčiť pomerne jednoducho:

Na obrázku vpravo vidíme, že hoci zoznamy *t1*,*t2* majú rôzne id, tak ich prvky majú id rovnaké!

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1, 2, 3]
t2 = t1.copy()

print("id(t1) = ",id(t1))
print("id prvkov zoznamu t1:")
for i in t1:
    print(id(i))
print("id(t2) = ",id(t2))
print("id prvkov zoznamu t2:")
for i in t2:
    print(id(i))
```

```
id(t1) =  2027638116608
id prvkov zoznamu t1:
140714479323576
140714479323608
140714479323640
id(t2) =  2027594197888
id prvkov zoznamu t2:
140714479323576
140714479323608
140714479323640
```

Problémy s plytkou kópiou

To môže predstavovať problém v situácii, že máme zoznam, ktorého prvky predstavujú ďalšie meniteľné objekty (napríklad ďalšie zoznamy):

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [[1,2], [3,4]]
t2 = t1.copy()

print("t1 = ",t1)
print("t2 = ",t2)
t1[0][0] = -1
t2[0].append(3)
print("t1 = ",t1)
print("t2 = ",t2)
```

1) Vidíme, že zmena prvku t1[0][0] sa dotkla aj prvku t2[0][0]
2) Rovnako pridanie prvku 3 do zoznamu t2[0] spôsobilo pridanie 3 aj to zoznamu t1[0]!

```
t1 =  [[1, 2], [3, 4]]
t2 =  [[1, 2], [3, 4]]
t1 =  [[-1, 2, 3], [3, 4]]
t2 =  [[-1, 2, 3], [3, 4]]
```

Problémy s plytkou kópiou

Ako sme si povedali, dôvod je ten, že prvky v plytkej kópii sú stále len referencie na prvky pôvodného zoznamu – všimnite si, že prvky v zozname *t2* majú **to isté id** ako prvky zoznamu *t1*:

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.

File Edit Format Run Options Window Help

```
t1 = [[1,2], [3,4]]
t2 = t1.copy()

print("t1 = ",t1)
print("id(t1) = ",id(t1))
for i in t1:
    print("Prvok ",i,"id = ",id(i))
print("t2 = ",t2)
print("id(t2) = ",id(t2))
for i in t2:
    print("Prvok ",i,"id = ",id(i))
```

```
t1 =  [[1, 2], [3, 4]]
id(t1) =  2093865832832
Prvok  [1, 2] id =  2093909496000
Prvok  [3, 4] id =  2093865839488
t2 =  [[1, 2], [3, 4]]
id(t2) =  2093909563776
Prvok  [1, 2] id =  2093909496000
Prvok  [3, 4] id =  2093865839488
```

Vytváranie hlbokej kópie zoznamu

Ak teda máme objekt, ktorý predstavuje napríklad zoznam takých hodnôt (objektov), ktoré znovu predstavujú meniteľné objekty (teda napríklad zoznam zoznamov) a chceme vytvoriť skutočne **nezávislú kópiu**, musíme vytvoriť tzv. **hlbokú kópiu (deep-copy).**

Na vytvorenie hlbokej kópie slúži v Pythone modul **copy**, v ktorom je k dispozícii metóda *copy*.**deepcopy*(obj)*** ktorá pre objekt *obj* vráti **hlbokú kópiu**.

T.j. napríklad pre zoznam vráti takú kópiu zoznamu, ktoré prvky už nie sú referencie na prvky pôvodného kopírovaného zoznamu, ale **samostatné nové objekty**, ktoré obsahujú tie isté hodnoty ako pôvodné kopírované objekty.

Vytváranie hlbokej kópie zoznamu

Vpravo vidíme použitie metódy *copy.***deepcopy()** ktorá vytvorí **hlbokú** kópiu zoznamu *t1*.

Všimnite si, že prvky zoznamu *t2* už majú všetky **iné id**, než aké je **id** prvkov zoznamu *t1*.

To znamená, že keď teraz budeme meniť zoznamy v zozname *t1*, už sa táto zmena **nedotkne** zoznamov v zozname *t2*.

