Урок 1

Знакомство с фреймворком

Зачем нужен Django. Сильные стороны фреймворка. Установка и настройка. Общие сведения о структуре проекта. Знакомство с urlpatterns. Первая страница и ее отображение.

Важно: требования

Так как наша компания следит за качеством преподаваемого материала, этот курс разработан с учетом последних версий используемых технологий:

- Python 3.6+;
- Django 2.1+.

Рекомендуемый уровень знаний

Чтобы успешно освоить данный курс, у вас должны быть основные знания о языке программирования Python по основным темам:

- переменные и типы данных;
- логические выражения и операторы ветвления (**if else**);
- циклы;
- итераторы (строки, списки, кортежи, словари);
- функции;
- основы ООП (объектно-ориентированное программирование);

Также вы должны знать:

- основы HTML;
- основы CSS;
- основы JavaScript и jQuery.

Зачем нужен Django

Планирование и реализация веб-сайтов требуют больших усилий. Django — один из лучших на сегодняшний день фреймворков, который позволяет быстро разрабатывать высокопроизводительные и полнофункциональные сайты. С помощью Django легко выстраиваются масштабируемые и легко расширяемые приложения для web с дизайном любой сложности.

Абстрагируясь от низкоуровневого процесса веб-строительства, Django предоставляет множество инструментов, реализующих стандартные шаблоны программирования (регистрация, авторизация, пагинация или разбиение на страницы и прочие).

Для работы с базами данных (БД) Django использует ORM (Object-Relational Mapping, объектно-реляционное отображение), что позволяет абстрагироваться от конкретной БД и работать с базой на объектном уровне. При необходимости довольно просто сменить одну БД на другую.

На Django написано много высоконагруженных веб-сайтов, например:

- Instagram;

- Pinterest;
- The Washington Post;
- NASA;
- Новостной портал Полит.Ру;
- Сайт радиостанции «Говорит Москва»;
- Сайт международного брокера недвижимости Tranio.

Сильные стороны фреймворка

Прежде чем приступить к рассмотрению Django, назовем его основные преимущества:

1. **ORM**. Django предоставляет простой механизм работы с базой без изучения синтаксиса SQL-запросов, а также возможность абстрагироваться от конкретной БД.
2. **Диспетчер URL на основе шаблонов и регулярных выражений.** Диспетчер URL — входная точка для любых запросов, связывающая адрес запроса с его обработчиком.
3. **Шаблонизатор**. Расширяемая система шаблонов с тегами и наследование позволяют быстро и удобно создавать динамические страницы любой сложности.
4. **Интернационализация**. Встроенная система интернационализации помогает переводить сайты на различные языки.
5. **Паттерн проектирования MVC (Model Template View)**. Django поощряет свободное связывание и строгое разделение частей приложения. Если следовать этой философии, легко вносить изменения в одну конкретную часть приложения без ущерба для остальных.
6. **Готовые модули — готовые решения**. У множества встроенных и подключаемых модулей есть готовые шаблоны для решения рутинных задач: регистрации, авторизации, интеграции с социальными сетями.
7. **Промежуточные слои (Middleware)**. Готовые обработчики запросов и ответов позволяют упростить рутинные задачи построения правильных http-ответов и разборки http-запросов, что особенно полезно для начинающих программистов.
8. **Система кеширования**. Кеширование страниц, которые часто используются, позволяет существенно снизить нагрузку на сервер сайта.

И это далеко не полный список всех возможностей фреймворка Django.

По сути, Django — это реализация паттерна **MVC** (**Model, View, Controller**), в котором проект делится на три слоя (уровня). Один из них — **Model** (модели) — отвечает за хранение и получение данных по запросу. Слой **View** (представления) отвечает за то, как данные видит пользователь. В Django эту роль играют шаблоны (**Templates**). Слой **Controller** (контроллеры) взаимодействует с пользователем, делает запросы к модели и передает их результат в представление. В Django роль контроллеров, как это ни странно, выполняют файлы с именем **views.py**, поэтому реализация **MVC** в Django называется **MVT** (**Model, View, Template**).

Если вы поймете философию и научитесь правильно пользоваться фреймворком, то писать сайты станет гораздо приятнее, а эффективность возрастет.

