
02. Още колекции

— 10 октомври 2024 —

Какво видяхте в предишната лекция?

- Свалете си Python и изберете IDE
- Кодът седи в `.py` файлове
- Основни типове данни - `int`, `float`, `complex`, `str`, `bool`, `None` (проверете типа с `type(x)`)
- Променливите сочат към стойност, те не са стойност
- Колекции - `list`, `tuple`
- Mutable vs Immutable

Нямате въпроси? (или имате, няма значение)

Е, ние имаме...

Какъв ще е резултатът от следните операции?

$3 / 2$	$\neq 1.5$
$3 // 2$	$\neq 1$
$0.1 + 0.2$	$\neq 0.30000000000000000004$
$0.1 + 0.1$	$\neq 0.2$
$0.5 - 0.2$	$\neq 0.3$
$0.3 / 3$	$\neq 0.09999999999999999999$

И още един

Какъв е типът?

```
>>> things = ['eggs', ('spam', 'spam', 'spam'), 'ham']
```

```
>>> type(things[1][2][3])
```

```
<class 'str'>
```

```
>>> things[1][2][3]
```

```
'm'
```

А така?

Малко по-различна ситуация:

```
>>> things = ('eggs', ['spam', 'spam', 'spam'], 'ham')
>>> things[1][2][3] = 'h'
>>> type(things[1][2][3])
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#69>", line 1, in <module>
```

```
    things[1][2][3] = 'h'
```

```
TypeError: 'str' object does not support item assignment
```

Списъци (cont.)

Списъците съдържат "референции" към елементи.

```
coffee, cheese, crackers, tea = 'coffee', 'cheese', 'crackers', 'tea'  
# unpacking
```

```
things_i_like = [coffee, cheese, crackers]  
things_you_like = [crackers, coffee, tea]
```

```
things_i_like[0] == things_you_like[1] # True  
things_i_like[0] is things_you_like[1] # True
```

Списъци (the fun stuff)

Това позволява някои интересни неща:

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']  
cheeses.append(cheeses)
```

```
cheeses[-1] is cheeses # True  
print(cheeses) # ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto', [...]]
```

Списъци (-ception)

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']  
teas = ['chai', 'earl grey', 'jasmine', 'oolong']
```

```
breakfast = [cheeses, teas]  
print(breakfast[0][1]) # bergkäse
```

```
breakfast[1][2] = ['шкембе', 'люти чушки', 'оцет с чесън']  
print(teas) # ?
```

```
['chai', 'earl grey', ['шкембе', 'люти чушки', 'оцет с чесън'],  
'oolong']
```


Методи на списъци

- `.index(element)` - Индекса на първото срещане на `element` в списъка или гърми с `ValueError`
- `.count(element)` - Броят срещания на `element` в списъка
- `.append(element)` - Добавя `element` в края на списъка
- `.extend(elements)` - Добавя елементите на `elements` в списъка (като `+` ама по-бързо)
- `.sort()` - Сещате се
- ...

Tuple (каквото забравихме)

Миналия път (не) казахме, че може и без скобите:

```
people = 'Niki', 'Kiro', 'Genata'  
people = 'Niki',
```

```
people = ('Niki') # най-вероятно не е каквото очаквате
```

- Имат методите `index` и `count` като на списъците
- Скучно

Любопитни неща

Ако имате n-торка, съдържаща само имена от лявата страна на присвояване, може да постигнете интересни ефекти:

```
(a, b) = 1, 2
```

```
print(a)  # 1
```

Още по-любопитни неща

Всъщност скобите изобщо не са задължителни:

```
a, b = 1, 2
```

```
print(a)  # 1
```

Още по-по-любопитни неща

```
numbers = (1, 2, 3)
```

```
a, b, c = numbers
```

Най-любопитни неща

```
a, *b, c = 1, 2, 3, 4, 5
```

```
a = 1
```

```
b = [2, 3, 4]
```

```
c = 5
```