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
import copy

t1 = [[1,2], [3,4]]
t2 = copy.deepcopy(t1)

print("t1 = ",t1)
print("id(t1) = ",id(t1))
for i in t1:
    print("Prvok ",i,"id = ",id(i))
print("t2 = ",t2)
print("id(t2) = ",id(t2))
for i in t2:
    print("Prvok ",i,"id = ",id(i))
```

```
t1 =  [[1, 2], [3, 4]]
id(t1) =  3163920617792
Prvok  [1, 2] id =  3163876961152
Prvok  [3, 4] id =  3163920552192
t2 =  [[1, 2], [3, 4]]
id(t2) =  3163920550912
Prvok  [1, 2] id =  3163920461376
Prvok  [3, 4] id =  3163920551936
```

Vytváranie hlbokej kópie zoznamu

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Pyth

File Edit Format Run Options Window Help

```
import copy

t1 = [[1,2], [3,4]]
t2 = copy.deepcopy(t1)

print("t1 = ",t1)
print("t2 = ",t2)
t1[0][0] = -1
t2[0].append(3)
print("t1 = ",t1)
print("t2 = ",t2)
```

```
t1 =  [[1, 2], [3, 4]]
t2 =  [[1, 2], [3, 4]]
t1 =  [[-1, 2], [3, 4]]
t2 =  [[1, 2, 3], [3, 4]]
```

1) Vidíme, že zmena prvku t1[0][0] sa už nedotkla prvku t2[0][0].
2) Rovnako pridanie prvku 3 do zoznamu t2[0] sa už nedotklo zoznamu t1[0]

Výseky (slice-notácia) = plytká kópia

Je potrebné si zapamätať, že v Pythone zoznamy, ktoré vznikajú pomocou slice-notácie (teda napríklad aj *t1*[:], ale aj rôzne ďalšie verzie, *t1[1:], t1[:-1], t1[::-1]* a podobne), **vždy** predstavujú plytkú kópiu zoznamu, na ktorý bola slice-notácia aplikovaná.

```
t1 = [[1,2], [3,4], [5,6]]
t2 = t1[::2] #vyberieme prvy a treti prvok z t1
print(id(t1), t1)
for i in t1:
    print(i,id(i))

print(id(t2), t2)
for i in t2:
    print(i,id(i))
```

```
2291210279296 [[1, 2], [3, 4], [5, 6]]
[1, 2] 2291210211520
[3, 4] 2291166423936
[5, 6] 2291166417280
2291210210368 [[1, 2], [5, 6]]
[1, 2] 2291210211520
[5, 6] 2291166417280
```

Pridávanie prvku do zoznamu

V Pythone je dôležité rozlišovať, ako prebieha pridávanie nového prvku do zoznamu:
a) metóda *list.***append***(x)* pridáva prvok *x* do existujúceho zoznamu *t1*
b) operácia *list* + [*x*] vytvorí nový zoznam, ktorý vráti

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1,2,3]
print(id(t1), t1)
t1.append(4)       #pridanie prvku cez metodu append prida do t1 novy prvok
print(id(t1), t1) #zmeni sa hodnota zoznamu, ale nie jeho id

t2 = [1,2,3]
print(id(t2), t2)
t2 = t2 + [4]      #pridanie prvku cez pomocou operatora + VYTVORI novy zoznam zo zoznamu t2 a prvku 4
print(id(t2), t2) #zmeni sa id zoznamu na ktory ukazuje premenna t2

print(t1 == t2)    #zoznamy su co do prvkov rovnake
```

```
1992096698624 [1, 2, 3]
1992096698624 [1, 2, 3, 4]
1992096697344 [1, 2, 3]
1992096697408 [1, 2, 3, 4]
True
```

Pridávanie prvku do zoznamu

Z toho vyplýva, že je dôležité rozlišovať, či dochádza k zmene pôvodného zoznamu (metóda **append**), alebo vytvoreniu nového zoznamu (operátor +).

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1, 2]
t2 = t1.append(3) #Kedze append meni povodny zoznam a nic nevracia
print(t1)         #Tu sa vypise zoznam [1,2,3] na ktory ukazuje t1
print(t2)         #Tu sa vypise None, co je navratova hodnota metody append
```

```
[1, 2, 3]
None
```

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
t1 = [1, 2]
t2 = t1 + [3] #Operator + vrati novy zretazeny zoznam
print(t1)     #Zoznam t1 ostal zachovany
print(t2)     #t2 obsahuje novy zoznam [1,2,3]
```

