List, dict, set comprehensions

Python поддерживает специальные выражения, которые позволяют компактно создавать списки, словари и множества.

На английском эти выражения называются, соответственно:

- List comprehensions
- Dict comprehensions
- Set comprehensions

К сожалению, официальный перевод на русский звучит как абстракция списков или списковое включение что не особо помогает понять суть объекта.

В книге использовался перевод "генератор списка", что, к сожалению, тоже не самый удачный вариант, так как в Python есть отдельное понятие генератор и генераторные выражения, но он лучше отображает суть выражения.

Эти выражения не только позволяют более компактно создавать соответствующие объекты, но и создают их быстрее. И хотя поначалу они требуют определенной привычки использования и понимания, они очень часто используются.

List comprehensions

> генераторы списков

List Comprehensions чаще всего на русский язык переводят как абстракция списков или списковое включение, является частью синтаксиса языка, которая предоставляет простой способ построения списков. Проще всего работу list comprehensions показать на примере. Допустим вам необходимо создать список целых чисел от 0 до n, где n предварительно задается.

```
vlans = [int(vl) for vl in items]
```

Список items:

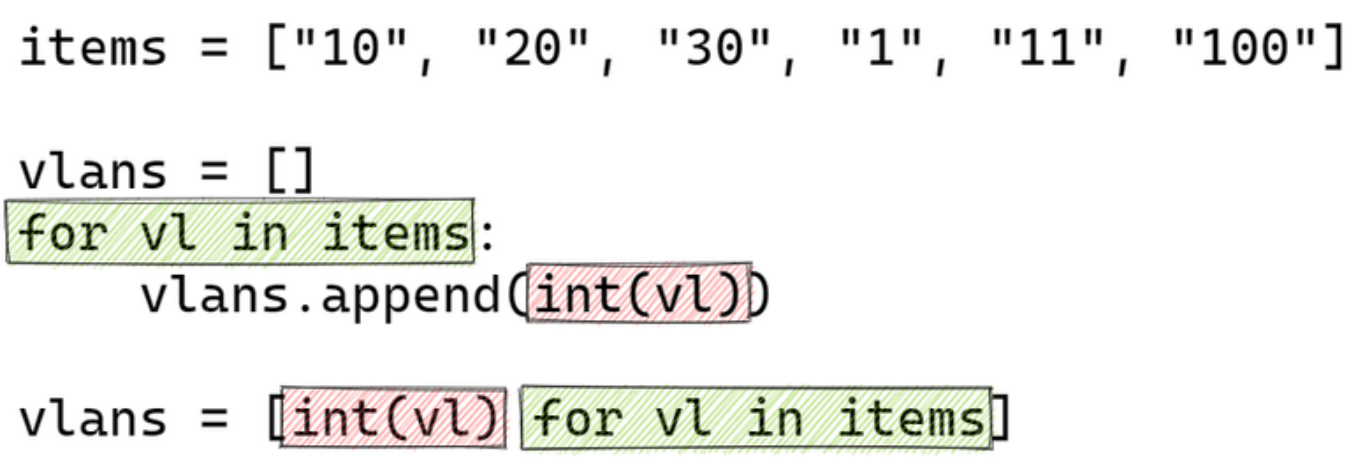
```
items = ["10", "20", "30", "1", "11", "100"]
```

В общем случае, list comprehension это выражение, которое преобразует итерируемый объект в список. То есть, последовательность элементов преобразуется и добавляется в новый список.

List comp выше аналогичен такой цикл:

```
items = ["10", "20", "30", "1", "11", "100"]

vlans = []
  for vl in items:
      vlans.append(int(vl))

print(vlans)
# [10, 20, 30, 1, 11, 100]
```

Соответствие между обычным циклом и генератором списка:



if list comprehensions

В list comprehensions можно использовать выражение `if`. Таким образом можно добавлять в список только некоторые объекты.

