
pyladies
Poland

PyTraning2.0 #4:
Co by było gdyby Python miał brodę? Czyli listy (do?) św. Mikołaja.

Listy, sety, tuple

Zadanie 1

Jakie typy danych tu widzimy?

Dzieci: Ania, Marcin, Sebastian, Weronika, Maks, Zbyszek, Marysia, Kasia

Prezenty: Klocki Lego, Gra planszowa, Książka, Ciepłe skarpety, Tablet, Sweter, Sweter, Rózga

Miasta: Poznań, Kraków, Bielsko Biała

Liczba prezentów: 5, 2, 1

Czy grzeczne? True, False, True, True, False, True, True, True

Typy danych

Typy proste

- Całkowity (int)
- Zmiennopozycyjny (float)
- Tesktowy (str)
- Logiczny (bool)
- Zespolony (complex)

Typy strukturalne

- Lista (list)
- Krotka (tuple)
- Zbiór (set)
- Słownik, tabela (dict)

Czas na reklamę

Czas na reklamę

```
>>> imie = 'Ania'
>>> dir(imie)
['capitalize', 'center', 'count', 'endswith', 'expandtabs', 'find', 'index',
'isdigit', 'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'replace', 'rfind', 'rindex', 'rjust', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper']
>>> print(imie.lower.__doc__)
S.lower()
string Return a copy of the string S converted to lowercase.
>>> help(imie)
no Python documentation found for 'Ania'
>>> print(type(imie))
<class 'str'>
>>> help(str)
```

Do pracy!

Listy i krotki

- uporządkowane sekwencje obiektów
- mogą przechowywać obiekty dowolnego typu
- tworzenie list/krotek polega na podaniu zestawu oddzielonych przecinkami obiektów zamkniętych w nawiasy kwadratowe/zwykłe


```
>>> prezenty = ['Klocki Lego']
>>> dzieci = ('Ania',)
```

Listy

- Mutowalne, zmienne – można usuwać i dodawać elementy
- Iterowalna - zaczyna się od 0 -> pierwszym elementem niepustej listy o nazwie li jest zawsze li[0] – można przechodzić po jej elementach
- mogą przechowywać dowolny obiekt i dynamicznie dodawać nowe pozycje

Krotki

- niezmienne - zawartość określamy tylko podczas jej tworzenia, potem nie możemy już je
- zachowują integralność danych
- są wydajniejsze (działają szybciej i zajmują mniej miejsca w pamięci), jeśli nie ma konieczności modyfikowania danej kolekcji obiektów, zaleca sie ich użycie zamiast list

Krotki

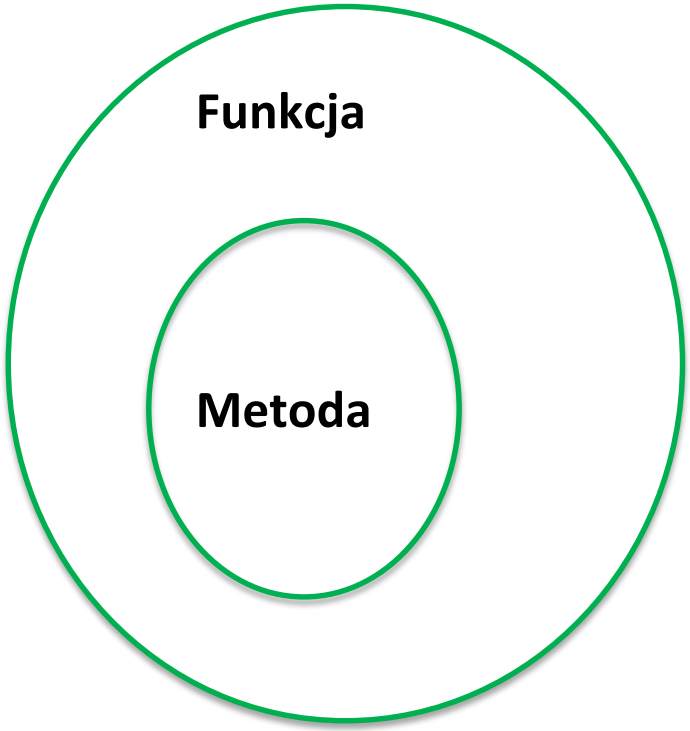
- Krotki definiujemy w identyczny sposób jak listę, lecz z jednym wyjątkiem -- zbiór elementów jest ograniczony w nawiasach okrągłych, zamiast w kwadratowych
- Podobnie jak w listach, elementy w krotce mają określony porządek. Są one indeksowane od `0`, więc pierwszym elementem w niepustej krotce jest zawsze `t[0]`
- Ujemne indeksy idą od końca krotki, tak samo jak w listach
- Krotki także można wycinać - kiedy wycinamy listę, dostajemy nową listę - gdy wycinamy krotkę dostajemy nową krotkę