Сравняване на списъци и кортежи

Сравняват се лексикографски:

```
>>> (1, 2) < (1, 3)
```

```
True
```

```
>>> (1, 2) < (1, 2)
```

```
False
```

```
>>> (1, 2) < (1, 2, 3)
```

```
True
```

```
>>> [1, 2] < [1, 3]
```

```
True
```

```
>>> (1, 2) < [1, 3] # tuple vs. list
```

```
# поражда грешка:
```

```
#      TypeError: unorderable types: tuple() < list()
```

Популярни структури от данни

Опашка (queue, FIFO buffer) - можете да ползвате списък.

```
adjectives = []
```

```
def add_adjective(items):  
    adjectives.append(items)
```

```
def get_adjective():  
    return adjectives.pop(0)
```

```
add_adjective('John')  
add_adjective('Terry')  
add_adjective('Graham')  
add_adjective('Eric')
```

```
print(', '.join(adjectives) + ', Michael, Terry') # John, Terry, Graham,  
Eric, Michael, Terry
```


Речник (dict)

- Речник = `dict` = hashtable = associative array
- Реда не е гарантиран
- Асоциира ключ със стойност

Речник (dict)

Индексите не винаги са достатъчно информативни

```
artist_names = {  
    'John': 'Cleese',  
    'Terry': 'Gilliam',  
    'Graham': 'Chapman',  
    'Eric': 'Idle',  
}  
  
print('Eric\'s last names is ' + artist_names['Eric'])
```

Dict

Можем да добавяме нови стойности във вече създаден речник

```
artist_names['Michael'] = 'Palin'
```

```
print(artist_names) # {'John': 'Cleese', 'Terry': 'Gilliam',  
                    #  'Graham': 'Chapman', 'Eric': 'Idle',  
                    #  'Michael': 'Palin'}
```

Речникът също е не подреден, or is it?

Dict

```
>>> ages = {'Вселена': 1.4E10, 'Сайтът на курса': 1}
```

```
>>> ages['Айнщайн'] = 76
```

```
>>> ages['Верди'] = 87
```

```
>>> ages['Христос'] = 33
```

```
>>> ages['Айнщайн']
```

```
76
```

```
>>> 'Христос' in ages
```

```
True
```

```
>>> ages.get('Стамат')
```

```
None
```

```
>>> ages.get('Стамат', 'няма такъв')
```

```
'няма такъв'
```

Три други начина за създаване на речник

Чрез наименувани параметри към конструктора (не питайте):

```
>>> dict(france="Paris", italy="Rome")
{'france': 'Paris', 'italy': 'Rome'}
```

Чрез списък от двойки:

```
>>> dict([('One', 'I'), ('Two', 'II')])
{'One': 'I', 'Two': 'II'}
```

Чрез списък от ключове и стойност по подразбиране:

```
>>> dict.fromkeys([1, 2, 3], 'Unknown')
{1: 'Unknown', 2: 'Unknown', 3: 'Unknown'}
```

Речници и хеш функции

- Функция от вид: обект → число
- Не е нужно да е инективна
- Ако два обекта са еднакви по стойност, те имат еднакъв хеш
- Възможно е различни обекти да имат еднакъв хеш
- За да работят речниците и множествата, ключовете трябва да могат да се сравняват с ==
- Добре е това да става по смислен начин
- Ключовете (в *Python*) трябва да са **immutable**

Множество (set)

- `set` = множество = колекция без повтарящи се елементи
- Редът не е гарантиран
- Нямаме пряк достъп до конкретен елемент
- Можем да проверяваме за принадлежност
- Можем да обхождаме всички (другата лекция ще видим как)

Set

Множества (за всякакви практически цели неразличими от математическата абстракция със същото име)

```
favourite_numbers = set()
favourite_numbers.add(13)
favourite_numbers.add(73)
favourite_numbers.add(32)
favourite_numbers.add(73)
favourite_numbers.add(1024)
favourite_numbers.add(73)
```

```
print(favourite_numbers) # {32, 73, 666, 13, 1024}
```


Set

Има синтаксис за създаване на множества (както може би сте се досетили)

```
favourite_numbers = {32, 73, 666, 13, 1024}
```

{ } не е празния set!