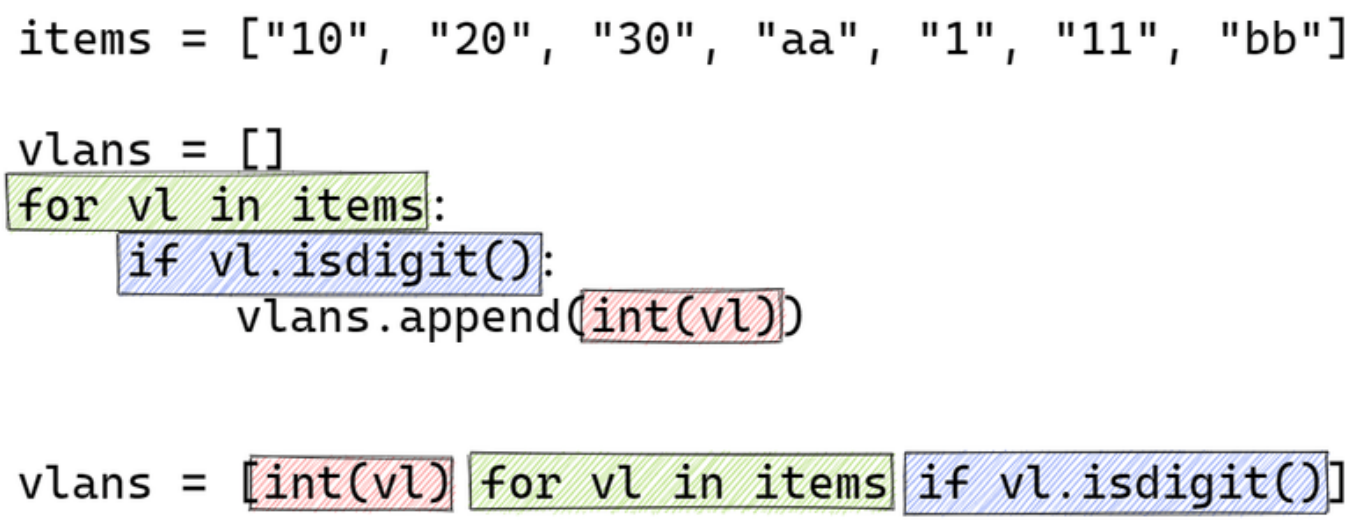
Например, такой цикл отбирает те элементы, которые являются числами, конвертирует их и добавляет в итоговый список only_digits:

```
items = ['10', '20', 'a', '30', 'b', '40']

only_digits = []

for item in items:
    if item.isdigit():
        only_digits.append(int(item))

print(only_digits)
# [10, 20, 30, 40]
```

Аналогичный вариант в виде `list comprehensions`:

```
items = ['10', '20', 'a', '30', 'b', '40']
only_digits = [int(item) for item in items if item.isdigit()]

print(only_digits)
# [10, 20, 30, 40]
```

Соответствие между циклом с условием и генератором списка с условием:



Конечно, далеко не все циклы можно переписать как генератор списка, но когда это можно сделать, и при этом выражение не усложняется, лучше использовать генераторы списка.

> **ПРИМЕЧАНИЕ**
>
> В Python генераторы списка могут также заменить функции `filter` и `map` и считаются более понятными вариантами решения.

List Comprehensions как обработчик списков

В языке Python есть две очень мощные функции для работы с коллекциями: `map` и `filter`. Они позволяют использовать функциональный стиль программирования, не прибегая к помощи циклов, для работы с такими типами как `list`, `tuple`, `set`, `dict` и т.п. Списковое включение позволяет обойтись без этих функций. Приведем несколько примеров для того, чтобы понять о чем идет речь.

Пример с заменой функции `map`.

Пусть у нас есть список и нужно получить на базе него новый, который содержит элементы первого, возведенные в квадрат. Решим эту задачу с использованием циклов:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = []
for i in a:
    b.append(i**2)
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [1, 4, 9, 16, 25, 36, 49]
```

Та же задача, решенная с использованием `map`, будет выглядеть так:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = list(map(lambda x: x**2, a))
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [1, 4, 9, 16, 25, 36, 49]
```

В данном случае применена `lambda`-функция, о том, что это такое и как ее использовать можете прочитать здесь.

Через списковое включение эта задача будет решена так:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [i**2 for i in a]
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [1, 4, 9, 16, 25, 36, 49]
```

Пример с заменой функции `filter`.

Построим на базе существующего списка новый, состоящий только из четных чисел:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = []
for i in a:
    if i%2 == 0:
        b.append(i)
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [2, 4, 6]
```

Решим эту задачу с использованием `filter`:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = list(filter(lambda x: x % 2 == 0, a))
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [2, 4, 6]
```

Решение через списковое включение:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [i for i in a if i % 2 == 0]
print('a = {}\nb = {}'.format(a, b))
# >>> a = [1, 2, 3, 4, 5, 6, 7]
# >>> b = [2, 4, 6]
```