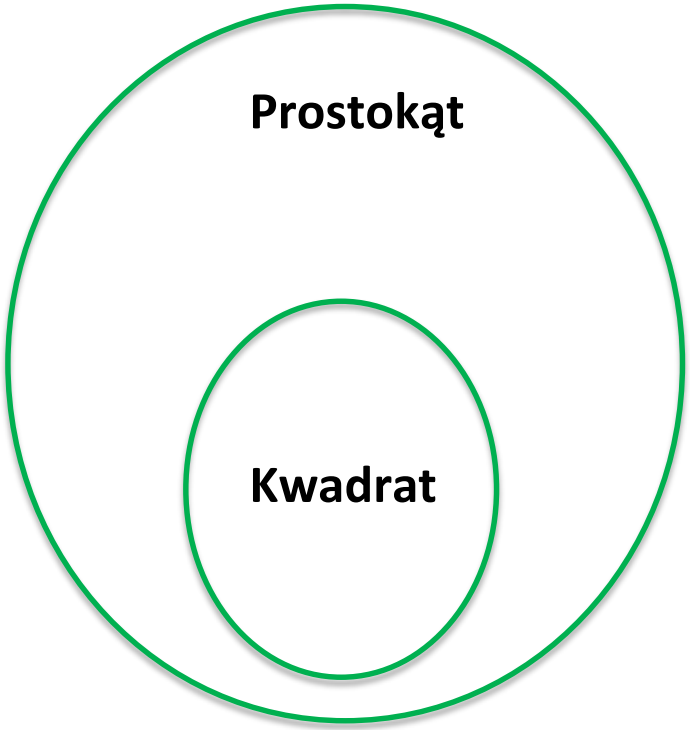
Sety

- nieuporządkowane kolekcje unikatowych elementów

```
>>> a = [1,1,1]
>>> b = set(a)
>>> b
set([1])
```

Metody i funkcje - przypomnienie





Metody i funkcje - Stringi

Modyfikujące:

capitalize()

count(str)

find(str)

lower()

replace(old, new)

rfind()

strip([chars])

swapcase()

title()

upper()

Sprawdzające:

endswith(str)

isalnum()

isalpha()

isdigit()

islower()

isnumeric()

isspace()

istitle()

isupper()

startswith(str)

Metody i funkcje - Stringi

Modyfikujące:

capitalize()
count(str)
find(str)
lower()
replace(old, new)
rfind(str)
strip([chars])
swapcase()
title()
upper()

Sprawdzające:

endswith(str))
isalnum()
isalpha()
isdigit()
islower()
isnumeric()
isspace()
istitle()
isupper()
startswith(str)

```
>>> text = 'ala ma kota'
>>> text.upper()
>>> text.replace('l', 'ni')
>>> text.strip('a')
```

```
>>> text = 'ala ma kota'
>>> text.startswith('ala')
>>> text.endswith('ala')
>>> text.isalpha()
```

Co jeszcze można zrobić ze stringami

Oprócz wywoływania metod i funkcji, łączenia (+), czy mnożenia przez liczby, możemy jeszcze je **ciąć** na różne sposoby.

Ważne: komputery **liczą od zera**!

```
>>> text = “ala ma kota”
>>> text[0]          - string[int]
>>> text[2:]         - string[int:]
>>> text[:5]         - string[:int]
>>> text[3:7]        - string[int:int]
>>> text[::2]        - stirng[::int]
>>> text[::-1]       - stirng[::int]
```

Metody i funkcje - Listy

```
append()
count()
extend()
index()
insert()
pop()
remove()
reverse()
sort()
```

```
len(list)
','.join(list)
sorted(list)
tuple(list)
```

Metody i funkcje - Krotki

count()

index()

len(tuple)

','.join(tuple)

sorted(tuple)

list(tuple)