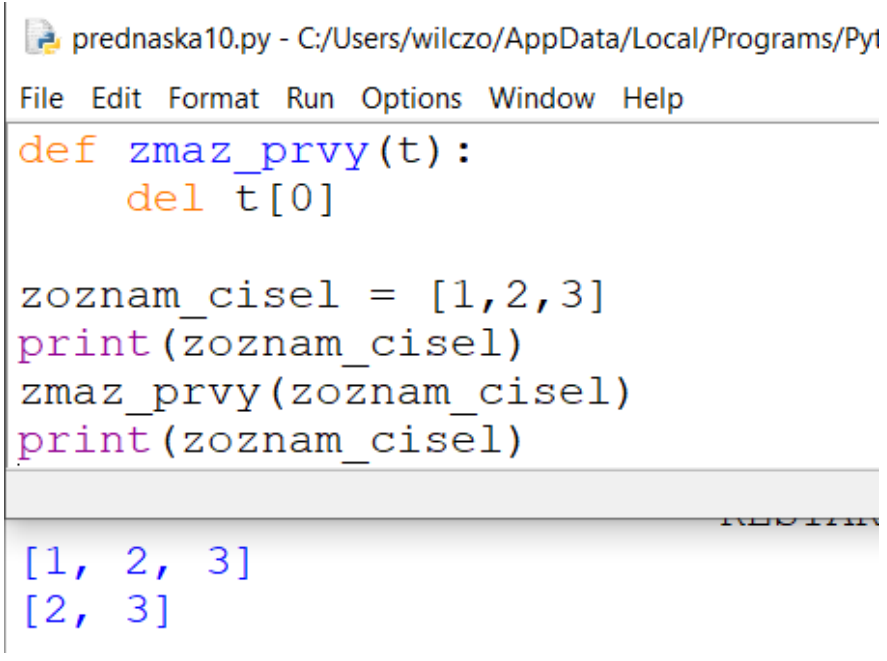
```
[1, 2]
[1, 2, 3]
```

Zoznamy ako argumenty funkcií

Keď sa odovzdá zoznam do volania funkcie ako argument, odovzdá sa **odkazom (referenciou).** To znamená, že funkcia dostane ako vstup len **referenciu** na daný zoznam.

Napríklad pri volaní funkcie *zmaz_prvy(t)* sa do parametra *t* uloží len **referencia** na *zoznam_cisel* (teda *t* a *zoznam_cisel* sa správajú ako **aliasy**).

Ak teda vo funkcii *zmaz_prvy(t)* dôjde k odstráneniu prvého prvku zoznamu, na ktorý ukazuje premenná *t*, cez **del** t[0], potom sa odstránenie prvého prvku týka zoznamu, ktorý bol argumentom funkcie - teda *zoznam_cisel.* Dôjde teda k **ZMENE** zoznamu *zoznam_cisel.*



Zoznamy ako argumenty funkcií

POZOR!!! Ako sme spomínali, výseky zo zoznamu (slice-notácia) vytvárajú nové zoznamy!

```
def zmaz_prvy_nespravne(t):
    t = t[1:]

zoznam_cisel = [1,2,3]
print(zoznam_cisel)
zmaz_prvy_nespravne(zoznam_cisel)
print(zoznam_cisel)
```

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

1) Argumentom funkcie bola premenná zoznam_cisel, teda parameter t predstavuje referenciu na zoznam, na ktorý ukazuje premenná zoznam_cisel
2) V tele funkcie t[1:] znamená, že sa pomocou slice-notácie vyrobí NOVÝ zoznam zo zoznamu t, v ktorom nebude prvý prvok
3) Zároveň sa referencia na tento nový zoznam vloží do premennej t
4) Problém je, že sa táto zmena neprejaví navonok, pretože nikde nedošlo k zmene zoznamu zoznam_cisel.
5) Navyše, premenná t je lokálnou premennou funkcie a po skončení volania funkcie sa stratí jej hodnota - teda zoznam, na ktorý ukazuje.

Zoznamy ako argumenty funkcií

Alternatívnou predošlej úlohy, teda využitia výseku zo zoznamu na vytvorenie verzie zoznamu *zoznam_cisel* bez prvého prvku by bolo, keby sa **vráti** referencia *t* z funkcie ako návratová hodnota. POZOR! Všimnite si, že táto verzia funkcie **NEZMENÍ** pôvodný zoznam *zoznam_cisel.*

```
def zmaz_prvy_v2(t):
    t = t[1:]
    return t

zoznam_cisel = [1,2,3]
print(zoznam_cisel)
novy_zoznam = zmaz_prvy_v2(zoznam_cisel)
print(zoznam_cisel)
print(novy_zoznam)
```

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

1) Argumentom funkcie pri volaní bola premenná zoznam_cisel, ktorá obsahuje referenciu na zoznam [1,2,3]
2) Rovnako parameter t funkcie bude predstavovať po volaní funkcie referenciu na ten istý zoznam.
3) Príkaz t = t[1:] z tohto zoznamu vyrobí NOVÝ zoznam, obsahujúci len prvky [2,3] a referenciu na tento zoznam vloží do premennej t.
4) V porovnaní s predošlým príkladom sa však teraz táto nová referencia ODOVZDÁ ako návratová hodnota z volania funkcie zmaz_prvy_v2.
5) Ak sa teda táto referencia vloží do premennej novy_zoznam, tak v programe bude po volaní funkcie existovať aj pôvodný zoznam zoznam_cisel so zoznamom [1,2,3] (tento zoznam ostal nezmenený), a aj nový zoznam [2,3] v premennej novy_zoznam.