Установка Django

Надеемся, что Python у вас уже установлен. Если нет, скачайте с python.org последнюю версию. Для работы хватит Python 3.6 или последующих версий, но лучше устанавливать самую свежую. Для Django будет достаточно версии 2.0 (желательно 2.1.3).

Установка под Ubuntu

1. **sudo apt-get install python3-pip —** устанавливаем **pip**.
2. **sudo pip3 install django —** устанавливаем **django**, используя **pip**.
3. **python3 -m django --version —** проверяем установку. Если увидели версию, то установка прошла успешно.

Замечание: при установке можно настроить виртуальное окружение для Django при помощи инструмента virtualenv (для нашего курса этого делать не нужно).

Установка под Windows

1. Запускаем командную оболочку с правами администратора (**WIN + R → cmd**).
2. Выполняем команду: **pip install django**.
3. **python -m django --version —** проверяем установку. Если увидели версию, то установка прошла успешно.

Django устанавливается в системную директорию **site-packages** в папке, где установлен Python.

Замечание: если возникли проблемы, необходимо проверить, прописан ли у вас путь к интерпретатору в переменной среды **PATH**. Для этого в командной строке выполняем: **python —version**. Если увидели номер версии Python — все хорошо, если нет — редактируем переменную **PATH**. Можно просто удалить Python и поставить заново, убедившись, что в окне настроек установки стоит галочка рядом с пунктом **add to PATH**. Потом обязательно перезагрузить компьютер.

Несколько слов про IDE

IDE — интегрированная среда разработки.

Можете использовать любую привычную вам IDE. Мы будем работать в PyCharm (для курса достаточно бесплатной версии **Community**).

В принципе, можно работать в любом текстовом редакторе с подсветкой синтаксиса (**Notepad++**, **Sublime**, **Visual Studio Code**) и запускать скрипты из командной строки. Это полезно для развития навыков, но требует больше времени.

Работать вообще без IDE можно, но крайне не рекомендуется.

Перед созданием нового проекта

Мы будем развивать проект интернет-магазина, созданный на курсе по HTML. Необходимо подготовить верстку страниц (например: **index.html**, **products.html**, **contacts.html**) и файл со стилями (**style.css**). Также должна быть папка с изображениями **img** и папка со шрифтами **fonts**. Все документы размещаем в отдельной папке (например, **C:\PyProjects\lesson_1\step_1**).

Создание нового проекта

Django установлен, файлы верстки подготовлены — пришло время создать проект **geekshop**. Для этого создаем папку на диске (например, в Windows: **C:\PyProjects\lesson_1\step_2**). Далее в папке Windows нажимаем **Shift+ПКМ** (правая клавиша мыши) и в меню выбираем пункт «Открыть окно команд». В Ubuntu просто нажимаем в папке ПКМ и выбираем пункт «Открыть в терминале».

Оказываемся в командной строке (терминале) в нужной папке. Выполняем команду:

```
python3 -m django startproject geekshop
```

В результате автоматически создается папка с именем проекта (**geekshop**). Переходим в эту папку прямо в командной строке (терминале):

```
cd geekshop
```

Внутри этой папки — еще одна с таким же именем (в ней настройки проекта), файлы **manage.py** и **db.sqlite3**. Мы можем это все увидеть, выполнив команду:

```
dir
```

Главное на данном этапе — не запутаться в структуре. Будем в дальнейшем считать корнем проекта папку, где находится файл **manage.py**.

Создание нового приложения

Проект принято разбивать на логические части — приложения. Разница между проектом и приложением в том, что первое является конфигурацией, а второе — кодом.

Проект — это экземпляр определенного набора кода django-приложений и конфигурация для них.

С технической точки зрения существует одно требование к проекту — наличие файла конфигурации, который определяет способ соединения с базой данных, список установленных приложений, каталог с шаблонами и так далее.

Приложение — это переносимый набор функциональности. Обычно включает в себя модели и представления, которые хранятся вместе в едином пакете языка Python.

Например, Django поставляется с рядом приложений: системой комментирования и автоматическим интерфейсом администратора. Их важной особенностью является то, что они переносимы и их можно использовать во множестве проектов.

