# Итераторы и генераторы

| **№ урока:** | 4 | **Курс:** | Python Essential |
|---|---|---|---|

| **Средства обучения:** | Python 3; интегрированная среда разработки (PyCharm или Microsoft Visual Studio + Python Tools for Visual Studio) |
|---|---|

## Обзор, цель и назначение урока

После завершения урока обучающиеся будут иметь представление об механизмах итераторов и генераторов, научатся создавать собственные итераторы, генераторы и простейшие сопрограммы.

## Изучив материал данного занятия, учащийся сможет:

- Иметь представление о внутренних механизмах работы цикла for в Python
- Создавать и использовать итераторы
- Создавать и использовать генераторы
- Использовать генераторы-выражения
- Использовать подгенераторы
- Иметь представление о yield-выражениях и сопрограммах и быть готовым к изучению асинхронного программирования в курсе Python Advanced

## Содержание урока

1. Итераторы
2. Генераторы
3. Генераторы-выражения
4. Подгенераторы
5. Yield-выражения
6. Сопрограммы

## 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



CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

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.

- *Функция-генератор (generator function)* – это функция, которая возвращает специальный *итератор генератора (generator iterator)* (также *объект-генератор – generator object)*. Она характеризуется наличием ключевого слова yield внутри функции.
- Термин *генератор (generator)*, в зависимости от контекста, может означать либо функцию-генератор, либо итератор генератора (чаще всего, последнее).
- Методы __iter__ и __next__ у генераторов создаются автоматически.
- yield замораживает состояние функции-генератора и возвращает текущее значение. После следующего вызова __next__() функция-генератор продолжает своё выполнение с того места, где она была приостановлена.
- Когда выполнение функции-генераторы завершается (при помощи ключевого слова return или достижения конца функции), возникает исключение StopIteration.
- Некоторые простые генераторы могут быть записаны в виде выражения. Они выглядят как выражение, содержащее некоторые переменные, после которого одно или несколько ключевых слов for, задающих, какие значения должны принимать данные переменные (синтаксис соответствует заголовку цикла for), и ноль или несколько условий, фильтрующих генерируемые значения (синтаксис соответствует заголовку оператора if). Такие выражения называются *выражениями-генераторами (generator expressions).*
- В Python 3 существуют так называемые *подгенераторы (subgenerators).* Если в функции-генераторе встречается пара ключевых слов yield from, после которых следует объект-генератор, то данный генератор делегирует доступ к подгенератору, пока он не завершится (не закончатся его значения), после чего продолжает своё исполнение.
- На самом деле yield является выражением. Оно может принимать значения, которые отправляются в генератор. Если в генератор не отправляются значения, результат данного выражения равен None.
- yield from также является выражением. Его результатом является то значение, которое подгенератор возвращает в исключении StopIteration (для этого значение возвращается при помощи ключевого слова return).
- Методы генераторов:
    - __next__() – начинает или продолжает исполнение функции-генератора. Результат текущего yield-выражения будет равен None. Выполнение затем продолжается до следующего yield-выражения, которое передаёт значение туда, где был вызван __next__. Если генератор завершается без возврата значения при помощи yield, возникает исключение StopIteration. Метод обычно вызывается неявно, то есть циклом for или встроенной функцией next().
    - send(value) – продолжает выполнение и отправляет значение в функцию-генератор. Аргумент value становится значением текущего yield-выражения. Метод send() возвращает следующее значение, возвращённое генератором, или выбрасывает исключение StopIteration, если генератор завершается без возврата значения. Если send() используется для запуска генератора, то единственным допустимым значением является None, так как ещё не было выполнено ни одно yield-выражение, которому можно присвоить это значение.
    - throw(type[, value[, traceback]]) – выбрасывает исключение типа type в месте, где был приостановлен генератор, и возвращает следующее значение генератора (или выбрасывает StopIteration). Если генератор не обрабатывает данное исключение (или выбрасывает другое исключение), то оно выбрасывается в месте вызова.
    - close() – выбрасывает исключение GeneratorExit в месте, где был приостановлен генератор. Если генератор выбрасывает StopIteration (путём нормального завершения или по причине того, что он уже закрыт) или GeneratorExit (путём отсутствия обработки данного исключения), close просто возвращается к месту вызова. Если же генератор возвращает очередное значение, выбрасывается исключение RuntimeError. Метод close() ничего не делает, если генератор уже завершён.



CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

- *Сопрограмма* (англ. *coroutine*) — компонент программы, обобщающий понятие подпрограммы, который дополнительно поддерживает множество входных точек (а не одну, как подпрограмма) и остановку и продолжение выполнения с сохранением определённого положения.
- Расширенные возможности генераторов в Python (выражения yield и yield from, отправка значений в генераторы) используются для реализации сопрограмм.
- Сопрограммы полезны для реализации асинхронных неблокирующих операций и кооперативной многозадачности в одном потоке без использования функций обратного вызова (callback-функций) и написания асинхронного кода в синхронном стиле.
- Python 3.5 включает в себе поддержку сопрограмм на уровне языка. Для этого используются ключевые слова async и await.

## Закрепление материала

- Что такое итерабельный объект?
- Что такое итератор?
- Что такое генератор?
- Какое исключение выбрасывает итератор, когда все элементы исчерпаны?
- Что такое подгенератор?
- Что такое выражение-генератор?
- Что является результатом yield-выражения?
- Какие методы есть у генераторов?
- Что такое сопрограмма?

## Дополнительное задание

Задание
Напишите функцию-генератор для получения n первых простых чисел.

## Самостоятельная деятельность учащегося

Задание 1
Напишите итератор, который возвращает элементы заданного списка в обратном порядке (аналог reversed).

Задание 2
Перепишите решение первого задания с помощью генератора.

Задание 3
Взяв за основу код примера 06-iterable_with_an_iterator.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
https://docs.python.org/3/tutorial/classes.html#generators
https://docs.python.org/3/tutorial/classes.html#generator-expressions
https://docs.python.org/3/reference/expressions.html#generator-expressions
https://docs.python.org/3/reference/expressions.html#yield-expressions
https://docs.python.org/3/reference/expressions.html#generator-iterator-methods
https://docs.python.org/3/reference/expressions.html#examples
https://www.python.org/dev/peps/pep-0255/
https://www.python.org/dev/peps/pep-0342/
https://www.python.org/dev/peps/pep-0380/



CyberBionic systematics
CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine
t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com

Статьи в Википедии о ключевых понятиях, рассмотренных на этом уроке
https://ru.wikipedia.org/wiki/Итератор
https://en.wikipedia.org/wiki/Generator_(computer_programming)
https://ru.wikipedia.org/wiki/Сопрограмма



CyberBionic Systematics ® 2015
19 Mariny Raskovoy Str., 5 floor
Kyiv, Ukraine

t. +380 (44) 361-8473
E-mail: edu@cbsystematics.com
Site:www.edu.cbsystematics.com