Лекция 6. Генераторы. Ламбда-функции. Начала аналитики

Модуль random и функция shuffle: перемешивание списков.

Функции map и filter: обработка списков без циклов.

Функция sum.

List comprehension вместо filter. Циклы без циклов.

Ламбда-функции - создание функций "на лету"

Генераторы

Принцип работы генератора.

List comprehension - циклы без циклов

Генераторы

Генераторы позволяют создавать простые последовательности «на лету» максимально простым кодом.

Разница в том, что в памяти хранится не вся последовательность, а только текущее значение.

Генератором называется любая функция, содержащая ключевое слово **yield**.

Yield

Yield - ключевое слово в Python, которое используется для возврата из функции **с сохранением состояния локальных переменных** этой функции.

При повторном вызове такой функции **выполнение продолжается с оператора yield**, на котором ее работа была прервана.

(При вызове обычной функции ее выполнение начинается «с нуля», состояние сбрасывается)

Генераторы (пример)

```
def create_range(start, end, step=1):
    while start < end:
        yield start
        start += step

print(create_range(1, 4, 1))

for i in create_range(1, 4):
  print("yield:", i)
```

```
my_range = range(1, 4, 1)
print(list(my_range))
```

```
range(1, 4)
[1, 2, 3]
```

```
<generator object create_range at 0x7fa9000689e0>
yield: 1
yield: 2
yield: 3
```

List comprehension

List comprehension

List comprehension offers a shorter syntax when you want **to create a new list based on the values of an existing list.**

comprehension: варианты перевода

Имя существительное		Частота
понимание	understanding, insight, comprehension, conception, realization, grasp	
осмысление	comprehension	
постижение	comprehension, osmose	
охват	coverage, sweep, reach, envelopment, incidence, comprehension	
разумение	comprehension	
понятливость	intelligence, comprehension, docility, apprehensiveness, perceptivity, sagacity	
включение	inclusion, insertion, engagement, comprehension, switching-on	

Создание последовательности **на базе** другой последовательности. Без **явного использования** циклов

List comprehension (продолжение)

```
# list comprehension - цикл без цикла
my_list = [x for x in range(1, 4, 1)]
print(my_list)
```

существующая последовательность

list comprehension

```
[1, 2, 3]
```

List comprehension с фильтрацией vs генератор

```
# стандартный алгоритм фильтрации целых чисел - мы все сделали вручную
test_list = list(range(10))
def create_filter(some_list):
  for el in some_list:
    if el % 2 == 0:
      yield el

print("yield с фильтром: ", list(create_filter(test_list)))
```

```
yield с фильтром:  [0, 2, 4, 6, 8]
```

```
# list comprehension
# все то же самое, только код намного короче.
# на самом деле внутри тоже зашит for...
my_list = [x for x in range(10) if x%2==0]
print(my_list)
```

```
[0, 2, 4, 6, 8]
```

Функции map и filter для списков

map – «отображение» операции на все элементы последовательности.

filter – фильтрация последовательности

Функция map

Применяет определенную функцию к каждому элементу в последовательности.

map(function, iterable, ...)

Параметры:

function - пользовательская функция, вызывается для каждого элемента,

iterable - последовательность или объект, поддерживающий итерирование.

Возвращаемое значение:

map object – итератор map

Функция map (продолжение)

```
def add_fn(*args):
    sum = 0
    for arg in args:
        sum += arg
    return sum

x = map(add_fn, [1, 2], [3, 4], [5, 6])
print(list(x))
```

```
[9, 12]
```

Функция map (продолжение)

Применяет определенную функцию к каждому элементу в последовательности.

map(function, iterable, ...)

- **В момент обработки в памяти находится только текущий элемент последовательности!**
- Поэтому – более низкое потребление памяти.
- А еще говорят, что она работает быстрее, чем for

Функция map (продолжение)

Примеры использования map:

- Сложение в последовательности
- Создание словаря из кортежей
- Как вычистить текст от знаков препинания

Функция filter

Фильтрует всю последовательность по заданному условию.

filter(function, iterable, ...)

Параметры:

function - функция, которая принимает элемент фильтруемого объекта, и должна вернуть bool значение,

iterable - последовательность или объект, поддерживающий итерирование.

Возвращаемое значение:

filter object – отфильтрованная последовательность

Функция filter (продолжение)

```
def filter_odd(x):
    if x%2 != 0:
        return x
odd_list = list(filter(filter_odd, numbers))
print(f"Все нечетные: {odd_list}")
```

```
Все нечетные: [1, 3, 5, 7, 9]
```

Функция filter (продолжение)

Фильтрует всю последовательность по заданному условию.

filter(function, iterable, ...)

- **В момент обработки в памяти находится только текущий элемент последовательности!**
- Нужно создать фукнцию, выполняющую алгоритм фильтрации

Функция filter и lambda-функции

```
numbers = list(range(10))
even_list = list(filter(lambda x: x%2 == 0, numbers))
print(f"Все четные: {even_list}")
```

```
Все четные: [0, 2, 4, 6, 8]
```

Функция sum

Суммирование элементов последовательности.

Объединение многомерных последовательностей

Функция sum

1) Суммирует все элементы последовательности

sum(iterable, start)

Параметры:

iterable - последовательность или объект, поддерживающий итерирование,

start – необязательный параметр. Начальное значение, которое прибавится к сумме

Возвращаемое значение:

int, float, complex – число, тип которого зависит от того, что суммируем

Функция sum (продолжение)

2) Объединяет несколько последовательностей

sum(iterable, start)

Параметры:

iterable - последовательность или объект, поддерживающий итерирование,

start – пустая последовательность нужного типа (list, tuple)

Возвращаемое значение:

объединенная последовательность, тип которой зависит от того, что подавалось на вход

Функция sum (продолжение)

Примеры использования map:

- Сложение элементов – int, float, complex
- Объединение последовательности (матрица)
- Расчет скалярного произведения векторов

Интересное: sum и ее альтернативы с примерами https://pythonist.ru/funkcziya-sum-v-python/

lambda-функции

Как устроена и работает лямбда-функция

Объявление лямбда-функций

Совместное использование лямбда-фунций, map и filter

lambda-функции

lambda-функция - это функция, которая:

- ❑ не имеет имени
- ❑ создается непосредственно в месте использования и сразу уничтожается
- ❑ записывается в одну строку
- ❑ всегда возвращает значение, но без ключевого слова return

```
lambda arguments: expression
```

```
f = lambda x, y: x * y # произведение двух чисел
```

lambda-функции (продолжение)

Примеры использования:

- Проверка числа на четность
- Вычислить квадраты чисел (+ map)
- Вычислить квадраты только четных чисел (+ map)
- Вычислить квадраты только четных чисел и без None (+ map, + filter)