Существует очень мало жестких правил для соответствия вашего кода этой схеме. Если создаете простой сайт, можете использовать единственное приложение. Если сложный с несколькими независимыми частями — например, интернет-магазин и форум — можете разнести их в разные приложения, что позволит использовать их отдельно в других проектах.

Главный критерий при разделении проекта на приложения — это их автономность, возможность в перспективе применять их в других проектах независимо друг от друга. Например, в любом проекте должна быть админка (**adminapp**) и система авторизации пользователей (**authapp**). В проектах интернет-магазинов должна быть корзина (**basketapp**). Создадим пока одно «главное» приложение **mainapp**:

```
python3 -m django startapp mainapp
```

Рекомендуется название приложения составлять из букв в нижнем регистре без подчеркиваний и дефисов. В конце всегда будем добавлять сочетание **app**, чтобы не путать папки с приложениями и другие. Когда создано приложение, автоматически появляется папка с его именем. Ее содержимое рассмотрим позже.

Запуск сервера разработки

Чтобы проверить, правильно ли установлен Django и создан наш проект, запустим сервер разработки.

Сервер разработки Django (его еще называют **runserver** — по имени команды, которая его запускает) — это встроенный легкий веб-сервер, который можно использовать при разработке сайта. Он включен в Django, чтобы быстро приступать к разработке сайта, не тратя время на конфигурирование боевого веб-сервера (например, Apache). Этот сервер разработки отслеживает изменения в коде и автоматически перезагружает его, помогая видеть вносимые изменения без перезагрузки веб-сервера.

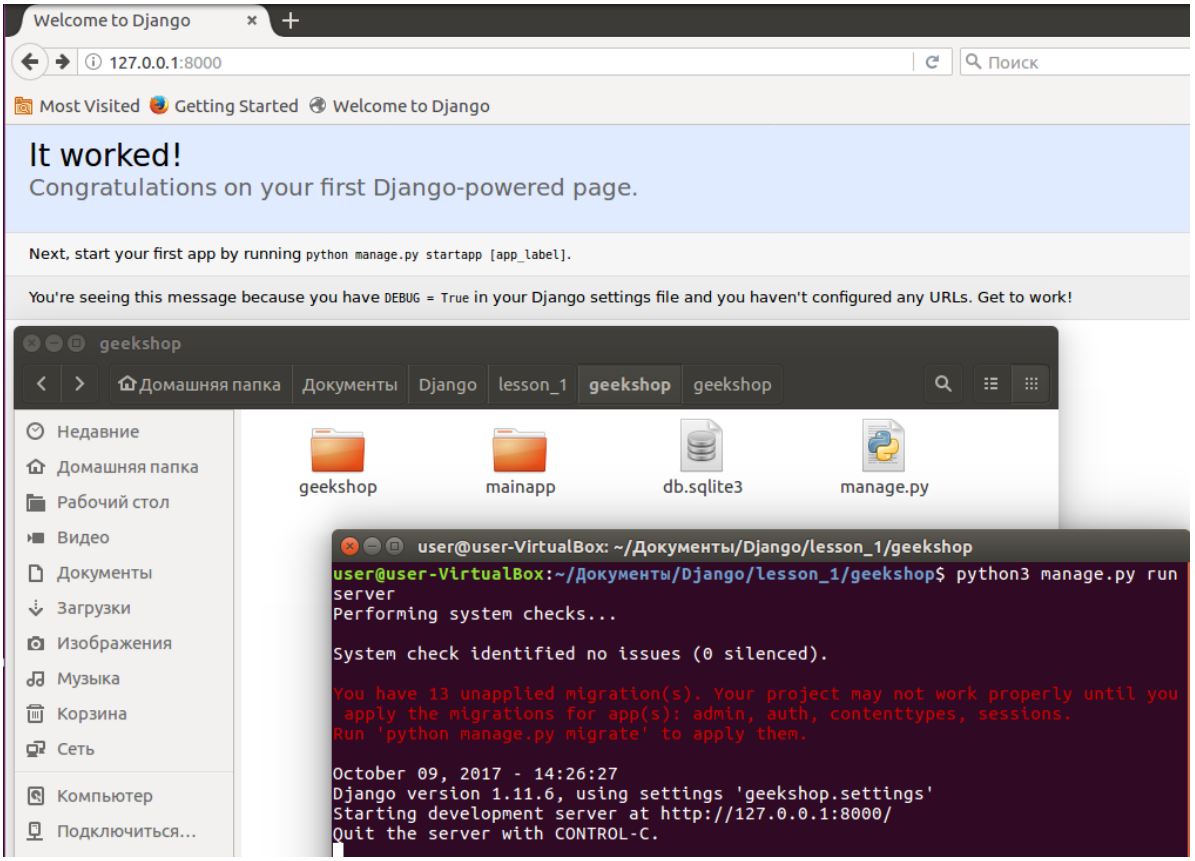
Чтобы запустить сервер, необходимо запустить командную строку в **корне проекта** (вы сейчас должны там находиться) и выполнить команду:

```
python3  manage.py runserver
```

