

Итераторы и итерируемые объекты. Генераторы.

Что такое итерируемый объект и итератор

Итератор – любой объект, реализующий метод `__next__`, который возвращает следующий элемент в очереди или выбрасывает исключение `StopIteration`, если не осталось элементов.

Итерируемый объект – любой объект, реализующий метод `__iter__` или `__getitem__`. Итерируемым объектом является любая коллекция: список, кортеж, словарь, и т.д.

Цель итерируемого объекта – создать итератор.

Для этого у него есть метод `__iter__`, при каждом обращении к которому создается новый итератор.

Цель итератора – пройти по элементам.

Для этого у него есть метод `__next__`, который возвращает элементы один за другим.

Как работает цикл for

В большинстве случаев, при обработке итерируемого объекта используется цикл for:

```
>>> items = [1, 2, 3]
>>> for item in items:
...     print(item)
...
1
2
3
```

Но если бы не было цикла for, то для эмуляции его работы, пришлось бы написать такой код:

```
>>> items = [1, 2, 3]
>>> it = iter(items)
>>> while True:
...     try:
...         print(next(it))
...     except StopIteration:
...         break
...
1
2
3
```

Как работает цикл for

```
1 items = [1, 2, 3]
2 it = iter(items) # Получаем итератор
3
4 # Используем итератор
5 next(it) # вызывает метод __next__
6 next(it) # 2
7 next(it) # 3
8 next(it) # Traceback
```

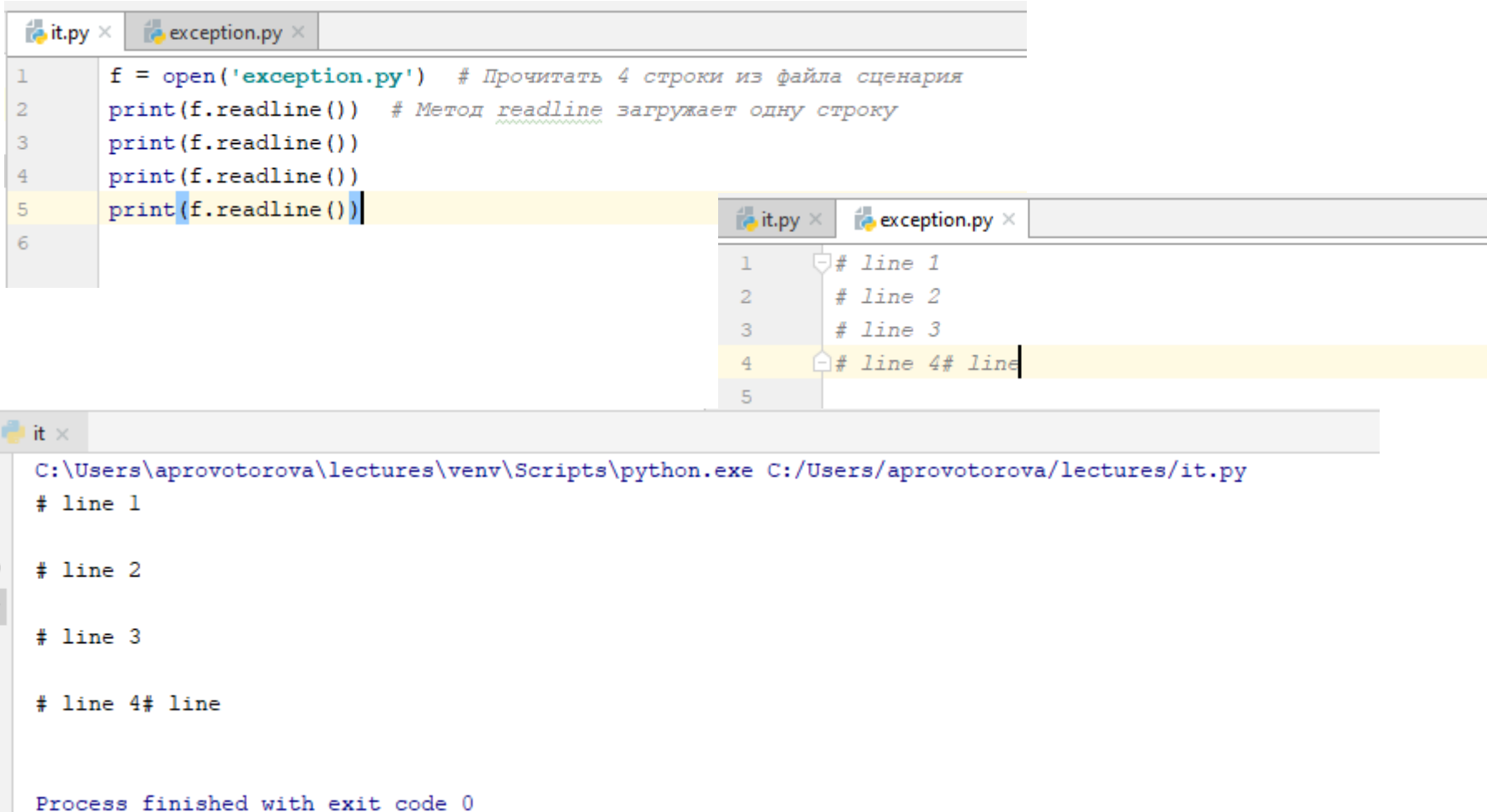
```
Traceback (most recent call last):
  File "C:/Users/aprovotorova/lectures/it.py", line 8, in <module>
    next(it) # Traceback
StopIteration
```