- Nie można dodawać elementów do krotki. Krotki nie posiadają metod typu `append`, czy też `extend`.
- Nie można usuwać elementów z krotki. Nie posiadają one ani metody `remove` ani metody `pop`
- Można wyszukiwać elementy w krotce wykorzystując metodę `index`.
- Można wykorzystać operator `in`, aby sprawdzić, czy krotka zawiera dany element

Co jeszcze można zrobić z listami i krotkami

- wywoływać metody i funkcję
- łączyć (+)
- mnożyć przez liczby
- **ciąć** na różne sposoby

Ważne: komputery **liczą od zera**!

```
>>> li = [1,2,3, [], 'Ala ma kota', ('A', 'A', 'A'),
'klocki Lego', 'Poznan']
>>> li[0]                          - list[int]
>>> li[2:]                         - list[int:]
>>> li[:5]                         - list[:int]
>>> li[3:7]                        - list[int:int]
>>> li[::2]                        - list[::int]
>>> li[::-1]                       - list[::int]
```

Metody i funkcje - Sety

add()
clear()
copy()
difference()
difference_update()
discard()
intersection()
intersection_update()
isdisjoint()
issubset()
issuperset()
pop()
remove()
symmetric_difference()
symmetric_difference_update()
union()
update()

Zadanie 2

Umieść wszystko w listach i krotkach

Dzieci: Ania, Marcin, Sebastian, Weronika, Maks, Zbyszek, Marysia, Kasia

Prezenty: Klocki Lego, Gra planszowa, Książka, Ciepłe skarpety, Tablet, Sweter, Sweter, Rózga

Liczba prezentów: 5, 2, 1

```
>>> dzieci = ('Ania', 'Marcin',
'Sebastian', 'Weronika', 'Maks', 'Zbyszek',
'Marysia', 'Kasia')
>>> prezenty = ['Klocki Lego', 'Gra
planszowa', 'Ksiazka', 'Cieple skarpety',
'Tablet', 'Sweter', 'Sweter', 'Rozga']
>>> co_komu = [dzieci, prezenty]
#zagnieżdżenie obiektów
>>> print(co_komu)
?????
>>> miasta = []; czy_grzeczne = ()
>>> print(type(miasta), type(czy_grzeczne))
?????
```

```
>>> co_komu = [dzieci, prezenty]
#zagnieżdżenie obiektów
>>> print(co_komu)
[('Ania', 'Marcin', 'Sebastian',
'Weronika', 'Maks', 'Zbyszek', 'Marysia',
'Kasia'), ['Klocki Lego', 'Gra planszowa',
'Ksiazka', 'Cieple skarpety', 'Tablet',
'Sweter', 'Sweter', 'Rozga']]


>>> miasta = []; czy_grzeczne = ()
>>> print(type(miasta), type(czy_grzeczne))
(<class 'list'>, <class 'tuple'>)
```

```
>>> print(len(miasta), len(czy_grzeczne))
>>> liczba_prezentów = [1] #jednopozycyjna lista
>>> liczba_prezentów_1 = (1,) #jednopozycyjna krotka
>>> liczba_prezentów_2 = (1) #to nie jest krotka! Co to jest?
>>> print(type(liczba_prezentów_1), type(liczba_prezentów_2))
<class 'tuple'> <clss 'int'>
```

Zadanie 3

Python jako Mikołaj nie sprawdza się – pomóż mu umieścić informacje w czy_grzeczne – tak, by nie tworzyć na nowo zmiennej, ale by zawierała poniższe informacje. Dodaj miasta do miast.

Miasta: Poznań, Kraków, Bielsko Biała

Czy grzeczne? True, False, True,

True, False, True, True, True

Zadanie 3

```
>>> miasta.append('Poznan')
>>> miasta.append('Krakow')
>>> miasta.append('Bielsko Biala')


>>> miasta.extend(['Poznan','Krakow', 'Bielsko Biala'])
```

Zadanie 3

```
>>> miasta = miasta + ['Poznan', 'Krakow', 'Bielsko Biala']
>>> miasta
['Poznan', 'Krakow', 'Bielsko Biala']
>>> czy_grzeczne = czy_grzeczne + (True, False, True, True, False, True, True, True)
>>> czy_grzeczne
(True, True, False, True, True, False, True, True, True)
```

