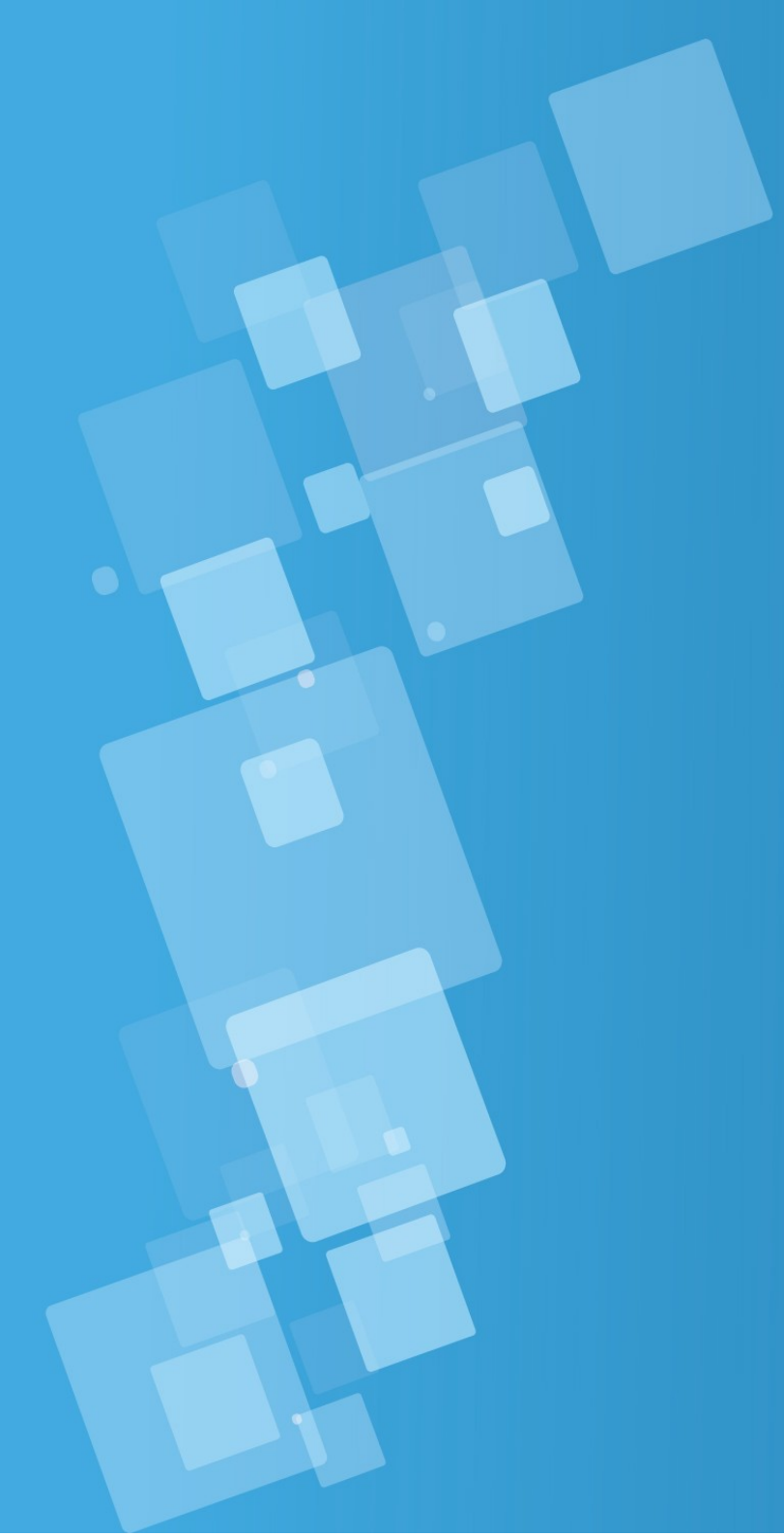

PEP 289 – Generator Expressions

Python List Comprehension

```
[expression(x) for x
in existing_list if
condition(x)]
```

Списковые включения

Списковые включения или генераторы списков – это способ построения нового списка за счет применения **выражения** к каждому элементу в последовательности. который связан с циклом **for** а также инструкции **if-else** для определения того, что в итоге окажется в финальном списке.

Пример

Do this	For this collection	In this situation
`[x**2`	`for x in range(0, 50)`	`if x % 3 == 0]`

Способы формирования списков

1) при помощи циклов
2) при помощи функции map()
3) при помощи list comprehension

1. При помощи цикла for

```
s = []
for i in range(10):
    s.append(i ** 3) # Добавляем к списку куб каждого числа
print(s)
# [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

2. При помощи функции map()

```
list(map(lambda x: x ** 3, range(0,10)))
# [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Лаконично!