Шаблон проектирования “Итератор”

Назначение:

- для доступа к содержимому агрегированных объектов без раскрытия их внутреннего представления
- для поддержки нескольких активных обходов одного и того же агрегированного объекта (желательно, но не обязательно)
- для предоставления единообразного интерфейса с целью обхода различных агрегированных структур.

Протокол итераций: итераторы файлов



The image shows a Python IDE with two windows. The top window, titled 'exception.py', contains a script that opens a file named 'exception.py' and reads its contents line by line using the `readline()` method. The script is as follows:

```
1 f = open('exception.py') # Прочитать 4 строки из файла сценария
2 print(f.readline()) # Метод readline загружает одну строку
3 print(f.readline())
4 print(f.readline())
5 print(f.readline())
6
```

The bottom window, titled 'it.py', shows the output of the script. It displays the contents of the file 'exception.py' line by line, as read by the script. The output is:

```
1 # line 1
2 # line 2
3 # line 3
4 # line 4# line
5
```

At the bottom of the IDE, a status bar indicates that the process finished with exit code 0.

SimpleIterator

Если нужно обойти элементы внутри объекта вашего собственного класса, необходимо построить свой итератор. Создадим класс, объект которого будет итератором, выдающим определенное количество единиц, которое пользователь задает при создании объекта.

Такой класс будет содержать конструктор, принимающий на вход количество единиц и метод `__next__()`, без него экземпляры данного класса не будут итераторами.

Simple Iterator

```
1 class SimpleIterator(object):
2
3     def __init__(self, limit):
4         self.limit = limit
5         self.counter = 0
6
7     def __iter__(self):
8         return self
9
10    def __next__(self):
11        if self.counter < self.limit:
12            self.counter += 1
13            return 1
14        else:
15            raise StopIteration
16
17
```


Генераторы

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.

Генераторы поддерживают метод `__next__()`, а значит являются разновидностью итераторов.

Генераторы списков

```
1 a = [1, 2, 3]
2 b = [i + 10 for i in a]
3
4 print(b)
```

```
1 from random import randint
2
3 nums = [randint(10, 20) for i in range(10)]
4
5 print(nums) # 16, 13, 17, 14, 14, 12, 18, 19, 18, 10]
```

```
1 a = "12"
2 b = "3"
3 c = "456"
4 comb = [i + j + k for i in a for j in b for k in c]
5 print(comb) # ['134', '135', '136', '234', '235', '236']
```

```
1 res = [x + y for x in [0, 1, 2] for y in [100, 200, 300]]
2
3 print(res) # [100, 200, 300, 101, 201, 301, 102, 202, 302]
4
5 res1 = []
6 for x in [0, 1, 2]:
7     for y in [100, 200, 300]:
8         res1.append(x + y)
9
10 print(res1) # [100, 200, 300, 101, 201, 301, 102, 202, 302]
```

Генераторы

- Функции-генераторы – выглядят как обычные инструкции *def*, но для возврата результатов по одному значению за раз используют инструкцию *yield*, которая приостанавливает выполнение функции.
- Выражения-генераторы – напоминают генераторы списков, о которых рассказывалось в предыдущем разделе, но они не конструируют список с результатами, а возвращают объект, который будет воспроизводить результаты по требованию.

Функции-генераторы: инструкция yield вместо return

- В отличие от обычных функций, которые возвращают значение и завершают работу, функции-генераторы автоматически приостанавливают и возобновляют свое выполнение, при этом сохраняя информацию, необходимую для генерации значений.
- Главное отличие функций-генераторов от обычных функций состоит в том, что функция-генератор *поставляет* значение, а не *возвращает* его – инструкция yield приостанавливает работу функции и передает значение вызывающей программе

Пример функции-генератора

```
1 def gen_squares(n):  
2     for i in range(n):  
3         yield i ** 2  
4  
5  
6     for i in gen_squares(5):  
7         print(i, end=', ')  
8  
9     # 0, 1, 4, 9, 16  
10
```


Генераторы предлагают лучшее решение с точки зрения использования памяти и производительности. Они дают возможность избежать необходимости выполнять всю работу сразу, что особенно удобно, когда список результатов имеет значительный объем или когда вычисление каждого значения занимает продолжительное время.

Выражения-генераторы: итераторы и генераторы списков

```
>>> [x ** 2 for x in range(4)]    # Генератор списков: создает список  
[0, 1, 4, 9]
```

```
>>> (x ** 2 for x in range(4))    # Выражение-генератор: создает  
<generator object at 0x011DC648> # итерируемый объект
```

Thanks for your
attention

 [artezio_software](#)

 info@artezio.com

www.artezio.com