Zadanie 4

Pomóż PyMikołajowi zrobić unikatową listę prezentów w porządku alfabetycznym.

```
>>> prezenty = ['Klocki Lego', 'Gra planszowa', 'Ksiazka',
'Cieple skarpety', 'Tablet', 'Sweter', 'Sweter', 'Rozga']
>>> unikatowe_prezenty = set(prezenty)
>>> unikatowe_prezenty
set(['Cieple skarpety', 'Ksiazka', 'Tablet', 'Gra planszowa',
'Rozga', 'Klocki Lego', 'Sweter'])
>>> unikatowe_prezenty
set(['Cieple skarpety', 'Ksiazka', 'Tablet', 'Gra planszowa',
'Rozga', 'Klocki Lego', 'Sweter'])
>>> unikatowe_prezenty.sort() #??
```

```
>>> prezenty = ['Klocki Lego', 'Gra planszowa', 'Ksiazka',
'Cieple skarpety', 'Tablet', 'Sweter', 'Sweter', 'Rozga']
>>> unikatowe_prezenty = set(prezenty)
>>> unikatowe_prezenty
set(['Cieple skarpety', 'Ksiazka', 'Tablet', 'Gra planszowa',
'Rozga', 'Klocki Lego', 'Sweter'])
>>> unikatowe_prezenty
set(['Cieple skarpety', 'Ksiazka', 'Tablet', 'Gra planszowa',
'Rozga', 'Klocki Lego', 'Sweter'])
>>> unikatowe_prezenty.sort()
>>> unikatowe_prezenty = sorted(unikatowe_prezenty)
>>> unikatowe_prezenty
['Cieple skarpety', 'Gra planszowa', 'Klocki Lego', 'Ksiazka',
'Rozga', 'Sweter', 'Tablet']
```

Porównanie - przypomnienie

W Pythonie możemy użyć
następujących operatorów do porównań:

==

is

!=

not

>=

<=

in

Oraz do ich łączenia:

and

or

Zadanie 5

Sprawdź na bazie naszych list, które operatory będą działały na listach i krotkach.

Zadanie 5

Sprawdź na bazie naszych list, które operatory będą działały na listach i krotkach.

==
is
!=
not
>=
<=
in
and
or

Operacja	Wynik
x or y	if *x* is false, then *y*, else *x*
x and y	if *x* is false, then *x*, else *y*
not x	if *x* is false, then True, else False

Warunkowość - przypomnienie

Składnia:

```
if WARUNEK:
   zrob_cos_tam()
elif WARUNEK_2:
   zrob_cos_tam_2()
else:
   zrob_cos_innego()
```

Przykład:

```
if len(lista) > 1:
   print('OK')
elif len(lista) < 1:
   print('Och')
else:
   print('FOCH')
```

Zadanie 6

Pomóż Mikołajowi wyświetlić po kolei na ekranie wszystkie unikatowe prezenty.

Zadanie 6

Pomóż Mikołajowi wyświetlić po kolei na ekranie wszystkie unikatowe prezenty.

```
>>> print("\n".join(unikatowe_prezenty))
Cieple skarpety
Gra planszowa
Klocki Lego
Ksiazka
Rozga
Sweter
Tablet
```

Zadanie 6

Pomóż Mikołajowi wyświetlić po kolei na ekranie wszystkie unikatowe prezenty.