После этого открыть любой браузер и написать адрес:

```
127.0.0.1:8000 или localhost:8000
```

Должна появиться страница **It worked**:



Команда запускает сервер локально на порту 8000. Сервер принимает только локальные соединения с вашего компьютера.

Этот сервер очень удобен во время разработки, но сопротивляйтесь искушению использовать его в боевом режиме. Он может обрабатывать только один запрос в единицу времени и не проходил никакого аудита безопасности. Вопросы развертывания сайта на боевом сервере рассмотрим в следующем курсе.

Сообщение о 13 непримененных миграциях — это пока нормально. Когда начнем работать с БД, они исчезнут.

Пользователям Windows рекомендуется в **корне проекта** создать текстовый файл с именем **run.bat** и в нем прописать команду запуска сервера:

```
python3 manage.py runserver
pause
```

Если вы сделали все правильно, у файла будет значок с шестеренкой. С его помощью можно быстро запускать сервер Django двойным кликом мыши.

Общие сведения о структуре проекта

Кратко рассмотрим назначение файлов и папок проекта на данный момент:

```
geekshop/
    geekshop/
        __init__.py
        settings.py
        urls.py
        wsgi.py
    manage.py
    db.sqlite3
    mainapp/
        migration/
            __init__.py
        admin.py
        apps.py
        models.py
        tests.py
        views.py
```

geekshop/ — папка с важными для проекта файлами.

> **__init__.py** — файл необходим для того, чтобы Python рассматривал данный каталог как пакет, то есть как группу модулей. Это пустой файл, и обычно не требуется добавлять что-либо в него.
>
> **settings.py** — настройки проекта Django.
>
> **url.py** — диспетчер url-адресов проекта.
>
> **wsgi.py** — файл, который потребуется для развертывания на боевом сервере.

manage.py — утилита командной строки, которая позволяет взаимодействовать с проектом различными методами. Наберите **python manage.py help** (в Ubuntu вместо **python** всегда необходимо писать **python3**), чтобы получить информацию о возможностях утилиты. Не надо изменять содержимое данного файла, он создан в каталоге для удобства.

db.sqlite3 — файл с БД проекта, который Django создает по умолчанию (можно использовать **mysql** или **postgresql**, но мы оставим **sqlite3**).

mainapp/ — папка с приложением **mainapp**.

> **migration/** — папка, где будут создаваться миграции при работе с БД (будем разбирать на следующих уроках).
>
> **admin.py** — файл для работы встроенной в Django админки.
>
> **apps.py** — вспомогательный файл Django, никогда не будем менять его содержимое.
>
> **models.py** — файл, где описываются **модели** django-приложения (будем создавать на следующих уроках).

tests.py — файл с тестами приложения (тесты будут рассматриваться на следующем курсе).

views.py — файл с контроллерами приложения, выполняющими его основную логику (сегодня создадим три контроллера для основных страниц **— index**, **products** и **contacts**).

В перспективе появятся еще папки — их назначение будет объясняться в дальнейшем. Сразу вручную создадим в **mainapp** папку с шаблонами (**templates**) со следующей структурой:

```
templates/
    mainapp/
        index.html
        products.html
        contacts.html
```

Это может показаться сложным сначала, но потом вы привыкнете к такой структуре — мы размещаем шаблоны внутри **templates** в папке с именем приложения, которое, по сути, становится для них пространством имен (**namespace**). Благодаря такому подходу при сборке проекта не возникнет конфликтов, даже если в разных приложениях будут шаблоны с одинаковыми именами. Файлы шаблонов — это страницы, созданные на курсе по HTML. Чуть позже мы скорректируем их содержимое.

