02. Колекции

13 октомври 2022

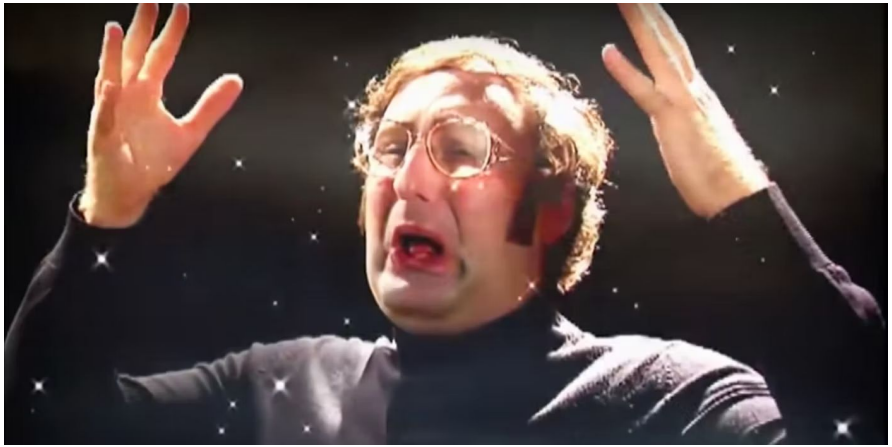
Какво (не) видяхте в предишната лекция?

- 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, dict, set`
- Mutable vs Immutable
- Контролни структури - `if, while, for, switch (break, continue)`
- **4** интервала индентация и **“:”** за начало на блока
- Дефиниране на функции и аргументи на функции
- Функциите (и не само) са обекти и могат да бъдат подавани като аргументи на други функции

Въпроси?

Какво (не) ще видите днес?

- Голи снимки
- Колекциите, които видяхме още в предишната лекция, но малко по-задълбочено
- Функции, които генерират такива колекции
- Функции, които използват такива колекции
- Функции, които едновременно използват и генерират такива колекции



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)
- set = стойности без повтаряне и без подредба (множество в математическия смисъл)
- dict = ключове/имена, зад които стоят стойности (без подредба (или пък със?))

Какво е колекция?

- В Python всички колекции са итерируеми (iterable)
- Един итерируем обект може да бъде обхождан последователно (поне веднъж)
- Някои могат да бъдат обхождани многократно

Списъци (List)

```
nice_things = ['coffee', 'cheese', 'crackers', 'tea']
for thing in nice_things:
    print(f'I tend to like {thing}')
```

Можем и просто да ги индексираме

```
print(nice_things[1]) # cheese
print(nice_things[-1]) # tea
print(nice_things[-3]) # cheese
```

Списъци (List)

```
cute_animals = ['cat', 'raccoon', 'panda', 'red panda', 'marmot']

cute_animals[1:3]  # ['raccoon', 'panda']
cute_animals[-1]  # 'marmot'
cute_animals[1:-1]  # ['raccoon', 'panda', 'red panda']
cute_animals[::-1]  # ['marmot', 'red panda', 'panda', 'raccoon', 'cat']
cute_animals[-1:0:-1]  # ['marmot', 'red panda', 'panda', 'raccoon']
cute_animals[-1:0:-2]  # ['marmot', 'panda']
```

Списъци (List)

Списъците съдържат "указатели" 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 (List)

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
- ...

Range

range връща итерируемо за интервал от числа

```
numbers = range(3)
for number in numbers:
    print('We can count to ' + str(number))
```

Range

```
numbers = range(10, 13)
for number in numbers:
    print('We can count to ' + str(number))
```

Range

range може и в обратен ред

```
numbers = range(13, 0, -1)
for number in numbers:
    print('We can count to ' + str(number))
```

