Итераторы

№ урока: 5 **Курс:** Python Essential

Средства обучения: PyCharm

Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление о механизме итераторов, научатся создавать собственные итераторы.

Изучив материал данного занятия, учащийся сможет:

- Иметь представление о внутренних механизмах работы цикла for в Python
- Создавать и использовать итераторы

Содержание урока

1. Что такое итератор?
2. Итерирование через итератор
3. Как цикл for работает с итераторами?
4. Создание собственных итераторов

Резюме

- *Контейнер* – это тип данных, который инкапсулирует в себе значения других типов.
- *Итерабельный объект* (в оригинальной терминологии – существительное «*iterable*») – это объект, который может возвращать значения по одному за раз. Примеры: все контейнеры и последовательности (списки, строки и т.д.), файлы, а также экземпляры любых классов, в которых определён метод __iter__() или __getitem__().
- Метод __iter__() возвращает объект-итератор. Метод __getitem__() возвращает элемент контейнера по ключу или индексу.
- Итерабельные объекты могут быть использованы внутри цикла for, а также во многих других случаях, когда ожидается последовательность (функции sum(), zip(), map() и т.д.).
- Когда итерабельный объект передаётся во встроенную функцию iter(), она возвращает итератор для данного объекта, который позволяет один раз пройти по значениям итерабельного объекта.
- При использовании итерабельных объектов, обычно не нужно вызывать функцию iter() или работать с итераторами напрямую, так как цикл for делает это автоматически.
- *Итератор (iterator)* – это объект, который представляет поток данных. Повторяемые вызовы метода __next__() (next() в Python 2) итератора или передача его встроенной функции next() возвращает последующие элементы потока.
- Если больше не осталось данных, выбрасывается исключение StopIteration. После этого итератор исчерпан и любые последующие вызовы его метода __next__() снова генерируют исключение StopIteration.
- Итераторы обязаны иметь метод __iter__, который возвращает сам объект итератора, так что любой итератор также является итерабельным объектом и может быть использован почти везде, где принимаются итерабельные объекты. Одним из исключений является код, который проходит по итератору несколько раз. Контейнеры (например, список), каждый раз создают новый итератор каждый раз, когда они передаются в функцию iter() или используются в цикле for. Попытка сделать это с итератором вернёт исчерпанный итератор из предыдущего цикла, и он будет выглядеть, как пустой контейнер.

План занятия:

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Итераторы в Python повсюду. Они элегантно встроенные в for циклы, включения, генераторы и так далее, но зачастую мы их просто не замечаем. Большинство встроенных типов данных итерабельные "из коробки": списки, кортежи, строки. Каждый из этих типов мы можем "перебрать" в цикле.
Итераторы в Python — это обычные объекты, которые можно итерировать. Такой объект будет возвращать по одному элементу данных за один раз.
Технически, итератор должен вмещать в себе два специальных метода: __iter__() и __next__(), которые обычно называются протоколом итератора.
Объект называется итерабельным, если мы можем получить из него итератор. В Python есть встроенная функция iter(), которая возвращает итератор объекта.
На самом деле, функция iter() просто вызывает метод __iter__() объекта.

Итерирование через итератор
Мы используем функцию next(), чтобы вручную итерировать все элементы итератора. Когда мы получим все элементы и итератор будет пуст, мы получим исключение StopIteration:

```
# Создаем список
my_list = ["One", "piece", "per", "time"]

# Получаем итератор с помощью iter()
my_iter = iter(my_list)

# Итерируем обьект с помощью next()

# Вывод: One
print(next(my_iter))

# next(obj) is same as obj.__next__()

# Вывод: piece
print(my_iter.__next__())

# Вывод: per
print(my_iter.__next__())

# Вывод: time
print(my_iter.__next__())

# Следующий вызов выбросит исключение StopIteration, так как элементы закочнились
next(my_iter)
One
piece
per
time
Traceback (most recent call last):
  File "temp.py", line 27, in <module>
    next(my_iter)
StopIteration
```

Мы уже знакомы со способом итерировать объект более элегантно с помощью цикла for. С его помощью мы можем итерировать любой объект, который умеет возвращать итератор: списки, строки, файлы и так далее:

```
my_list = ["One", "piece", "per", "time"]

for item in my_list:
    print(item)
```

Как цикл for работает с итераторами?
Цикл for умеет автоматически выполнять итерацию по списку. Фактически цикл for может выполнять итерацию любого итератора. Давайте подробнее рассмотрим, как на самом деле реализован цикл for в Python.

```
for element in iterable:
    pass
```


CyberBionic systematics
CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine
Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

```
    # какие-то операции над элементом
```

На самом деле пример выше аналогичен следующему:

```
# получение итератора итерабельного объекта
iter_obj = iter(iterable)

# безсконечный цикл
while True:
    try:
        # получение следующего елемента
        element = next(iter_obj)
        # какие-то операции над элементом
    except StopIteration:
        # Если получили StopIteration, выходим из цикла
        break
```

Таким образом, внутри цикла for создается объект-итератор iter_obj путем вызова iter() для итерируемого объекта.
Фактически, цикл for на самом деле является бесконечным циклом while.
Внутри цикла он вызывает next() для получения следующего элемента и выполняет тело цикла for с этим значением. После того как все элементы исчерпаны, вызывается исключение StopIteration, которое перехватывается внутри, и цикл заканчивается. Обратите внимание, что любые другие исключения не будут обработаны.

Создание собственных итераторов
Создать итератор в Python просто. Нам просто нужно реализовать методы __iter__() и __next__().
Метод __iter__() возвращает сам объект итератора. При необходимости можно выполнить некоторую инициализацию.
Метод __next__() должен возвращать следующий элемент в последовательности. По достижении конца и при последующих вызовах он должен вызывать StopIteration.
Пример, который возвращает следующее число из ряда Фибоначчи на каждой итерации:

```
class Fibonacci:

    def __init__(self, max):
        self.max = max

    def __iter__(self):
        self.count = 0
        self.last_numbers = (0, 1)

        return self

    def __next__(self):
        last_number, current_number = self.last_numbers
        last_number, current_number = current_number, last_number + current_number

        self.last_numbers = last_number, current_number
        self.count += 1

        if self.count > self.max:
            raise StopIteration

        return last_number

fibo = Fibonacci(10)

for number in fibo:
    print(number)
```


CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com

Закрепление материала

- Что такое итерабельный объект?
- Что такое итератор?
- Как создать свой итератор?

Дополнительное задание

Задание
Напишите итератор, который возвращает элементы заданного списка в обратном порядке (аналог reversed).

Самостоятельная деятельность учащегося

Задание 1
Реализуйте цикл, который будет перебирать все значения итерабельного объекта iterable

Задание 2
Взяв за основу код примера example_5.py, расширьте функциональность класса MyList, добавив методы для очистки списка, добавления элемента в произвольное место списка, удаления элемента из конца и произвольного места списка.

Рекомендуемые ресурсы

Документация Python
https://docs.python.org/3/tutorial/classes.html#iterators

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Итератор

CyberBionic systematics

CyberBionic Systematics ®
19 Eugene Sverstyuk Str., 5 floor
Kyiv, Ukraine

Tel. 0 800 337 146
E-mail: edu@cbsystematics.com
Site: www.edu.cbsystematics.com | itvdn.com