

Druhy test

Druhy **test** bude na cviceniach v 11. tyzdni.

O teste:

- test bude za **20 bodov!**
- podobne ulohy ako na cviceniach.
- na papier.
- doneste si vlastne papiere na riesenie uloh na necisto!
- z celej latky prebranej do 11. tyzdna.

Priebeh testu

- Na vypracovanie testu budete mať dost času!
- Neponahajte sa, pozorne si precitajte zadanie, a ulohu najprv vyrieste na necisto.
- Po skončení testu môžete z cvicenia odísť, ale môžete aj na cviceni ostáť a riešiť ulohy z minulých cviceni alebo z testu alebo pracovať na projekte.

Kratkodoby plan

T10 (aktualny tyzden):

- Prednaska: Zoznamy (pokrocilejsie casti)
- Cvicenia: ulohy na zoznamy

T11:

- Prednaska: preriesim ulohy z desiateho tyzdna, nebudem preberat novu latku
- Cvicenia: test

T12:

- Prednaska: zaverecne informacie
- Cvicenia:
 - moznost nahliadnut do opravenych testov
 - posledna sanca konzultovat s cviciacim projekt

PROG1: Prednáška 10

Zoznamy (lists)

Cast druha: Pokročilejšie casti

Domaca uloha

1. Precitajte si sekcie 10.10 – 10.14 v knihe.
2. Vyrieste ulohy z Cvicenia 10.

Příklad s hledáním maxima

Pozrite si skript

`prednaska10maximum.py`

Objekty

Zdroj: <https://docs.python.org/3/reference/datamodel.html>

Data su v pythone reprezentovane pomocou **objektov**.

Kazdy objekt ma:

- **Identitu** (adresa objektu v pamati pocitaca)
- **Hodnotu** (value)
- **Typ**

Typy objektov s nemenitelnou hodnotou:

integer, float, string, ...

Typy objektov s menitelnou hodnotou: zoznamy, ...

Objekty

Priklad:

```
>>> a=[1,2,3]
>>> a
[1, 2, 3]
>>> type(a)
<class 'list'>
>>> id(a)
47068024
```

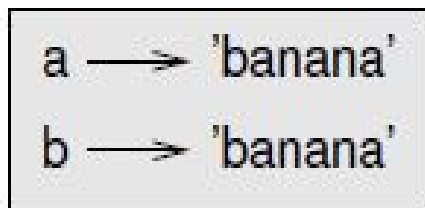
Ked urobime priradenie `a=[1,2,3]`, premenna `a` odkazuje na objekt ktoreho hodnota je `[1,2,3]`, ktoreho typ je `list` a ktoreho identita je `47068024`. (funkcia `id()` vrati identitu objektu)

Objekty

```
>>> a='banana'  
>>> b='banana'
```

Odkazuju premenne a, b na ten isty objekt?

Mohla by nastat jedna z tychto situacii:



Objekty

```
>>> a='banana'  
>>> b='banana'
```

Odkazuju premenne a, b na ten isty objekt?

Identitu objektu mozeme zistit pomocou funkcie id().

```
>>> a='banana'  
>>> b='banana'  
>>> id(a)  
47113856  
>>> id(b)  
47113856
```

Cize situacia vyzera takto:



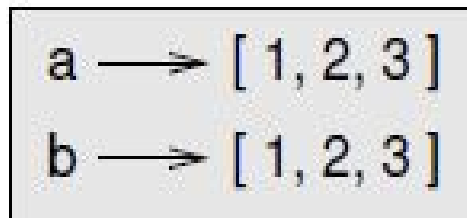
Objekty

```
>>> a=[1,2,3]
```

```
>>> b=[1,2,3]
```

Odkazuju premenne a, b na ten isty objekt?

Mohla by nastat jedna z tychto situacii:



Objekty

```
>>> a=[1, 2, 3]
```

```
>>> b=[1, 2, 3]
```

Odkazuju premenne a, b na ten isty objekt?

```
>>> a=[1, 2, 3]
```

```
>>> b=[1, 2, 3]
```

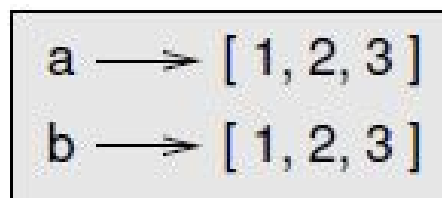
```
>>> id(a)
```

```
3853488
```

```
>>> id(b)
```

```
47068024
```

Cize situacia je takato:



Premenne a,b odkazuju na **rozne objekty s rovnakymi hodnotami!**

Objekty

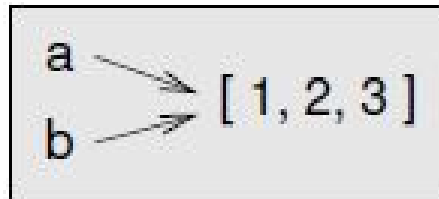
Zhodnost objektov mozeme testovat aj pomocou operatora `is`.

```
>>> a='banana'  
>>> b='banana'  
>>> a is b  
True  
>>> a=[1, 2, 3]  
>>> b=[1, 2, 3]  
>>> a is b  
False
```

Aliasing

```
>>> a = [1, 2, 3]
>>> b = a
>>> b is a
True
```

Tu nastava pripad:



Dve rozne premenne odkazuju na ten isty objekt – tomuto hovorime **aliasing**.