Примеры list comprehensions

С помощью генератора списка также удобно получать элементы из вложенных словарей:

```
    london_co = {
        'r1' : {
        'hostname': 'london_r1',
        'location': '21 New Globe Walk',
        'vendor': 'Cisco',
        'model': '4451',
        'ios': '15.4',
        'ip': '10.255.0.1'
        },
```

```
        'r2' : {
        'hostname': 'london_r2',
        'location': '21 New Globe Walk',
        'vendor': 'Cisco',
        'model': '4451',
        'ios': '15.4',
        'ip': '10.255.0.2'
        },
        'sw1' : {
        'hostname': 'london_sw1',
        'location': '21 New Globe Walk',
        'vendor': 'Cisco',
        'model': '3850',
        'ios': '3.6.XE',
        'ip': '10.255.0.101'
        }
    }

    In [14]: [london_co[device]['ios'] for device in london_co]
    Out[14]: ['15.4', '15.4', '3.6.XE']

    In [15]: [london_co[device]['ip'] for device in london_co]
    Out[15]: ['10.255.0.1', '10.255.0.2', '10.255.0.101']
```

Полный синтаксис генератора списка выглядит так:

```
[expression for item1 in iterable1 if condition1
            for item2 in iterable2 if condition2
            ...
            for itemN in iterableN if conditionN ]
```

Это значит, можно использовать несколько for в выражении.

Например, в списке vlans находятся несколько вложенных списков с VLAN'ами:

```
vlans = [[10, 21, 35], [101, 115, 150], [111, 40, 50]]
```

Из этого списка надо сформировать один плоский список с номерами VLAN. Первый вариант – с помощью циклов for:

```
result = []

for vlan_list in vlans:
    for vlan in vlan_list:
        result.append(vlan)


print(result)
# [10, 21, 35, 101, 115, 150, 111, 40, 50]
```

Получить плоский список из нескольких вложенных, генератором списков:

```
vlans = [[10, 21, 35], [101, 115, 150], [111, 40, 50]]
result = [vlan for vlan_list in vlans for vlan in vlan_list]

print(result)
# [10, 21, 35, 101, 115, 150, 111, 40, 50]
```

Соответствие между двумя вложенными циклами и генератором списка с двумя циклами:

```
all_vlans = [[10, 21, 35], [101, 115, 150], [111, 40, 50]]

result = []
for vl_list in all_vlans:
    for vl in vlan_list:
        result.append(vl)

result = [vl for vl_list in all_vlans for vl in vl_list]
```

Можно одновременно проходиться по двум последовательностям, используя `zip`:

```
    vlans = [100, 110, 150, 200]
    names = ['mngmt', 'voice', 'video', 'dmz']

    result = ['vlan {}\n name {}'.format(vlan, name) for vlan, name in
zip(vlans, names)]

    In [26]: print('\n'.join(result))
    vlan 100
     name mngmt
    vlan 110
     name voice
```

```
    vlan 150
     name video
    vlan 200
     name dmz
```

List Comprehensions как генератор

Есть ещё одни способ создания списков, который похож на списковое включение, но результатом работы является не объект класса `list`, а генератор.
Подробно про генераторы написано в “Уроке 15. Итераторы и генераторы“.

Предварительно импортируем модуль sys, он нам понадобится:

```
import sys

# Создадим список, используя списковое включение :
a = [i for i in range(10)]

#проверим тип переменной a:
print(type(a))

# >>> <class 'list'>

# и посмотрим сколько она занимает памяти в байтах:
print(sys.getsizeof(a))

# >>> 192
```

Для создания объекта-генератора, используется синтаксис такой же как и для спискового включения, только вместо квадратных скобок используются круглые:

```
b = (i for i in range(10))
type(b)
# >>> <class 'generator'>
print(sys.getsizeof(b))
# >>> 120
```

Обратите внимание, что тип этого объекта `generator`, и в памяти он занимает места меньше, чем список, это объясняется тем, что в первом случае в памяти хранится весь набор чисел от 0 до 9, а во втором функция, которая будет нам генерировать числа от 0

до 9. Для наших примеров разница в размере не существенна, рассмотрим вариант с 10000 элементами:

```
c = [i for i in range(10000)]
print(sys.getsizeof(c))
# >>> 87624
d = (i for i in range(10000))
print(sys.getsizeof(d))
# >>> 120
```

Сейчас уже разница существенна, как вы уже поняли, размер генератора в данном случае не будет зависеть от количества чисел, которые он должен создать.

Если вы решаете задачу обхода списка, то принципиальной разницы между списком и генератором не будет:

```
for val in a:
    print(val, end=' ')
# >>> 0 1 2 3 4 5 6 7 8 9
for val in b:
    print(val, end=' ')
# >>> 0 1 2 3 4 5 6 7 8 9
```

Но с генератором нельзя работать также как и со списком: нельзя обратиться к элементу по индексу и т.п.

Dict comprehensions

> генераторы словарей

Генераторы словарей аналогичны генераторам списков, но они используются для создания словарей.

