

# Python Essential

Итераторы и генераторы

# Python Essential

После урока обязательно



Повторите этот урок в видео формате на [ITVDN.com](http://itvdn.com)

Доступ можно получить через руководство вашего учебного центра



Проверьте как Вы усвоили данный материал на [TestProvider.com](http://testprovider.com)

# Итераторы и генераторы

# Итераторы и генераторы

## Итерабельные объекты



- **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__()`.
- 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.

# Итераторы и генераторы

## Итераторы



- **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.

# Итераторы и генераторы

## Генераторы



- **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-генератора завершается (при помощи ключевого слова `return` или достижения конца функции), возникает исключение **StopIteration**.

# Итераторы и генераторы

## Выражения-генераторы



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)**.

```
function(x, y) for x in range(10) for y in range(5) if x != y
```

# Итераторы и генераторы

## Подгенераторы



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.

```
def generator():  
    ...  
    yield from subgenerator()  
    ...
```



# Итераторы и генераторы

## Yield-выражения



- 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`).

```
def generator():  
    ...  
    data = yield  
    ...
```

# Итераторы и генераторы

## Методы генераторов

Метод	Описание
<code>__next__()</code>	Начинает или продолжает исполнение функции-генератора. Результат текущего <code>yield</code> -выражения будет равен <code>None</code> . Выполнение затем продолжается до следующего <code>yield</code> -выражения, которое выдаёт значение туда, где был вызван <code>__next__</code> . Если генератор завершается без выдачи значения при помощи <code>yield</code> , возникает исключение <code>StopIteration</code> . Метод обычно вызывается неявно, то есть циклом <code>for</code> или встроенной функцией <code>next()</code> .
<code>send(value)</code>	Продолжает выполнение и отправляет значение в функцию-генератор. Аргумент <code>value</code> становится значением текущего <code>yield</code> -выражения. Возвращает выданное значение. Если <code>send()</code> используется для запуска генератора, то единственным допустимым значением является <code>None</code> , так как ещё не было выполнено ни одно <code>yield</code> -выражение, которому можно присвоить это значение.
<code>throw(type[, value[, traceback]])</code>	Выбрасывает исключение типа <code>type</code> в месте, где был приостановлен генератор, и возвращает следующее значение генератора (или выбрасывает <code>StopIteration</code> ).
<code>close()</code>	Выбрасывает исключение <code>GeneratorExit</code> в месте, где был приостановлен генератор. Если генератор возвращает очередное значение, выбрасывается исключение <code>RuntimeError</code> .

# Итераторы и генераторы

## Сопрограммы



- **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`.