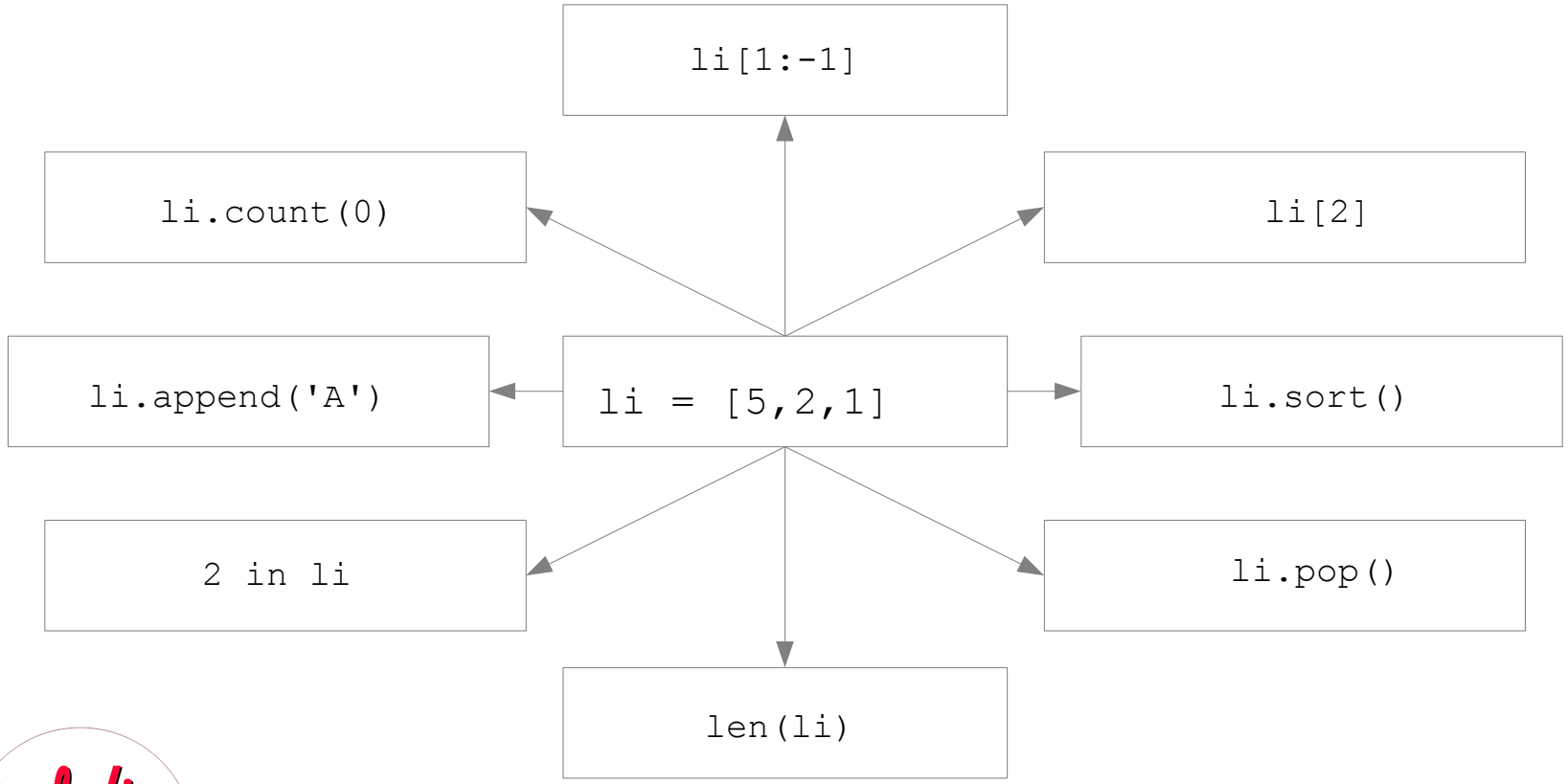
```
>>> for prezent in unikatowe_prezenty:
    print(prezent)


Cieple skarpety
Gra planszowa
Klocki Lego
Ksiazka
Rozga
Sweter
Tablet
```

Zapętlimy się?

Pętla for. Czyli powtarzanie poleceń.

