PROG1: Prednáška 10

Zoznamy (lists)

Cast druha: ***Pokrocilejsie casti***

Dalsi test

Najblizsi test bude na cviceniach buduci tyzdni .

O teste:

- test bude za **13 bodov**!
- podobne ulohy ako na cviceniach.
- na papier.
- z celej latky prebranej do toho tyzdna.
- Doneste si vlastny papier!

Buduci tyzden budem na prednaske riesit priklady z cviceni.

Domaca uloha

1. Precitajte si sekcie 10.10 – 10.14 v knihe.

2. Vyrieste ulohy z Cvicenia 10.

Objekty

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

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

Kazdy objekt ma:
- **Identitu** (o identite mozeme uvazovat ako o adrese 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
```

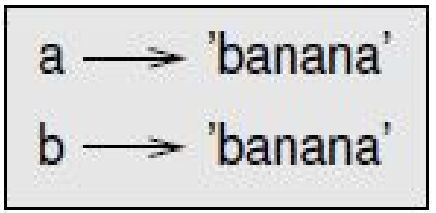
Ked urobime priradenie a=[1,2,3], premenna a odkazuje na objekt ktoreho hodnota je [1,2,3], ktoreho typ je list a ktoreho identita je 47068024. (prikaz id() vrati identitu objektu)

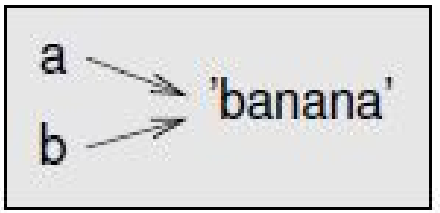
Objekty

```
>>> 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
```

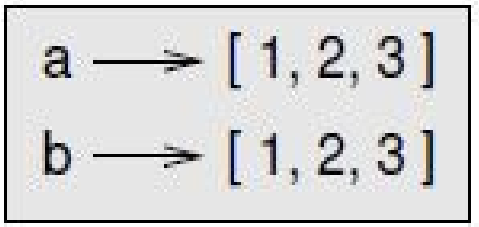
Cize situacia vyzera takto:

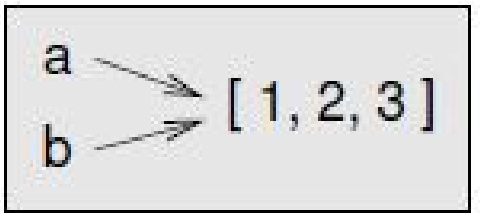
Objekty

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

Odkazuju premenne a, b na ten isty objekt?

Mohla by nastat jedna z tychto situacii:





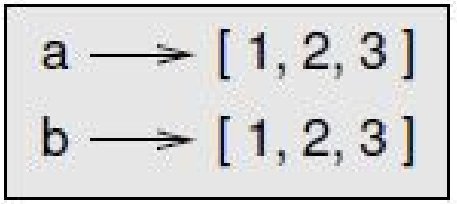
Objekty

```
>>> 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

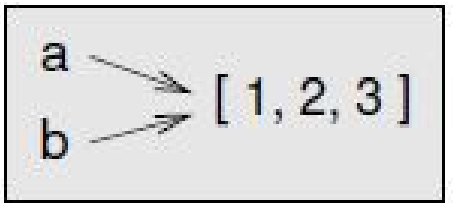
Zhodnost objektov mozeme testovat aj pomocou operatora is.

```
>>> 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 urobit kopiu zoznamu, je lepsie vyhnut sa aliasingu a radsej vytvorit novy zoznam (teda vytvorit novy objekt typu zoznam) s rovnakou hodnotou. To mozeme urobit nasledovne:

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

Potom mame:

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

Zoznamy ako argumenty funkcii

Doposial sme vytvarali funkcie, ktore pre vstupne zoznamy vracali nove zoznamy alebo nejake ine hodnoty.

Priklad:

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

Priklad pouzitia:

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

Zoznamy ako argumenty funkcii

Doposial sme vytvarali funkcie, ktore pre vstupne zoznamy vracali nove zoznamy alebo nejake ine hodnoty.

Mozeme ale vytvarat aj funkcie, ktore budu priamo menit vstupne zoznamy!

Zoznamy ako argumenty funkcii

Co vytlaci tento skript?

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

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

Ked sa zavola funkcia s argumentom cisla, do funkcie sa dostane odkaz na samotny objekt, na ktory premenna cisla odkazuje. Ak funkcia tento objekt pozmeni, bude premenna cisla po vykonani funkcie odkazovat na **zmeneny** objekt.

Zoznamy ako argumenty funkcii

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

Priklad: Metoda append meni povodny zoznam.

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

Zoznamy ako argumenty funkcii

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

Priklad 2: Operacia + vytvara novy zoznam.

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

Zoznamy ako argumenty funkcii

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

Priklad 3:

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

Tato funkcia nevymaze prvy prvok zo zoznamu! (Pretoze operacia t[1:] vytvori novy zoznam.)

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

Zoznamy ako argumenty funkcii

Priklad 3:

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

Tato funkcia nevymaze prvy prvok zo zoznamu! (Pretoze operacia t[1:] vytvori novy zoznam.)

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

POINTA: Pri citani dokumentacie k operaciam na zoznamoch si vsimajte, ci operacia vytvara novy zoznam, alebo meni povodny zoznam!