Aliasingu je dobre sa vyhybat. Casto sposobuje chyby v programe.

Aliasing a chyby

```
>>> a = [1, 2, 3]
```

```
>>> b = a
```

Aliasingu je dobre sa vyhybat. Casto sposobuje chyby v programe.

Napriklad ak zmenime 0-ty prvok v zozname b, automaticky sa zmeni aj 0-ty prvok v zozname a (na co je lahke zabudnut).

```
>>> a=[1, 2, 3]
```

```
>>> b=a
```

```
>>> b[0]=5
```

```
>>> a
```

```
[5, 2, 3]
```

Kopie zoznamov

Ak chceme urobiť kópiu zoznamu, je lepšie vyhnúť sa aliasingu a radšej vytvoriť nový zoznam (teda vytvoriť nový objekt typu zoznam) s rovnakou hodnotou. To môžeme urobiť nasledovne:

```
>>> a=[1,2,3]
>>> b=a[:]
```

Potom máme:

```
>>> a=[1,2,3]
>>> b=a[:]
>>> b
[1, 2, 3]
>>> a is b
False
>>> b[0]=5
>>> a
[1, 2, 3]
```


Copy vs deepcopy

Pozrite si skript

`prednaska10deepcopy.py`

Vytvorenie noveho zoznamu VS zmena povodneho

Je dolezite rozlisovat, ci operacia vytvara nový zoznam, alebo meni povodny zoznam!

Priklad: Metoda append meni povodny zoznam.

```
>>> t1 = [1, 2]
>>> t2 = t1.append(3)
>>> t1
[1, 2, 3]
>>> t2
None
```

Vytvorenie noveho zoznamu VS zmena povodneho

Je dolezite rozlisovat, ci operacia vytvara nový zoznam,
alebo meni povodny zoznam!

Priklad 2: Operacia + vytvara nový zoznam.

```
>>> t1=[1, 2, 3]
>>> t3=t1+[4]
>>> t1
[1, 2, 3]
>>> t3
[1, 2, 3, 4]
```

Zoznamy ako argumenty funkcií

Doposiaľ sme vytvárali funkcie, ktoré pre vstupné zoznamy vracali nové zoznamy alebo nejaké ine hodnoty.

Príklad:

```
def tail(t):  
    return t[1:]
```

Príklad použitia:

```
>>> letters = ['a', 'b', 'c']  
>>> rest = tail(letters)  
>>> rest  
['b', 'c']
```

Zoznamy ako argumenty funkcií

Doposiaľ sme vytvárali funkcie, ktoré pre vstupné zoznamy vracali nové zoznamy alebo nejaké ine hodnoty.

Mozeme ale vytvárať aj funkcie, ktoré budú priamo meniť vstupné zoznamy!

Zoznamy ako argumenty funkcii

Co vypise tento skript?

```
def delete_head(t):  
    del t[0]
```

```
cisla=[1,2,3]  
delete_head(cisla)  
print(cisla)
```

Keď sa zavola funkcia s argumentom `cisla`, do funkcie sa dostane odkaz na samotný objekt, na ktorý premenná `cisla` odkazuje. Ak funkcia tento objekt pozmeni, bude premenná `cisla` po vykonaní funkcie odkazovať na **zmenený** objekt.

Zoznamy ako argumenty funkcii

Zly priklad:

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

Tato funkcia nevymaze prvý prvok zo zoznamu! (Operácia `t[1:]` vytvorí nový zoznam. Prikaz `t=t[1:0]` vytvorí novú lokálnu premennú `t` a tá bude odkazovať na tento nový zoznam)

```
>>> t4 = [1, 2, 3]  
>>> bad_delete_head(t4)  
>>> t4  
[1, 2, 3]
```

Zoznamy ako argumenty funkcií

Zlý príklad:

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

Tato funkcia nevymaze prvý prvok zo zoznamu! (Operácia `t[1:]` vytvorí nový zoznam. Príkaz `t=t[1:]` vytvorí novú lokálnu premennú `t` a tá bude odkazovať na tento nový zoznam)

```
>>> t4 = [1, 2, 3]  
>>> bad_delete_head(t4)  
>>> t4  
[1, 2, 3]
```

POINTA: Je dôležité rozlišovať, či operácia **vytvára nový zoznam, alebo mení pôvodný zoznam!**