Генераторы списков в Python

List Comprehension

Разберем несколько способов создания генератора списков и увидим некоторые их вариации, например:

1. Простой генератор списка
2. Генераторы списков с одиночными и вложенными условиями if
3. Генератор списка с одним и несколькими условиями if и else
4. Генератор списков с вложенными циклами for

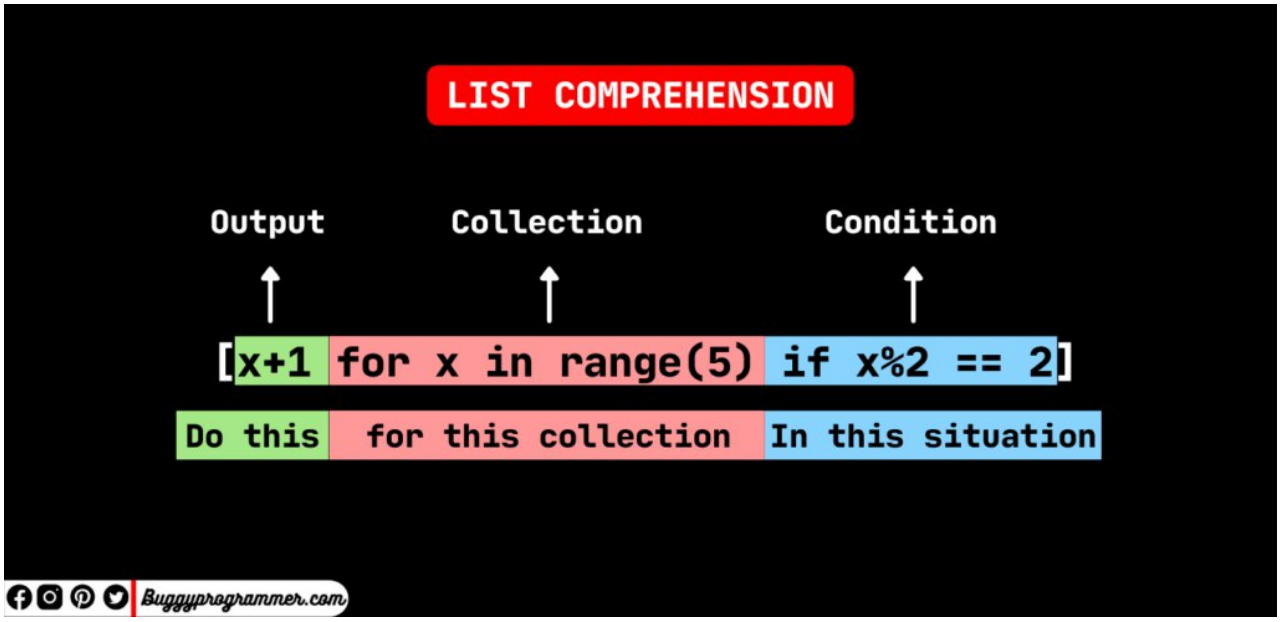
Помимо этого, мы также рассмотрим следующие концепции:

- Цикл for vs. генератор списка
- Каковы преимущества генератора списка
- Когда использовать, а когда лучше избегать генератора списков

Что же такое генераторы списков в Python?

Итак, начнем с синтаксиса генератора списка. Генератор списка – это одна строка кода, которую вы пишете в квадратных скобках. Он состоит из трех компонентов:

- цикл for
- условие и выражение (condition)
- результат (output)



Пример простого генератора списка

Приведенный ниже фрагмент кода является примером простейшего генератора списка. Здесь мы просто перебираем lst и сохраняем все его элементы в списке a:

```
lst = [1,2,3,4,5,6,7,8,9,10]
```

```
# простой генератор списка
a = [x for x in lst]
print(a)

# Результат:
# [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Этот код полностью эквивалентен следующему:

```
for x in lst:
    a.append(x)
```

Но в первом случае для создания списка a нам даже не нужен метод append. Вместо этого мы используем генератор.

Идем дальше. В приведенном выше генераторе списка можно использовать любое выражение для изменения исходных элементов lst, например:

```
# добавить любое число к каждому элементу lst и сохранить результат в a
a = [x+1 for x in lst]

# вычесть любое число из каждого элемента lst и сохранить в a
a = [x-1 for x in lst]

# умножить каждый элемент lst на любое число и сохранить в  a
a = [x*2 for x in lst]
```

Генератор списка с одиночным и вложенным условием if

В генератор списка также можно добавить if-условие, которое может помочь нам отфильтровать данные. Например, в приведенном ниже коде мы сохраняем в список c все значения lst, большие 4 :

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
c = [x for x in lst if x > 4]
print(c)

# Результат:
# [5, 6, 7, 8, 9, 10]
```

Этот код выполняет то же самое, что и приведенный ниже:

```
for x in lst:
    if x > 4:
        a.append(x)
```

