03. Още колекции (и още)

12 октомври 2023

Какво (не) видяхте в предишната лекция?

- 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
- Малко по-задълбочено колекциите `list` и `tuple`

Въпроси?

Въпрос за живота, вселената и tuples / lists

Какъв ще е резултатът от следния код?

```
>>> other_things = (42, [1, 2, 3], 'baba')
>>> other_things[1] += [4]

Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    other_things[1] += [4]
TypeError: 'tuple' object does not support item assignment

>>> other_things
(42, [1, 2, 3, 4], 'baba')
```

А така?

Какъв ще е резултатът от следния код?

```
>>> my_favourite_things = ['spam'] * 100
```

Впечатляващо, а?

Какво (не) ще видите днес?

- Голи снимки
- Колекциите, които видяхме още в първата 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

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
```

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))
```

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
```

Една полезна функция 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}
```

Къде са голите снимки?

Имаме още

Тери Джоунс (да, от Монти Пайтън) в ролята на “Голият органист”

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'])
```

{} 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

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

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

Речникът също е неподреден, or is it?

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
- 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
```

Map/Filter/Reduce/All/Any/Lambda

- `map(function, iterable)` създава колекция от резултатите от прилагането на function върху всеки елемент от iterable
- `filter(function, iterable)` създава колекция само с елементите, за които function върне `True`
- `reduce(function, iterable)` вика function с елементите на колекцията, докато сведе всичко до една стойност (във `functools` вж.тук)
- `all(iterable)` всички елементи се оценяват на истина
- `any(iterable)` поне един от елементите се оценява на истина
- lambda функции - анонимни функции, които често вървят ръка за ръка с горните

за любознателните: `map()` и `filter()` са мързеливи

Comprehensions

- Изрази, които *генерират* колекции
- Елегантен заместител на `map()` и/или `filter()`
- Колекциите могат да са динамични

List comprehension

```
[израз for променлива in поредица if условие]
```

```
>>> [x * x for x in range(0, 10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

>>> [x * x for x in range(0, 10) if x % 2]
[1, 9, 25, 49, 81]
```

List comprehension

Един list comprehension може да се вложи в друг, защото връща нещо итерируемо

```
>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]

[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Generator expression

- Кръгли скоби вместо квадратни
- Като list comprehension, но се изпълнява динамично (lazy evaluation)
- На всяка стъпка итератора оценява условието и израза за следващата стойност

Set comprehension

Като list comprehension, но с `{}`

```
>>> import math
>>> {int(math.sqrt(x)) for x in range(1, 100)}
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

Dict comprehension

```
>>> {i: chr(65 + i) for i in range(10)}

{0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H', 8: 'I',
9: 'J'}
```

Колекции за хора без компания в петък вечер

- deque - двупосочни опашки
- OrderedDict - речник, който помни реда
- defaultdict - речник със стойност по подразбиране
- Counter - речник, който брои повтарящи се стойности
- namedtuple - кортеж с именовани полета

Deque

```
from collections import deque

adjectives = deque()

def add_adjective(items):
    adjectives.append(items)
def get_adjective():
    return adjectives.popleft()

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

Defaultdict

```
from collections import defaultdict

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 = defaultdict(list)
    for owner, cheese in cheeses_data:
        by_owner[owner].append(cheese)
    return by_owner
```

Упражнение - Бикове и Крави (1)

```
import random # Засега - магия. След няколко лекции - ежедневие.

LENGTH = 4 # Константа за дължина на числата

def get_secret():
    """Generate random 4-digit number with unique digits (no zero)."""
    all_digits = list(map(str, range(1, 10)))
    random.shuffle(all_digits)
    return ''.join(all_digits[:LENGTH])
```

Упражнение - Бикове и Крави (2)

```
def compare(num1, num2):
    """Return bulls/cows count between two numbers. See the rules."""
    bulls, cows = 0, 0
    for dig1, dig2 in zip(num1, num2):
        if dig1 == dig2:
            bulls += 1
        elif dig1 in num2:
            cows += 1
    return bulls, cows
```

Упражнение - Бикове и Крави (3)

```
SECRET = get_secret()
print('Намислих си четирицифрено число без повторение на цифрите и без
нули. Опитай да познаеш...')
while True:
    guess = input('Въведи предположение: ')
    bulls, cows = compare(SECRET, guess)
    if bulls == LENGTH:
        print(f'Позна, машино!')
        break
    else:
        print(f'Имаш {bulls} бика и {cows} крави.')
```

Упражнение - Бикове и Крави (4)

```
while True:
    guess = input('Въведи предположение: ')      # Вмъкваме малко валидация в while цикъла
    if not guess.isdigit():
        print('Това май не е число.')
        continue
    if len(guess) != LENGTH:
        print('Числото трябва да е четирицифрено.')
        continue
    if len(set(guess)) != LENGTH:
        print('Числото трябва да съдържа само уникални цифри.')
        continue
    if '0' in guess:
        print('Числото не може да съдържа нула.')
        continue
    bulls, cows = compare(SECRET, guess)
    …
```

Въпроси?