Лекция 3 "Словари, множества и выражения-генераторы"

часть 2 Генераторы

Финансовый университет при Правительстве РФ, лектор С.В. Макрушин

v 0.6 02.10.2020

Разделы:

In [1]:

```
# загружаем стиль для оформления презентации
from IPython.display import HTML
from urllib.request import urlopen
html = urlopen("file:./lec_v1.css")
HTML(html.read().decode('utf-8'))
```

Out[1]:

Выражения-генераторы

Выражения-генераторы для списков

In [1]:

```
lst_val = [1, 2, 7, 11, 8, 2]
# создание списка с помощью цикла:
lst_new = []
for el in lst_val:
    lst_new.append(el * 2)
lst_new
```

Out[1]:

```
[2, 4, 14, 22, 16, 4]
```

In [2]:

```
# создание аналогичного списка при помощи генератора:
lst_gen = [el * 2 for el in lst_val]
lst_gen
```

Out[2]:

```
[2, 4, 14, 22, 16, 4]
```

In [3]:

```
# создание списка с помощью цикла и условия (фильтра):
lst_new2 = []
for el in lst_val:
    if el % 2 == 0: # число el четное (остаток от деления на 2 равен 0)
        lst_new2.append(el * 2)
lst_new2
```

Out[3]:

```
[4, 16, 4]
```

In [4]:

```
# создание аналогичного списка при помощи генератора:
lst_gen2 = [el * 2 for el in lst_val if el % 2 == 0]
lst_gen2
```

Out[4]:

```
[4, 16, 4]
```

Задача обхода и модификации списка

Пример: отфильтровать список, оставив в нем только нечетные элементы

In [8]:

```
# исходный список:
filt_list = [1, 3, 2, 8, 4, 11, 8, 9]
```

In [9]:

```
# НЕ работающий вариант:
for el in filt_list:
    if el % 2 == 0:
        del el # НЕ модифицирует список, удаляет переменную el
filt_list
```

Out[9]:

```
[1, 3, 2, 8, 4, 11, 8, 9]
```

In [15]:

```
filt_list = [1, 3, 2, 8, 4, 11, 10, 9]
print(f'длина списка: {len(filt_list)}')

# Еще один НЕ работающий вариант:
for ind, el in enumerate(filt_list):
    print(f'ind: {ind}, el: {el}')
    if el % 2 == 0:
        print('removing')
        del filt_list[ind] # мы "пилим сук, на котором сидим": удаление элемента влияет на
        # после удаления элемента с индексом ind его место занимает следующий элемент =>
        # на следующей итерации он не будет рассмотрен!
filt_list
```

```
длина списка: 8
ind: 0, el: 1
ind: 1, el: 3
ind: 2, el: 2
removing
ind: 3, el: 4
removing
ind: 4, el: 10
removing
```

Out[15]:

```
[1, 3, 8, 11, 9]
```

В результате выполнения `del filt_list[ind]` итератор "перескочил" значение 8, 11, 9.

In [16]:

```
filt_list = [1, 3, 2, 8, 4, 11, 10, 9]

for ind in range(len(filt_list)-1, -1, -1): # идем с конца в начало с шагом -1
    # в явном виде итерируемся по индексу (целочисленная перменная ind), а не по элементам
    el = filt_list[ind]
    print(f'ind: {ind}, el: {el}')
#     do_action(element)
    if el % 2 == 0:
        print('removing')
        del filt_list[ind]
        # удаление элементов при обходе с хвоста списка не меняет индексов предыдущих элеме
        # после удаления мы не "перескакиваем" очередное (предыдущее) значение
filt_list # РАБОТАЕТ!
```

```
ind: 7, el: 9
ind: 6, el: 10
removing
ind: 5, el: 11
ind: 4, el: 4
removing
ind: 3, el: 8
removing
ind: 2, el: 2
removing
ind: 1, el: 3
ind: 0, el: 1
```

Out[16]:

```
[1, 3, 11, 9]
```

In [22]:

```
# фильтарция списков при помощи генераторов - оптимальное решение для большинства случаев:
filt_list = [1, 3, 2, 8, 4, 11, 10, 9]
filt_list2 = filt_list
filt_list = [el for el in filt_list if el % 2 == 1] # просто, быстро и правильно работает!
filt_list
```

Out[22]:

```
[1, 3, 11, 9]
```

In [23]:

```
# важный нюанс: исходный объект списка не изменился
# это показывают другие переменные, ссылающиеся на него:
filt_list2
```

Out[23]:

```
[1, 3, 2, 8, 4, 11, 10, 9]
```

In [25]:

```
# фильтрация с сохранением результата в том же объекте списка
# решает задачу изменения исходного объекта, если ссылка на объект filt_list хранится в дру
filt_list = [1, 3, 2, 8, 4, 11, 10, 9]
filt_list2 = filt_list
filt_list[:] = [el for el in filt_list if el % 2 == 1]
filt_list
```

Out[25]:

```
[1, 3, 11, 9]
```

In [27]:

```
filt_list2 # работает!
```

Out[27]:

```
[1, 3, 11, 9]
```

>

Пример: задача приведения списка к "плоскому" виду

- к оглавлению

"Плоский" список (flatten list)