Мы также можем добавить в наш генератор списка вложенное условие if. Например, в приведенном ниже коде мы сохраняем в список d все элементы lst, значения которых больше 4 и кратны 2 :

```
d = [x for x in lst if x > 4 if x%2 == 0]

# Результат:
# [6, 8, 10]
```

Этот код эквивалентен данному:

```
for x in lst:
    if x > 4:
        if x % 2 == 0:
            a.append(x)
```

Генератор списка с одним и несколькими условиями if и else

Хорошо, теперь давайте посмотрим, как мы можем добавить else после if в генератор списка. Напишем простой генератор списка, который будет сохранять в список e все значения lst, большие 4. Если же какое-то значение меньше 4, вместо него будет сохранена строка less than 4.

```
lst = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# с условиями if и else
e = [x if x > 4 else 'less than 4' for x in lst]
print(e)

# Результат:
# ['less than 4', 'less than 4', 'less than 4', 'less than 4', 5, 6, 7, 8, 9, 10]
```

Следующий код выполняет ту же задачу:

```
for x in lst:
    if x > 4:
        d.append(x)
    else:
        d.append('less than 4')
```

А теперь давайте посмотрим, как работают генераторы списков с несколькими if и else.

В приведенном ниже примере мы сохраняем строку Two, если значение кратно 2. А если значение кратно 3, мы сохраняем Three. Во всех остальных случаях мы сохраняем not 2 & 3.

```
f = ['Two' if x%2 == 0 else "Three" if x%3 == 0 else 'not 2 & 3' for x in lst]
print(f)

# Результат:
# ['not 2 & 3', 'Two', 'Three', 'Two', 'not 2 & 3', 'Two', 'not 2 & 3', 'Two', 'Three', 'Two']
```

Как это работает? Чтобы понять это, мы можем разделить всё условие на три части, после каждого else:

'Two' if x%2 == 0 else "Three" if x%3 == 0 else 'not 2 & 3'

Таким образом, если первое условие if истинно, тогда элемент будет принимать значение Two – в противном случае мы вместо сохранения какого-либо значения перейдем ко второму условию if. По такому же принципу работает команда elif. Во втором if-условии в элемент сохранится Three, если утверждение истинно. В противном случае программа проверит следующее условие. Этого условия у нас нет, поэтому элементу будет присвоено значение, идущее сразу после else, — not 2 & 3.

Мы можем добиться того же результата, написав код следующим образом:

```
for x in lst:
    if x%2 == 0:
        f.append('Two')
    elif x%3 == 0:
        f.append('Three')
    else:
        f.append('not 2 & 3')
```

Теперь-то вы видите силу генератора списков? Он выполняет задачу всего в одной строке, в то время как традиционный цикл for состоит из 7.

Генератор списка с вложенным циклом for

Хорошо! Теперь давайте разберем использование вложенного цикла for в генераторе списка.

Чтобы понять, как это работает, давайте рассмотрим приведенный ниже пример. Здесь мы генерируем все возможные комбинации элементов двух списков: [1, 2, 3] и [3, 2, 1].

```
lst = [1,2,3]
lst_rev = [3,2,1]
g = [(x,y) for x in lst for y in lst_rev]
print(g)

# Результат:
```

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

Традиционным способом эта задача решалась бы так:

```
for x in lst:
    for y in lst_rev:
        f.append((x,y))
```

Хорошо, а теперь, как и обещали, давайте сравним обычный цикл for и генератор списков.

Циклы vs. генератор списков

Выше мы видели, как генератор списков позволяет выполнять задачу всего в одну строчку, в то время как цикл for требует написания нескольких строк.

Генератор списков не только более компактен, но также его эффективность выше. В некоторых случаях он оказывается в два раза быстрее, чем цикл for.

Однако если вы хотите выполнить более одного простого условия, генератор списков не сможет справиться с этим без ущерба для удобочитаемости. Это одна из его основных проблем.

Преимущества генераторов списков

Генератор списков — не только простое, компактное и быстрое, но и надежное решение во многих ситуациях. Его можно использовать в самых разных обстоятельствах. Например, для сопоставления и фильтрации в дополнение к генерации базового списка. Вам не нужно каждый раз изобретать велосипед. Это одна из причин, по которой генераторы списков считаются более «питоничными», чем цикл for.

Когда использовать генератор списков (а когда его лучше избегать)

Вы можете использовать генератор списков, если выполняете простую фильтрацию, модификации или форматирование итерируемых объектов. Он также будет хорошим выбором, если вы хотите, чтобы ваш код был компактным и читабельным.

Кроме того, вы можете использовать генератор, когда для вас важно даже небольшое увеличение производительности.

Однако следует избегать использования генератора списков, если вам нужно добавить слишком много условий для фильтрации или модификации, поскольку это сделает ваш код излишне сложным и трудным для чтения.