Zoznamy ako argumenty funkcií

Na teste / skúške si musíte vždy dávať dobrý pozor, ako je zadaná úloha!

Teda, ak máte definovať funkciu, ktorej parametrom je zoznam, treba dávať dobrý pozor:

a) či má funkcia vrátiť **nový zoznam**, ktorý je modifikáciou vstupného zoznamu a vstupný zoznam má **zostať nezmenený,** teda po volaní funkcie je zoznam, ktorý tvoril argument volania, **nezmenený,**

b) alebo má funkcia **priamo modifikovať vstupný zoznam,** teda po volaní funkcie je zoznam, ktorý tvoril argument volania, **zmenený.**

Generovanie náhodných čísiel

- Ak potrebujeme v programe generovať náhodné čísla, môžeme na to použiť modul **random**.
- Modul **random** poskytuje veľa rôznych metód pre generovanie náhodných čísiel, viac na https://docs.python.org/3/library/random.html
- Napríklad, ak potrebujeme generovať náhodné celé číslo z množiny {*a*, *a*+1, …, *b*}, môžeme na to použiť metódu *random*.**randint**(*a*,*b*), ktorá vráti náhodné číslo od *a* po *b*, vrátane.
- Čiže napríklad volanie *random*.**randint**(1,6) vráti náhodné číslo z množiny {1,2,3,4,5,6} (simuluje hod kockou).

Vstavané funkcie pre iterovateľné objekty

- Vstavaná funkcia **max()** vráti **najväčšiu** hodnotu vo vstupnom iterovateľnom objekte (napr. zozname).
- Vstavaná funkcia **min()** vráti **najmenšiu** hodnotu vo vstupnom iterovateľnom objekte (napr. zozname).
- Vstavaná funkcia **sum()** vráti **súčet** hodnôt vo vstupnom iterovateľnom objekte (napr. zozname).
- Vstavaná funkcia **sorted() vráti nový objekt,** ktorý predstavuje **utriedenú verziu** (vo vzostupnom poradí, od min po max) vstupného iterovateľného objektu (napr. zoznamu). Pôvodný vstup zostane **neutriedený!**

Vstavané funkcie pre iterovateľné objekty

```
t = [10, -4, 2, 0, 3, -2]
najvacsi = max(t)
najmensi = min(t)
suma = sum(t)
t_utriedeny = sorted(t)
print("t = ",t)
print("max(t) = ",najvacsi)
print("min(t) = ",najmensi)
print("sum(t) = ",suma)
print("sorted(t) = ",t_utriedeny)
```

```
t =  [10, -4, 2, 0, 3, -2]
max(t) =  10
min(t) =  -4
sum(t) =  9
sorted(t) =  [-4, -2, 0, 2, 3, 10]
```

Funkcia **isinstance**()

Ako sme si spomínali, objekty v jazyku Python uchovávajú dáta, pričom dáta majú svoje:
a) id
b) typ
c) hodnotu

Ak potrebujeme v Pythone otestovať, akého typu sú dáta, môžeme na to použiť vstavanú funkciu **isinstance***(object, classinfo)*, ktorá vráti *True / False* podľa toho, či je objekt *object* inštanciou triedy *classinfo*.

Keďže na predmete B-PROG1 sa objektom ani triedam nevenujeme, v jednoduchosti si len povedzme, že ako hodnotu argumentu *classinfo* vieme použiť napríklad:
- *int* pre testovanie, či je *object* celé číslo
- *float* pre testovanie, či je *object* desatinné číslo
- *str* pre testovanie, či je *object* reťazec
- *list* pre testovanie, či je *object* zoznam

Funkcia **isinstance**()

prednaska10.py - C:/Users/wilczo/AppData/Local/Programs/Python/Python312/prednaska10.py (3.12.6)

File Edit Format Run Options Window Help

```
def faktorial(n):
    if not isinstance(n, int):
        print("Faktorial je definovany len pre cele cisla!")
        return None
    elif n < 0:
        print("Faktorial nie je definovany pre zaporne cisla!")
        return None
    elif n == 0:
        return 1
    else:
        return n*faktorial(n-1)


print(faktorial(5))     #vrati a vypise 5! = 120
print(faktorial(-10))   #vypise chybu (zaporny vstup) a vrati None
print(faktorial(3.14)) #vypise chybu (vstup nie je cele cislo) a vrati None
```

```
120
Faktorial nie je definovany pre zaporne cisla!
None
Faktorial je definovany len pre cele cisla!
None
```