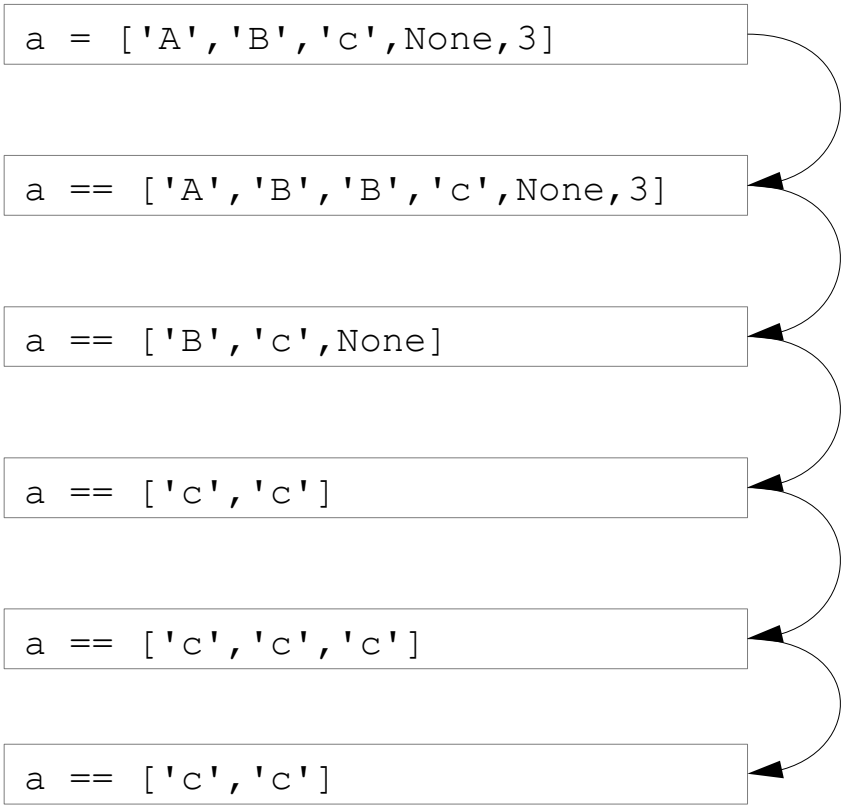
Za tydzień ;)

Podsumowanie



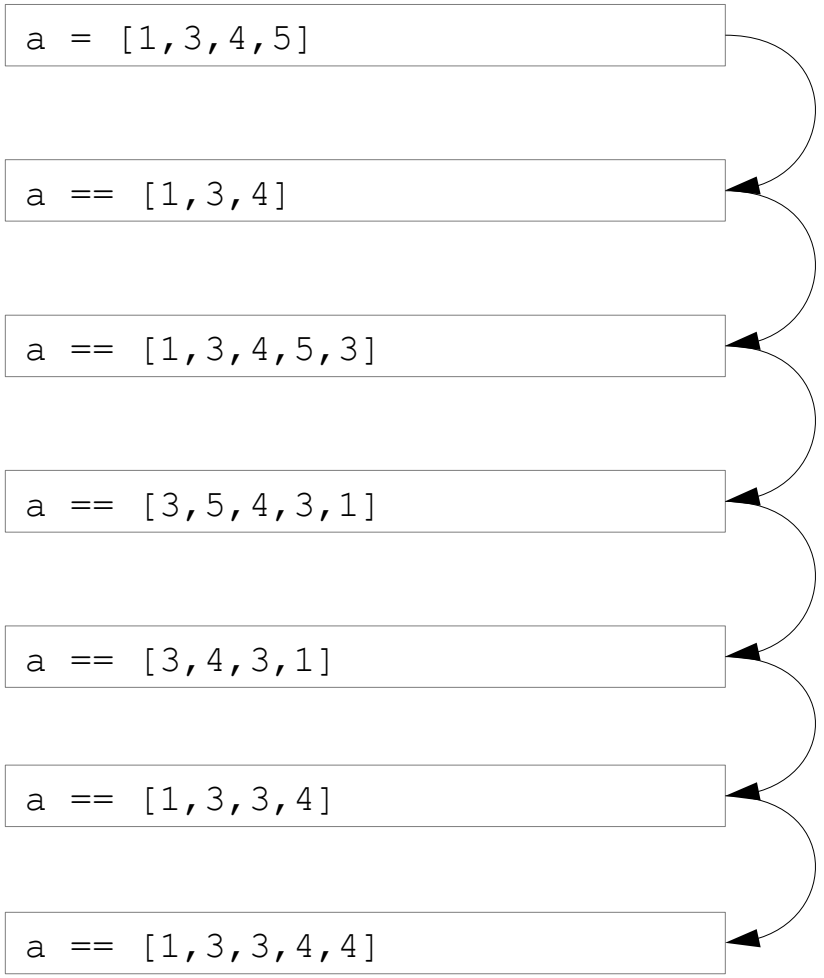
Podsumowanie

```
1) a = a[2:5]
2) a = [a[-2]] +
   [a[1]]
3) a = a[:2]
4) a = [a[-1]]*3
5) a = a[:2] + a[1:]
```



Podsumowanie

```
1) a.reverse()
2) a.sort()
3) a.pop()
4) a.append(4)
5) a = a + [5,3]
6) a.remove(5)
```



Pink Panther's To Do list:

- To do
- To do
- To do, to do, to do, to do, to doooo