# Смотрите наши уроки в видео формате

ITVDN.com



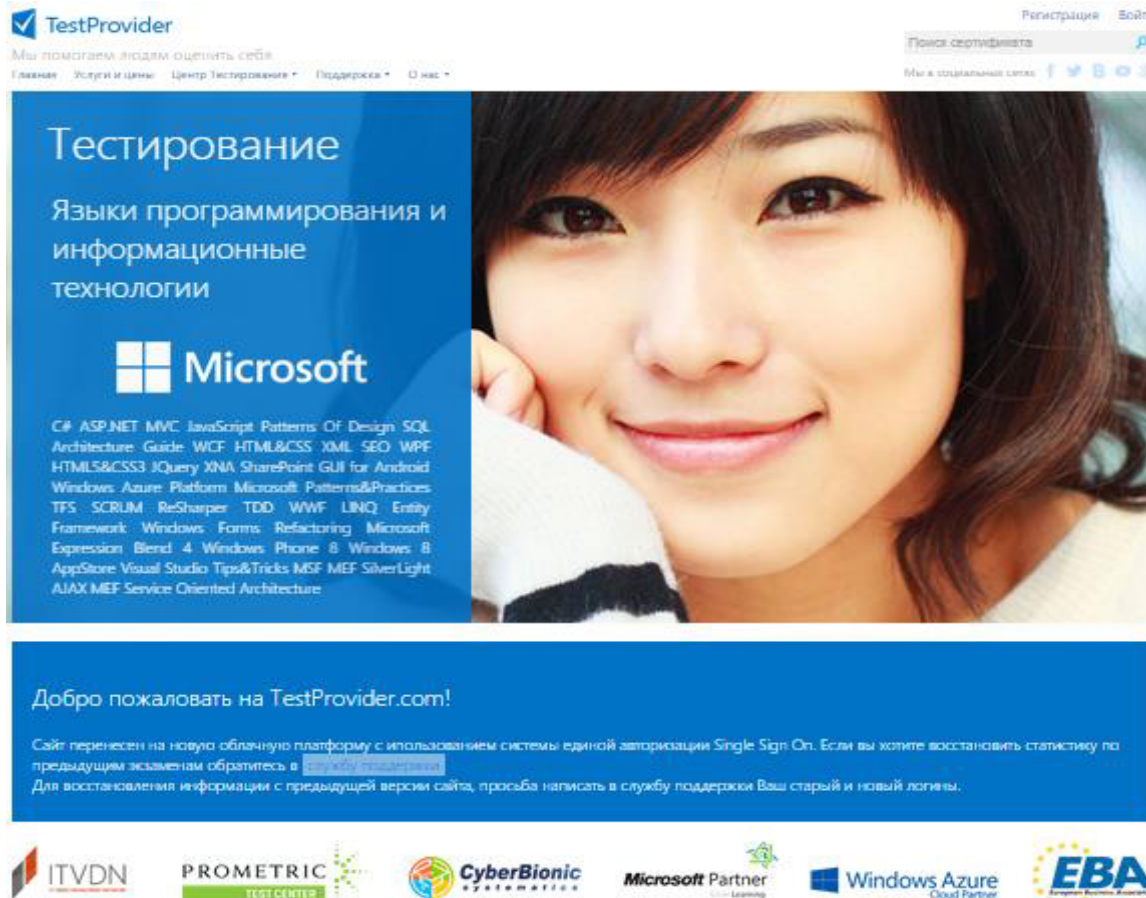
Посмотрите этот урок в видео формате на образовательном портале [ITVDN.com](http://itvdn.com) для закрепления пройденного материала.

Все курсы записаны сертифицированными тренерами, которые работают в учебном центре CyberBionic Systematics



# Проверка знаний

TestProvider.com



TestProvider

Мы помогаем людям оценить себя

Главная Услуги и цены Центр Тестирования Поддержка О нас

Регистрация Войти

Поиск сертификата

Мы в социальных сетях

## Тестирование

Языки программирования и информационные технологии

**Microsoft**

C# ASP.NET MVC JavaScript Patterns OF Design SQL Architecture Guide WCF HTML&CSS XML SEO WPF HTML5&CSS3 JQuery XNA SharePoint GUI for Android Windows Azure Platform Microsoft Patterns&Practices TFS SCRUM ReSharper TDD WWF LINQ Entity Framework Windows Forms Refactoring Microsoft Expression Blend 4 Windows Phone 8 Windows 8 AppStore Visual Studio Tips&Tricks MSF MEF SilverLight AJAX MEF Service Oriented Architecture

Добро пожаловать на TestProvider.com!

Сайт перенесен на новую облачную платформу с использованием системы единой авторизации Single Sign On. Если вы хотите восстановить статистику по предыдущим экзаменам обратитесь в [службу поддержки](#). Для восстановления информации с предыдущей версии сайта, просба написать в службу поддержки Ваш старый и новый логины.

ITVDN PROMETRIC TEST CENTER CyberBionic Microsoft Partner Windows Azure Cloud Partner EBA

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и общей оценки знаний IT специалиста.

После каждого урока проходите тестирование для проверки знаний на [TestProvider.com](http://TestProvider.com)

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.



# Python Essential

Q&A

# Информационный видеосервис для разработчиков программного обеспечения