`{}` е празен речник, по простата причина, че речниците са доста по-често използвана структура от множествата

More sets

```
>>> unique_numbers = {2, 3, 5, 6}
>>> unique_numbers
{2, 3, 5, 6}
>>> unique_numbers.add(5)
>>> unique_numbers
{2, 3, 5, 6}
>>> unique_numbers.remove(5)
>>> unique_numbers
{2, 3, 6}
>>> my_list = [5, 1, 6, 6, 2, 3, 5, 5]
>>> set(my_list)
{1, 2, 3, 5, 6}
```

More sets

можем да проверяваме за принадлежност

```
73 in favourite_numbers # True
```

Операции с множества

```
>>> {1, 2, 3} | {2, 3, 4}
```

```
{1, 2, 3, 4}
```

```
>>> {1, 2, 3} & {2, 3, 4}
```

```
{2, 3}
```

```
>>> {1, 2, 3} - {2, 3, 4}
```

```
{1}
```

```
>>> {1, 2, 3} ^ {2, 3, 4}
```

```
{1, 4}
```

```
>>> {1, 2, 3} < {2, 3, 4}
```

```
False
```

```
>>> {2, 3} < {2, 3, 4} # < - подмножество
```

```
True
```

```
>>> {2, 3} == {2.0, 3}
```

```
True
```

Една полезна употреба на set()

```
a_lot_of_numbers = [1, 4, 2, 6, 2, 3, 6, 8, 9, 3, 2, 1, 5, 4, 2]
print(f"Уникалните елементи са: {set(a_lot_of_numbers)}")
# Уникалните елементи са: {1, 2, 3, 4, 5, 6, 8, 9}
```

Да си подредим данните (в главата)

Когато имаме данни, най-логично е да ги слагаме в колекции.

- `list` (a.k.a. `array`, масив) = подредена последователност от стойности
- `tuple` = непроменяема по състав подредена последователност от обекти (~списък, но не съвсем)
- `dict` = ключове/имена, зад които стоят стойности (без подредба (или пък със?))
- `set` = стойности без повтаряне и без подредба (множество в математическия смисъл)

И сега за новината, която нямате търпение да чуете!

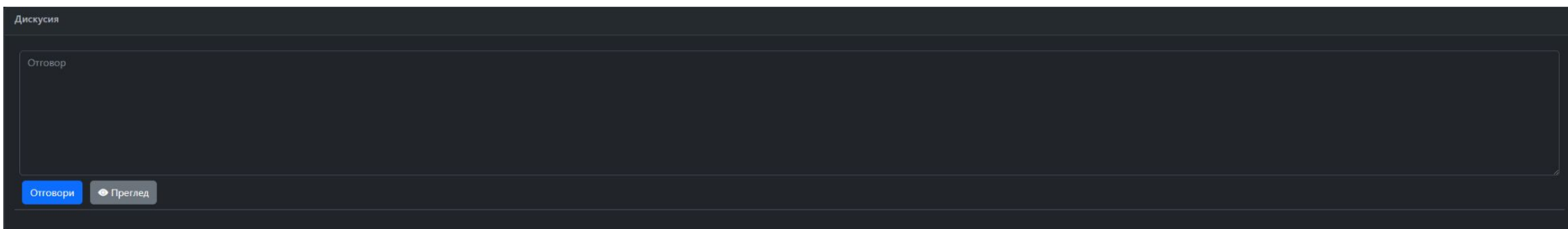


Но преди това...

- В Python пишем имената на променливите в snake_case
- Опитваме се да са описателни
- xx, i, var и прочие - не са
- Именувайки променливата се опитваме да отговорим на въпроса “за какво служи тази променлива”

Имате първо домашно...

- Сега!
- Срокът е до 18:00 на 15.10 (следващият вторник)
- Внимателно се запознайте с това [как да \(не\) си изпращате задачите](#)
- Отдолу има поле за дискусия, ако имате въпроси - ползвайте го



- Знаем, че няма да ни послушате, но...
- Пробвайте да не го оставяте за последния момент, за да можем да ви върнем обратна връзка

Въпроси?