In [5]:

```
list2d = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
```

In [6]:

```
flat_list = []

for sublist in list2d:
    for item in sublist:
        flat_list.append(item)

flat_list
```

Out[6]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [7]:

```
# вложенные генераторы списков:
flat_list = [item for sublist in list2d for item in sublist]
flat_list
```

Out[7]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [8]:

```
%%timeit
list2d = [[1, 2, 3], [4, 5, 6], [7], [8, 9]] * 10
for sublist in list2d:
    for item in sublist:
        flat_list.append(item)
```

```
8.67 µs ± 464 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [9]:

```
%%timeit
list2d = [[1, 2, 3],[4, 5, 6], [7], [8, 9]] * 10
[item for sublist in list2d for item in sublist]
```

```
3.6 µs ± 108 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

=> Генераторы списков обычно существенно быстрее аналогичных циклов.

Приведение к плоскому виду с помощью функции `sum()` :

- Вид функции `sum(iterable, start=0)`
- Docstring: Return the sum of a 'start' value (default: 0) plus an iterable of numbers

In [10]:

```
list2d = [[1, 2, 3],[4, 5, 6], [7], [8, 9]]
sum(list2d, [])
```

Out[10]:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

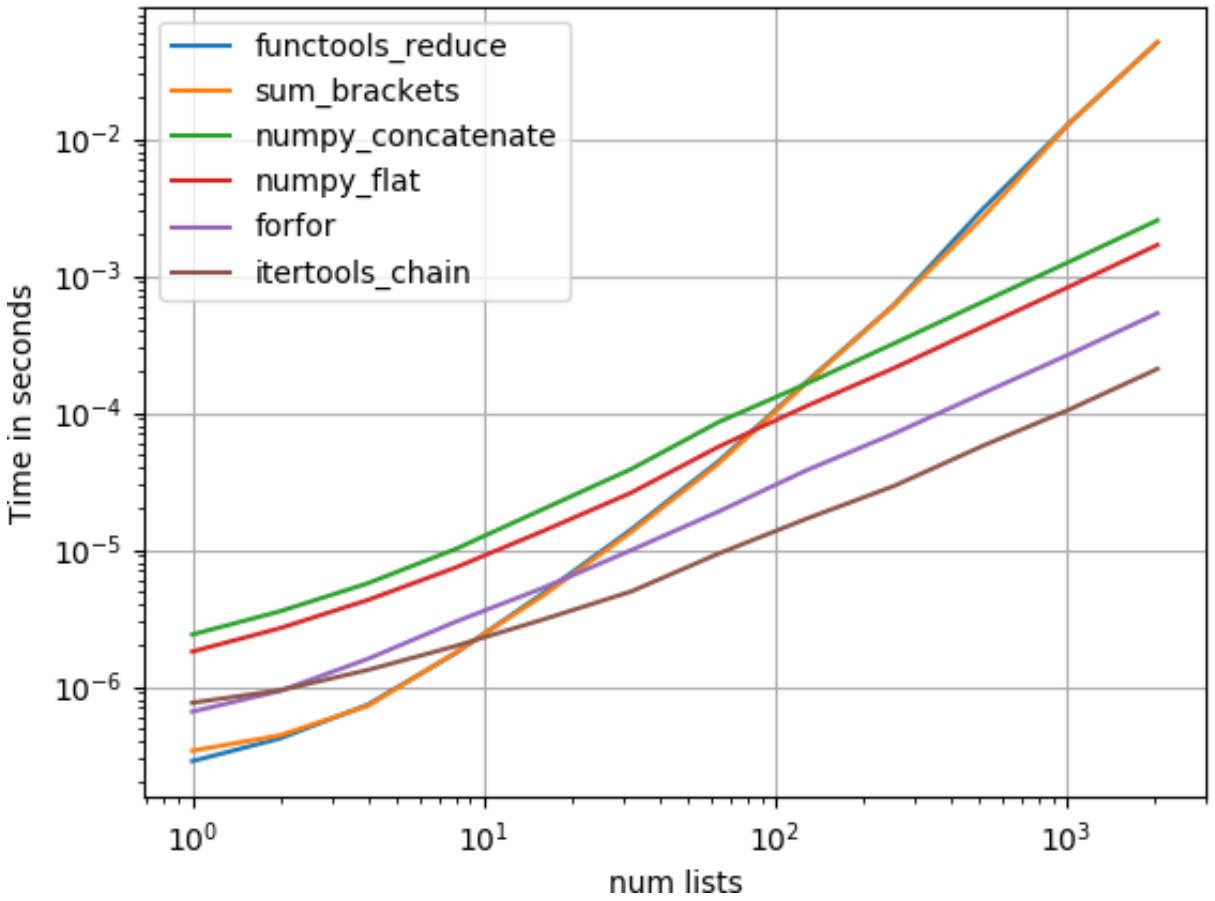
In [11]:

```
%%timeit
list2d = [[1, 2, 3], [4, 5, 6], [7], [8, 9]] * 10
sum(list2d, [])
```

```
8.73 µs ± 99.3 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [12]:

```
import functools
import itertools
import numpy

def forfor(a):
    return [item for sublist in a for item in sublist]


def sum_brackets(a):
    return sum(a, [])


def functools_reduce(a):
    return functools.reduce(operator.concat, a)


def itertools_chain(a):
    return list(itertools.chain.from_iterable(a))


def numpy_flat(a):
    return list(numpy.array(a).flat)


def numpy_concatenate(a):
    return list(numpy.concatenate(a))
```



Эффективность различных механизмов получения плоского списка

>

Выражения генераторы, генераторы множеств и словарей

Выражения-генераторы

- к оглавлению

In [38]:

```
lst_val
```

Out[38]:

```
[1, 2, 7, 11, 8, 2]
```

In [39]:

```
cor_gen = (el * 2 for el in lst_val)
cor_gen
```

Out[39]:

```
<generator object <genexpr> at 0x000001BE879C1F68>
```

In [40]:

```
# использование выражения генератора в качестве источника итерируемых данных:
sum(cor_gen)
```

Out[40]:

```
62
```

In [51]:

```
# передаем генератор напрямую:
sum((el * 2 for el in lst_val))
```

Out[51]:

```
62
```

In [53]:

```
# передаем генератор напрямую (упрощенный синтаксис):
sum(el * 2 for el in lst_val)
```

Out[53]:

```
62
```

In [45]:

```
cor_gen = (el * 2 for el in lst_val)
cor_gen
```

Out[45]:

```
<generator object <genexpr> at 0x000001BE87A8C830>
```

In [46]:

```
# итерирование по генератору с помощью цикла:
for e in cor_gen:
    print(e)
```

```
2
4
14
22
16
4
```

In [50]:

```
cor_gen = (el * 2 for el in lst_val)
cor_gen
```

Out[50]:

```
<generator object <genexpr> at 0x000001BE87A8C7D8>
```

In [49]:

```
next(cor_gen)
```

Out[49]:

```
4
```

In [54]:

```
import math
```

In [55]:

```
sum((math.sin(v) for v in range(10000)))
```

Out[55]:

```
1.9395054106807064
```

In [56]:

```
# Допустим и более удобный синтаксис:
sum(math.sin(v) for v in range(10000))
```

Out[56]:

```
1.9395054106807064
```

Генераторы множеств

- к оглавлению

Помимо генераторов списков язык Python поддерживает генераторы множеств. Синтаксис генераторов множеств похож на синтаксис генераторов списков, только вместо квадратных скобок используются фигурные скобки.

In [57]:

```
sg1 = {x**2 for x in [1, 2, 1, 2, 1, 2, 3]}
sg1
```

Out[57]:

```
{1, 4, 9}
```

In [58]:

```
sg2 = [x**3 for x in sg1]
sg2
```

Out[58]:

```
[1, 64, 729]
```

In [59]:

```
{x for x in [1, 2, 1, 2, 1, 2, 3] if x % 2 == 0}
```

Out[59]:

```
{2}
```

In [60]:

```
# нужно помнить, что так пустое множество НЕ объявляется:
v = {}
```

In [61]:

```
type(v)
```

Out[61]:

```
dict
```

In [62]:

```
v2 = set()
```

In [63]:

```
type(v2)
```

Out[63]:

```
set
```

In [64]:

```
# Однако:
{e for e in [1, 2, 3] if e > 10}
```

Out[64]:

```
set()
```

Генераторы словарей

- к оглавлению

Помимо генераторов списков язык Python поддерживает генераторы словарей. Синтаксис генераторов словарей похож на синтаксис генераторов списков, но имеет два отличия:

- выражение заключается в фигурные скобки, а не в квадратные;
- внутри выражения перед циклом for указываются два значения через двоеточие, а не одно:
  - значение, расположенное слева от двоеточия: ключ
  - значение, расположенное справа от двоеточия: значение элемента.

In [65]:

```
keys = ['a', 'b'] # Список с ключами
values = [1, 2]  # Список со  значениями
d18 = {k: v for (k,  v) in zip(keys,  values)}
d18
```

Out[65]:

```
{'a': 1, 'b': 2}
```

In [66]:

```
{k: 2 * v for (k,  v) in d18.items()}
```

Out[66]:

```
{'a': 2, 'b': 4}
```

In [67]:

```
# как и в цикле for скобки при распаковке картежа можно опускать:
{k: 2 * v for k, v in d18.items()}
```

Out[67]:

```
{'a': 2, 'b': 4}
```

In [68]:

```
{e: e ** 2 for e in range(10)}
```

Out[68]:

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

In [69]:

```
{k: 2 * v for k, v in d18.items() if v % 2 == 0}
```

Out[69]:

```
{'b': 4}
```

In [70]:

```
{k: 0 for k in d18}
```

Out[70]:

```
{'a': 0, 'b': 0}
```

Задание к следующему разделу

По книге Н. Прохоренок:

Глава 11 Пользовательские функции

По книге М. Саммерфильд:

Глава 4 Управляющие структуры и функции (разедел "Собственные функции")