В будущем при создании нового приложения всегда будем создавать в нем папку **templates** с такой структурой. Имя папки — **templates**, потому что Django автоматически просматривает папки с таким именем в поисках шаблонов. Можно поместить все шаблоны в одну папку в корне проекта и прописать ее в файле настроек **settings.py** (константа **TEMPLATES**, список **‘DIRS’: []**), но при наращивании функциональности это приведет к путанице в именах шаблонов. Поэтому сразу привыкнем делать правильно.

Посвятите достаточно времени изучению структуры проекта — в дальнейшем она будет усложняться. Попробуйте удалить и создать проект заново несколько раз.

Замечание: имя внешней папки, в которой расположен проект, можно менять как угодно, имена папок в корне проекта — нельзя.

Файл настроек settings.py и раздача статики

Настроим проект. Откроем файл **settings.py** в любом текстовом редакторе. Обойдемся пока минимальными правками.

Всегда будем дописывать имя нового приложения в список **INSTALLED_APPS**. Если этого не сделать, будут проблемы с миграциями и шаблонами.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'mainapp',
]
```

Также добавим три строки кода в конец файла — это позволит организовать раздачу статических файлов (папки **CSS**, **JS**, **IMG**, **FONTS**) силами сервера Django. Можно настроить для этого отдельный сервер (**NGNIX**). Главное — результат: файлы должны быть доступны по url-адресу **127.0.0.1:8000/static/**.

```
STATIC_URL = '/static/'

STATICFILES_DIRS = (
    os.path.join(BASE_DIR, "static"),
)
```

Сами статические файлы разместим в папке **static** в корне проекта, которая будет иметь следующую структуру:

```
static/
    css/
    fonts/
    img/
    js/
```

Копируем файлы из верстки магазина в соответствующие папки.

Замечание: обратите внимание на значение константы **DEBUG = True** по умолчанию — это режим разработчика, при котором в браузер выводится максимум отладочной информации. Вторая важная константа, **BASE_DIR** — это путь к **корню проекта**.

Несколько слов об URL

Прежде чем приступить к созданию первой страницы, подробнее познакомимся с url-обработчиком и общим подходом Django к работе с красивыми URL.

Элегантная схема URL — важная составляющая высококачественного веб-приложения. Django поощряет создание красивых схем URL и не захламляет их мусором, подобным .**php** или **.asp**.

Проектируя модель URL для своего приложения, вы создаете модуль Python, который называют менеджером URL-ов. Для обработки адресов используются шаблоны путей, которые прописаны в файле **urls.py** (он в папке **geekshop** в корне проекта). Добавим в него четыре строки:

```
from django.urls import path
from django.contrib import admin
import mainapp.views as mainapp

urlpatterns = [
    path('', mainapp.main),
    path('products/', mainapp.products),
    path('contact/', mainapp.contact),
    path('admin/', admin.site.urls),
]
```

Первая строка — это импорт модуля **views.py** из приложения **mainapp**. Остальные строки — это записи соответствия шаблонов url-адресов и функций-обработчиков из файла **mainapp.views** при помощи функции **path()**.

За счет такого подхода получаем логику типа: «пользователь запросил такой url-адрес — обработаем его соответствующей функцией из **mainapp.views.py**».

Замечание: обратите внимание, что функция **path()** первым аргументом принимает часть пути, вторым — ссылку на функцию-обработчик. Не забывайте импортировать модули с функциями-обработчиками!

На следующих уроках изучим возможности перехвата значений из url-адреса и передачи в функцию-обработчик (например, номер категории товара или **PrimaryKey** продукта).

На самом деле при работе с запросами в функцию-обработчик всегда по умолчанию передается еще дополнительная информация — объект **request**. Он играет большую роль в Django и является в некотором смысле буфером обмена между контроллерами (**views.py**) и шаблонами (**templates**).

Замечание: объект **request** — это по сути контекстный процессор, который прописан в файле **settings.py** в константе **TEMPLATES[“OPTIONS”][“context_processors”]** — список, один из элементов которого **django.template.context_processors.request**.

Необходимо учитывать, что Django сработает на **первое совпадение** запрашиваемого url-адреса с путем из списка. Поэтому если какой-то из адресов обрабатывается не той функцией — просто поменяйте местами элементы в **urlpatterns**.

Если адрес не подходит ни под одно из выражений — Django вызовет исключение 404.

Замечание: будьте внимательны к символам «/» в конце адреса — в Django это имеет значение.

Первая страница и остальные

Мы уже сделали много подготовительных операций: создали структуру проекта, приложение, папку с тремя шаблонами. Добавили папку со статическими файлами. Прописали настройки в конфигурационном файле и файле диспетчера url-адресов.

Осталось написать контроллеры в файле **views.py** (папка приложения **mainapp**):