Например, такое выражение:

```
d = {}

for num in range(1, 11):
    d[num] = num**2
```

```
print(d)
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Можно заменить генератором словаря:

```
d = {num: num**2 for num in range(1, 11)}

print(d)
# {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Еще один пример, в котором надо преобразовать существующий словарь и перевести все ключи в нижний регистр. Для начала, вариант решения без генератора словаря:

```
r1 = {'ios': '15.4',
      'ip': '10.255.0.1',
      'hostname': 'london_r1',
      'location': '21 New Globe Walk',
      'model': '4451',
      'vendor': 'Cisco'}

lower_r1 = {}

for key, value in r1.items():
    lower_r1[key.lower()] = value

In [35]: lower_r1
Out[35]:
{'hostname': 'london_r1',
 'ios': '15.4',
 'ip': '10.255.0.1',
 'location': '21 New Globe Walk',
 'model': '4451',
 'vendor': 'Cisco'}
```

Аналогичный вариант с помощью генератора словаря:

```
r1 = {'ios': '15.4',
  'ip': '10.255.0.1',
  'hostname': 'london_r1',
  'location': '21 New Globe Walk',
```

```
  'model': '4451',
  'vendor': 'Cisco'}

lower_r1 = {key.lower(): value for key, value in r1.items()}

In [38]: lower_r1
Out[38]:
{'hostname': 'london_r1',
 'ios': '15.4',
 'ip': '10.255.0.1',
 'location': '21 New Globe Walk',
 'model': '4451',
 'vendor': 'Cisco'}
```

Как и list comprehensions, dict comprehensions можно делать вложенными. Попробуем аналогичным образом преобразовать ключи во вложенных словарях:

```
london_co = {
    'r1' : {
    'hostname': 'london_r1',
    'location': '21 New Globe Walk',
    'vendor': 'Cisco',
    'model': '4451',
    'ios': '15.4',
    'ip': '10.255.0.1'
    },
    'r2' : {
    'hostname': 'london_r2',
    'location': '21 New Globe Walk',
    'vendor': 'Cisco',
    'model': '4451',
    'ios': '15.4',
    'ip': '10.255.0.2'
    },
    'sw1' : {
    'hostname': 'london_sw1',
    'location': '21 New Globe Walk',
    'vendor': 'Cisco',
    'model': '3850',
    'ios': '3.6.XE',
    'ip': '10.255.0.101'
    }
}
```

```
    lower_london_co = {}

    for device, params in london_co.items():
        lower_london_co[device] = {}
        for key, value in params.items():
            lower_london_co[device][key.lower()] = value

    In [42]: lower_london_co
    Out[42]:
    {'r1': {'hostname': 'london_r1',
      'ios': '15.4',
      'ip': '10.255.0.1',
      'location': '21 New Globe Walk',
      'model': '4451',
      'vendor': 'Cisco'},
     'r2': {'hostname': 'london_r2',
      'ios': '15.4',
      'ip': '10.255.0.2',
      'location': '21 New Globe Walk',
      'model': '4451',
      'vendor': 'Cisco'},
     'sw1': {'hostname': 'london_sw1',
      'ios': '3.6.XE',
      'ip': '10.255.0.101',
      'location': '21 New Globe Walk',
      'model': '3850',
      'vendor': 'Cisco'}}
```

Аналогичное преобразование с dict comprehensions:

```
    result = {device: {key.lower(): value for key, value in
params.items()}
              for device, params in london_co.items()}

    In [44]: result
    Out[44]:
    {'r1': {'hostname': 'london_r1',
      'ios': '15.4',
      'ip': '10.255.0.1',
      'location': '21 New Globe Walk',
      'model': '4451',
      'vendor': 'Cisco'},
     'r2': {'hostname': 'london_r2',
      'ios': '15.4',
      'ip': '10.255.0.2',
```

```
    'location': '21 New Globe Walk',
    'model': '4451',
    'vendor': 'Cisco'},
  'sw1': {'hostname': 'london_sw1',
    'ios': '3.6.XE',
    'ip': '10.255.0.101',
    'location': '21 New Globe Walk',
    'model': '3850',
    'vendor': 'Cisco'}}
```

Set comprehensions

> генераторы множеств

Генераторы множеств в целом аналогичны генераторам списков.

Например, надо получить множество с уникальными номерами VLAN'ов:

```
vlans = [10, '30', 30, 10, '56']

unique_vlans = {int(vlan) for vlan in vlans}

In [47]: unique_vlans
Out[47]: {10, 30, 56}
```

Аналогичное решение, без использования set comprehensions:

```
vlans = [10, '30', 30, 10, '56']

unique_vlans = set()

for vlan in vlans:
    unique_vlans.add(int(vlan))

In [51]: unique_vlans
Out[51]: {10, 30, 56}
```

Теги: python programing iterators comprehensions

Отредактировать эту страницу

Последнее обновление ***4 сент. 2023 г.*** *от* ***Stavis***