**Генераторы в Python**

**Цель:**

1. Научиться использовать генераторы списков и словарей для быстрого создания коллекций с применением различных трансформаций и фильтраций.
2. Научиться создавать и применять генераторы для последовательной обработки потока информации, экономии памяти.

**Задания на тему генераторы списков и словарей:**

1. Создайте список всех четных чисел от 0 до 20 включительно, используя генератор списка.

Ожидаемый результат: [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

1. Создайте список квадратов чисел от 1 до 10.

Ожидаемый результат: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

1. У вас есть два списка: ['a', 'b', 'c'] и [1, 2, 3]. Создайте словарь, используя элементы первого списка как ключи, и второго — как значения.

Ожидаемый результат:{'a': 1, 'b': 2, 'c': 3}

1. Создайте словарь, в котором ключами будут числа от 1 до 5, а значениями — их квадраты.

Ожидаемый результат: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

**Задания на тему генераторов в Python.**

1. Создайте простой генератор, который возвращает квадраты чисел от 1 до n.
2. Создайте генератор для обработки файла log.txt. В результате создайте новый текстовый файл, в котором будут содержаться все записи с ошибками.
3. Создайте функцию генератор для вычисления числа Фибоначчи.
4. Создайте генератор, который принимает два списка и чередует их элементы.

Теоретические сведения

# Генераторы

В Python просто генераторы и генераторы списков - разные вещи. Здесь есть проблема перевода с английского. То, что мы привыкли называть генератором списка, в английском варианте звучит как "**list comprehension**" и к генераторам никакого отношения не имеет.

Слово "comprehension" (понимание, осмысление) оказывается как бы не в тему при переводе на русский. Получается что-то вроде "понимание списка". Поэтому мы говорим "генератор списка", понимая под словом "генератор" не объект, а синтаксическую конструкцию, которая генерирует, то есть создает, список.

С другой стороны, объекты-генераторы - это особые объекты-функции, которые между вызовами сохраняют свое состояние. В цикле for они ведут себя подобно итерируемым объектам, к которым относятся списки, словари, строки и др. Однако генераторы поддерживают метод \_\_next\_\_(), а значит являются разновидностью итераторов.

Быстрым способом создания относительно простых объектов-генераторов являются генераторные выражения - **generator expressions**. Синтаксис этих выражений похож на синтаксис генераторов списков. Однако они возвращают разные типы объектов. Первый - объект-генератор. Второй - список.

Сначала рассмотрим генераторы списков, чтобы привыкнуть к синтаксической конструкции.

## Генераторы списков

В Python генераторы списков позволяют создавать и быстро заполнять списки.

Синтаксическая конструкция генератора списка предполагает наличие итерируемого объекта или итератора, на базе которого будет создаваться новый список, а также выражение, которое будет что-то делать с извлеченными из последовательности элементами перед тем как добавить их в формируемый список.

>>> a = [1, 2, 3]

>>> b = [i+10 **for** i **in** a]

>>> a

[1, 2, 3]

>>> b

[11, 12, 13]

В примере выше генератором списка является выражение [i+10 for i in a]. Здесь a - итерируемый объект. В данном случае это другой список. Из него извлекается каждый элемент в цикле for. Перед for описывается действие, которое выполняется над элементом перед его добавлением в новый список.

Обратите внимание, что генератор создает новый список, а не изменяет существующий. Если надо изменить текущую переменную, ей надо присвоить новое значение:

>>> a = [1, 2, 3]

>>> a = [i+10 **for** i **in** a]

>>> a

[11, 12, 13]

Генераторы списков относятся к разряду "синтаксического сахара" языка программирования Python. Другими словами, без них можно обойтись:

>>> **for** index, value **in** enumerate(a):

... a[index] = value + 10

...

>>> a

[11, 12, 13]

Если в программе может быть несколько ссылок на список, генераторами надо пользоваться осторожно:

>>> ls0 = [1,2,3]

>>> ls1 = ls0

>>> ls1.append(4)

>>> ls0

[1, 2, 3, 4]

>>> ls1 = [i+1 **for** i **in** ls1]

>>> ls1

[2, 3, 4, 5]

>>> ls0

[1, 2, 3, 4]

Здесь мы предполагаем, что изменение списка через одну переменную, будут видны через другую. Однако если изменить список генератором, то переменные будут указывать на разные списки.

Перебираемым в цикле for объектом может быть быть не только список. В примере ниже в список помещаются строки файла.

>>> lines = [line.strip() **for** line **in** open('text.txt')]

>>> lines

['one', 'two', 'three']

В генератор списка можно добавить условие:

>>> **from** random **import** randint

>>> nums = [randint(10, 20) **for** i **in** range(10)]

>>> nums

[18, 17, 11, 11, 15, 18, 11, 20, 10, 19]

>>> nums = [i **for** i **in** nums **if** i%2 == 0]

>>> nums

[18, 18, 20, 10]

Генераторы списков могут содержать вложенные циклы:

>>> a = "12"

>>> b = "3"

>>> c = "456"

>>> comb = [i+j+k **for** i **in** a **for** j **in** b **for** k **in** c]

>>> comb

['134', '135', '136', '234', '235', '236']

## Генераторы словарей и множеств

Если в выражении генератора списка заменить квадратные скобки на фигурные, то можно получить не список, а словарь:

>>> a = {i:i\*\*2 **for** i **in** range(11,15)}

>>> a

{11: 121, 12: 144, 13: 169, 14: 196}

При этом синтаксис выражения до for должен быть соответствующий словарю, то есть включать ключ и через двоеточие значение. Если этого нет, будет сгенерировано множество:

>>> a = {i **for** i **in** range(11,15)}

>>> a

set([11, 12, 13, 14])

>>> b = {1, 2, 3}

>>> b

set([1, 2, 3])

## Генераторы

Выражения, создающие объекты-генераторы, похожи на выражения, генерирующие списки, словари и множества за одним исключением. Чтобы создать генераторный объект, надо использовать круглые скобки:

>>> a = (i **for** i **in** range(2, 8))

>>> a

<generator object <genexpr> at 0x7efc88787910>

>>> **for** i **in** a:

... print(i)

...

2

3

4

5

6

7

Второй раз перебрать генератор в цикле for не получится, так как объект-генератор уже сгенерировал все значения по заложенной в него "формуле". Поэтому генераторы обычно используются, когда надо единожды пройтись по итерируемому объекту.

Кроме того, генераторы экономят память, так как в ней хранятся не все значения, скажем, большого списка, а только предыдущий элемент, предел и формула, по которой вычисляется следующий элемент.

Выражение, создающее генератор, это сокращенная запись следующего:

>>> **def** **func**(start, finish):

... **while** start < finish:

... **yield** start \* 0.33

... start += 1

...

>>> a = func(1, 4)

>>> a

<generator object func at 0x7efc88787a50>

>>> **for** i **in** a:

... print(i)

...

0.33

0.66

0.99

Функция, содержащая yield возвращает объект-генератор, а не выполняет свой код сразу. Тело функции исполняется при каждом вызове метода \_\_next\_\_(). В цикле for это делается автоматически. При этом функция сохраняет значения переменных от предыдущего вызова.

Если нет необходимости использовать функцию многократно, проще использовать выражение:

>>> b = (i\*0.33 **for** i **in** range(1,4))

>>> b

<generator object <genexpr> at 0x7efc88787960>

>>> **for** i **in** b:

... print(i)

...

0.33

0.66

0.99