```
from django.shortcuts import render


def main(request):
    return render(request, 'mainapp/index.html')


def products(request):
    return render(request, 'mainapp/products.html')


def contact(request):
    return render(request, 'mainapp/contact.html')
```

В контроллер всегда передается как минимум один аргумент — объект запроса **request**. В нашем коде контроллеры пока не выполняют никакой логики, а просто возвращают результат рендеринга шаблонов. Обратите внимание на путь к шаблонам — он задается относительно папки **templates**.

Мы пользуемся функцией **render()** из модуля **django.shortcuts**, которая должна получить как минимум два аргумента: объект **request** (по сути, он через нее пробрасывается в шаблон) и путь к шаблону. Внимание: **контроллер всегда** должен **возвращать объект ответа** — должен быть **return.**

Можно пробовать! Запускаем сервер Django и снова переходим по адресу: **127.0.0.1:8000** — мы должны увидеть главную страницу магазина. Пока она без стилей и шрифтов.

Пробуем прописать адрес **127.0.0.1:8000/products/** — должны увидеть страницу с каталогом продуктов. И по адресу **127.0.0.1:8000/contact/** — страницу с контактными данными магазина.

Если одна из страниц не отображается — смотрим код ошибки в браузере и действуем.

Возможные причины:

- ошибка в имени папки с шаблонами;
- ошибки в именах шаблонов;
- ошибки в url-путях;
- ошибки в именах функций-обработчиков;
- ошибки в **import**.

Финальный штрих

Скорректируем файлы, чтобы страницы отображались корректно.

В путях к статическим файлам дописываем **/static/**:

- стили и шрифты:

```
<link rel="stylesheet" type="text/css" href="/static/css/style.css">
<link rel="stylesheet" href="/static/fonts/font-awesome/css/font-awesome.css">
```

- изображения (в том числе и **в файле стилей css!**):

```
<img src="/static/img/arrow.png" alt="arrow">
```

Ссылки в меню корректируем с учетом указанных путей:

```
<ul class="menu">
    <li><a href="/" class="active">домой</a></li>
    <li><a href="/products/">продукты</a></li>
    <li><a href="/contact/">контакты</a></li>
</ul>
```

После правок все должно корректно отображаться. Не забывайте нажимать **CTRL+F5** для очистки кеша в браузере (Chrome).

Практическое задание

1. Подготовить исходники для проекта — три страницы из верстки магазина. Разместить их в одной папке. Мы рекомендуем сразу использовать свою вёрстку, а не магазин мебели из курса HTML. Так ваш проект не будет 1001 клоном и ваше портфолио будет отличаться от сотни других.

2. Установить Django и PyCharm. Создать проект и в нем — приложение **mainapp**. Проверить, что все работает.

3. Разместить шаблоны и статические файлы в соответствующих папках. Настроить проект — файл **settings.py**. Отредактировать файл диспетчера url-адресов **urls.py**.

4. Написать функции-обработчики для всех страниц — файл **views.py** в приложении **mainapp**. Проверить работу всех страниц проекта в черновом режиме (без стилей и изображений).

5. Откорректировать пути к статическим файлам и адреса гиперссылок в меню. Проверить, что все работает верно: стили и изображения грузятся, гиперссылки работают.

6. Сделать файл **run.bat** для быстрого запуска django-сервера.

Все проблемы и пути их решения подробно обсудим на следующем занятии.

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Django 2.1.
2. WiKi — URN.