3. При помощи конструкции **list comprehension**

```
[x**3 for x in range(10)]
# [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Условие в конце включения

[«возв. значение» for «элемент списка» in «список» **if** «условие»]

```
#Получить все нечетные цифры  в диапазоне от 0 до 9
[x for x in range(10) if x%2 == 1]
#[1, 3, 5, 7, 9]

# Возведение в квадрат
[x**2 for x in range(10)]
```

Условие в начале включения

```
# Замена отрицального диапазона нулем

>>> original_prices = [1.25, -9.45, 10.22, 3.78, -
5.92, 1.16]

>>> prices = [i if i > 0 else 0 for i in
original_prices]

>>> prices

[1.25, 0, 10.22, 3.78, 0, 1.16]
```

Условие в начале включения

```
from string import ascii_letters

letters = 'hытфтrцзqп'  # набор букв из разных алфавитов


# Разграничиваем буквы на английские и не английские

is_eng = [f'{letter}-ДА' if letter in ascii_letters else f'{letter}-НЕТ' for letter in letters]


# ['h-ДА', 'ы-НЕТ', 't-ДА', 'ф-НЕТ', 'т-НЕТ', 'r-ДА', 'ц-НЕТ', 'з-НЕТ', 'q-ДА', 'п-НЕТ']
```

Вызов функции в выражении генераторов

```
# Замена отрицального диапазона нулем

original_prices = [1.25, -9.45, 10.22, 3.78, -5.92,
1.16]
def get_price(price):
    return price if price > 0 else 0


prices = [get_price(i) for i in original_prices]
```

Вложенная генерация

```
# Представим список из слов, который мы хотим привести
к сплошному списку из букв. Двойная итерация: по
словам и по буквам

words = ['Я', 'изучаю', 'Python']

res = [letter for word in words for letter in word]
print(letters)

>>>res
['Я','и','з','у','ч','а','ю','P','y','t','h','o','n']
```

Вложенная генерация

```
key = ["name", "age", "weight"]
value = ["Lilu", 25, 100 ]


[{x, y}  for x in key  for y in value ]


[
{'Lilu', 'name'}, {25, 'name'}, {100, 'name'},
{'Lilu', 'age'}, {25, 'age'}, {100, 'age'},
{'weight', 'Lilu'}, {'weight', 25}, {'weight', 100}
]
```

Вложенная генерация

```
>>> matrix = [[i for i in range(5)] for _ in range(6)]
>>> matrix
[
    [0, 1, 2, 3, 4],
    [0, 1, 2, 3, 4],
    [0, 1, 2, 3, 4],
    [0, 1, 2, 3, 4],
    [0, 1, 2, 3, 4],
    [0, 1, 2, 3, 4]
]
```

Внешний генератор [... for _ in range(6)] создает 6 строк в то время как внутренний генератор[i for i in range(5)]заполняет каждую строку значениями.

Вложенная генерация

```
# Преобразование матрицы в плоский вид

matrix = [
...     [0, 0, 0],
...     [1, 1, 1],
...     [2, 2, 2],
... ]
>>> flat = [col for row in matrix for col in row]
>>> flat
[0, 0, 0, 1, 1, 1, 2, 2, 2]
```

Вложенная генерация

```
# Генерация таблицы умножения от 1 до 5

T =[[x*y for x in range(1, 6)] for y in range(1, 6)]

print(T)
[[1, 2, 3, 4, 5],
 [2, 4, 6, 8, 10],
 [3, 6, 9, 12, 15],
 [4, 8, 12, 16, 20],
 [5, 10, 15, 20, 25]]
```

Когда использовать генератор списков ?

- Использовать для выполнения простых фильтраций, модификаций или форматирования итерируемых объектов.
- Для увеличение производительности.
- Для компактности
- Следует избегать использования генератора списков, если вам нужно добавить слишком много условий это делает код трудным для чтения.

Python

Dictionary Comprehension

Генераторы словарей

Генерация словаря похожа на генерацию списка и предназначена для создания словаря.

```
d = {}
for num in range(1, 10):
    d[num] = num**2
print(d)
```

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

```
D = { num: num**2 for num in range(1, 10)}
>>>d
```

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

Генераторы словарей

```
#Созданим словарь по списку кортежей

items = [('c', 3), ('d', 4), ('a', 1), ('b', 2)]

dict_variable = { key:value for (key,value) in items }

print(dict_variable)
```

Что если не будет значения :value ?

Set comprehensions!

Условие if

```
# Добавим в конструкцию генератора условие фильтрации

dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
# Проверка, больше ли элемент, чем 2

filtered = {k:v for (k,v) in dict1.items() if v>2}

print(filtered)
# {'e': 5, 'c': 3, 'd': 4}
```

Условие if

Фильтрация по возрасту

```
ages = {
    'kevin': 12,
    'marcus': 9,
    'evan': 31,
    'nik': 31
}
f = {k:val for (k, val) in ages.items() if val > 25}
print(new_ages)
```

Несколько условий if

#Последовательные операторы if работают так, как если бы между ними были логические **and**.

```
dict = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
r = {k:v for (k,v) in dict.items() if v>2 if v%2 == 0}
print(r)
# {'d': 4}
```

Вложенные генераторы словарей

```
dict = {'first':{'a':1}, 'second':{'b':2}}
fd = {o_key: {float(i_val) for (i_key, i_val) in o_val.items()}
for (o_key, o_val) in dict.items()}
print(fd)
# {'first': {1.0}, 'second': {2.0}}
```

Код имеет вложенный генератор словаря, то есть один генератор внутри другого. Как видите, вложенный генератор словаря может быть довольно трудным как для чтения, так и для понимания. Использование генераторов при этом теряет смысл (ведь мы их применяем для улучшения читабельности кода).

Использование enumerate функции

```
names = ['Harry', 'Hermione', 'Ron', 'Neville', 'Luna']
index = {k:v for (k, v) in enumerate(names)}
print(index)
{'Harry':0, 'Hermione':1, 'Ron':2, 'Neville':3, 'Luna':4}
```

Когда использовать генераторы словарей?

Во всех случаях что и при генерации списков.

Заключение

Подобные конструкции позволяют создавать не только списки (list comprehension) и словари (dictionary comprehension), генераторы (generator expression – при помощи «()»), а также множества (set comprehension – при помощи «{}» и картежи «tuple()»). Принцип везде один и тот же.