Tuple

алтернативен синтаксис за кортежи

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

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

Има методите 'index' и 'count' като на списъците

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

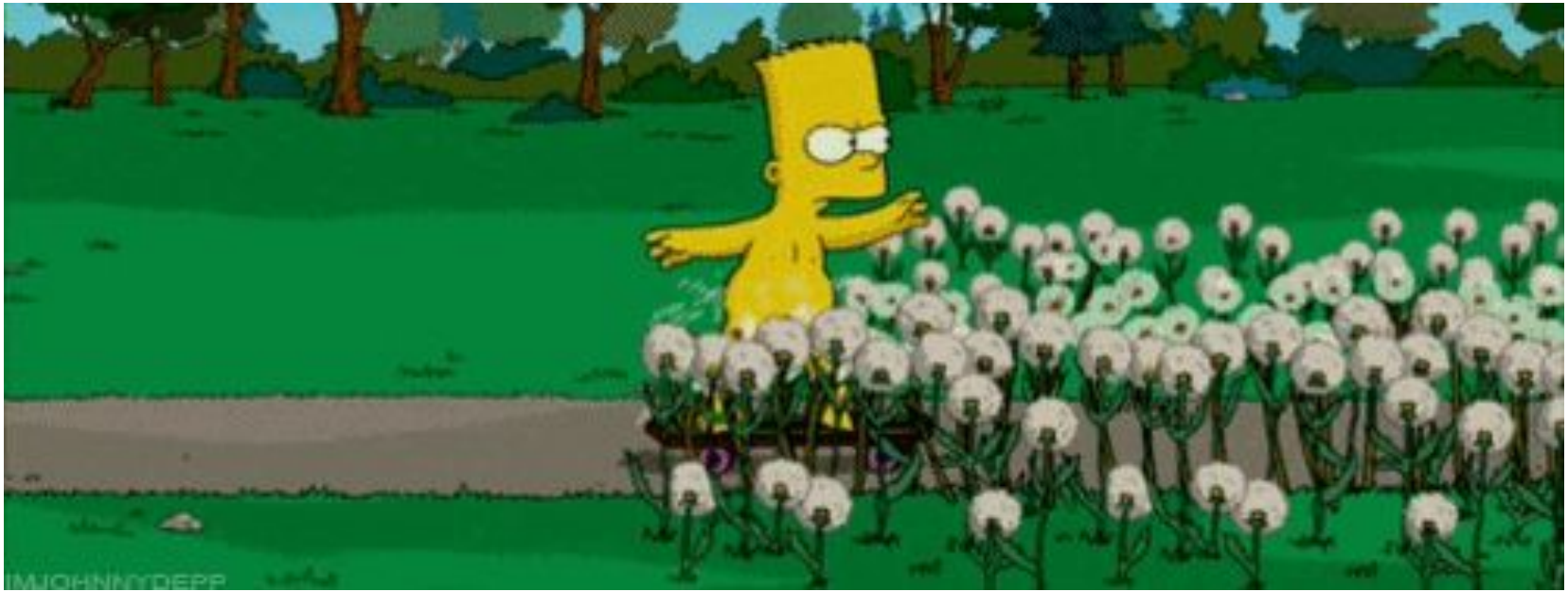
```
numbers = (1, 2, 3)

a, b, c = numbers
```

Най-любопитни неща

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

```
a = 1
b = [2, 3, 4]
c = 5
```

Къде са голите снимки?



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('Magic')
add_adjective('Woody Allen')
add_adjective('Zombie')
add_adjective('Superhero')

print(' '.join(adjectives) + ' Jesus!') # Magic Woody Allen Zombie
Superhero Jesus!
```

Sets

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}
```

Sets

Множествата са итеруеми и НЕ*(баш)*подредени

```
for num in favourite_numbers:
    print('I really like the number ' + str(num))
```

Добре де, ще спрем ли да обясняваме за (не) подредеността на dict/set?

dict() now uses a “compact” representation pioneered by PyPy. The memory usage of the new dict() is between 20% and 25% smaller compared to Python 3.5. PEP 468 (Preserving the order of **kwargs in a function.) is implemented by this. **The order-preserving aspect of this new implementation is considered an implementation detail and should not be relied upon** (this may change in the future, but it is desired to have this new dict implementation in the language for a few releases before changing the language spec to mandate order-preserving semantics for all current and future Python implementations; this also helps preserve backwards-compatibility with older versions of the language where random iteration order is still in effect, e.g. Python 3.5). (Contributed by INADA Naoki in issue 27350. Idea originally suggested by Raymond Hettinger.)

Sets

можем да проверяваме за принадлежност

```
73 in favourite_numbers # True
```

Sets

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

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

{} не е празния set!

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
```

Dict

Индексите не винаги са достатъчно информативни

```
artist_names = {
    'Eddie': 'Vedder',
    'Maynard': 'Keenan',
    'Matthew': 'Bellamy',
    'James': 'LaBrie',
}

print('Eddie\'s last names is ' + artist_names['Eddie'])
```

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

Dict

Можем да добавяме нови стойности във вече създаден речник

```
names['Devin'] = 'Townsend'

print(names) # {'Devin': 'Townsend', 'Matthew': 'Bellamy',
             #  'Eddie': 'Vedder', 'James': 'LaBrie', 'Maynard': 'Keenan'}
```

Речникът също е неподреден, or is it?

Три други начина за създаване на речник

Чрез наименовани 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
- Желателно е ключовете да са immutable

Кой какво сиренье има

```
data = [('John', 'Tilsit'), ('Eric', 'Cheshire'), ('Michael', 'Camembert'),
        ('Terry', 'Gouda'), ('Terry', 'Port Salut'), ('Michael', 'Edam'),
        ('Eric', 'Ilchester'), ('John', 'Fynbo')]

def cheeses_by_owner(cheeses_data):
    by_owner = {}
    for owner, cheese in cheeses_data: # <- tuple unpacking
        if owner in by_owner:
            by_owner[owner] .append(cheese)
        else:
            by_owner[owner] = [cheese]
    return by_owner
```

Въпроси?