Урок 1

Знакомство с фрэймворком

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

Рекомендуемый уровень знаний

Для успешного освоения данного курса, у вас уже должны быть основные знания о языке программирования Python.

Вот основные темы, которые вы должны знать перед началом данного курса:

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

Также вы должны знать:

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

Зачем нужен Django

Планирование и реализация веб-сайтов всегда сопровождаются большими затратами усилий. Django представляет собой 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 Веб с дизайном любой степени 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, рус. объектно-реляционное отображение), что позволяет абстрагироваться от конкретной БД и работать с базой на объектном уровне. Также, при необходимости, довольно просто сменить одну БД на другую.

На Django написано много больших и высоконагруженных веб-сайтов, вот лишь несколько примеров:

- Новостной портал – http://polit.ru/
- Сайт радиостанции Говорит Москва – http://govoritmoskva.ru/
- Сайт по продаже недвижимости – https://tranio.ru/

Сильные стороны фрэймворка

Прежде чем приступить к рассмотрению самого Django, кратко рассмотрим основные его преимущества:

- ORM. Django предоставляет простой механизм работы с базой без изучения синтаксиса SQL-запросов, а также возможность абстрагироваться от конкретной БД.

- Диспетчер URL на основе регулярных выражений. Диспетчер URL – входная точка для любых запросов, связывающая адрес запроса с его обработчиком.
- Шаблонизатор. Расширяемая система шаблонов с тегами и наследование позволяет быстро и удобно создавать динамические страницы любой сложности.
- Интернационализация. Встроенная система интернационализации помогает переводить сайты на различные языки.
- Паттерн проектирования MVC (Model View Template). Django поощряет свободное связывание и строгое разделение частей приложения. Если следовать этой философии, то легко вносить изменения в одну конкретную часть приложения без ущерба для остальных частей.
- Готовые модули – готовые решения. Большое количество встроенных и подключаемых модулей имеют уже готовые шаблоны для решения многих рутинных задач, таких как: регистрация, авторизация, интеграция с социальными сетями и прочие.
- Промежуточные слои (Middleware). Готовые обработчики запросов и ответов позволяют упростить рутинные задачи построение правильных http-ответов и разборки http-запросов, что особенно полезно для начинающих программистов.
- Система кэширования. Кэширование часто используемых страниц позволяет существенно снизить нагрузку на сервер вашего сайта.

И это далеко не полный список всех возможностей фрэймворка Django.

По сути, Django – это реализация паттерна «MVC» (Model, View, Controller), в котором проект делится на 3 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 данным фрэймворком, то написание ваших сайтов станет гораздо эффективнее и приятнее.

Установка Django

Надеемся, что, Python у вас уже установлен, если нет: https://www.python.org/downloads/.

Для работы нам будет достаточно любой версии Питона 3.6 или новее (лучше устанавливать самую свежую).

Django будет достаточно версии 1.10 (желательно 1.11).

Установка под Ubuntu

1. `sudo apt-get install python3-pip` – устанавливаем pip.
2. `sudo pip3 install django` – устанавливаем django, используя pip.
3. `django-admin --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. `django-admin --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 Питона – все хорошо, иначе – редактируем переменную `PATH`. Можно просто удалить Питон 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.

Вы можете использовать любую, привычную вам IDE, мы будем работать в PyCharm (для нашего 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 некоторых 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 этой папке («Windows») нажимаем Shift+ПКМ (правая клавиша мыши) – в меню выбираем пункт «Открыть окно команд». В «Ubuntu» просто нажимаем в папке ПКМ и выбираем пункт «Открыть в терминале».

Итак, мы оказались в командной строке (терминале) в нужной папке. Выполняем команду:

```
django-admin 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 папка с таким же именем (в ней настройки проекта), файл `manage.py` и файл `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.

Но в чём же разница между проектом и приложением? Разница в том, что первое является конфигурацией, а второе – кодом:

Проект – это экземпляр определённого набора кода django-приложений и конфигурация для этих приложений.

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 некой функциональности, обычно включает в себя модели и представления, которые хранятся вместе в едином пакете языка Python.

Например, Django поставляется с рядом приложений, таких как система комментирования и автоматический интерфейс администратора. Важной особенностью этих приложений является то, что они переносимы и их можно использовать во множестве проектов.

Существует очень мало жёстких правил для соответствия вашего кода этой схеме. Если вы создаёте простой сайт, вы можете использовать единственное приложение. Если вы создаёте сложный сайт с несколькими независимыми частями, такими как интернет-магазин и форум, возможно, вы пожелаете разнести их в отдельные приложения, что позволит использовать их раздельно 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»:

```
django-admin 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 другие папки. После создания приложения автоматически появляется папка с его именем. Ее содержимое рассмотрим позже.

Запуск сервера разработки

Для проверки правильности установки Django и созданного нами проекта давайте запустим сервер разработки.

Сервер разработки Django (также называемый «runserver» по имени команды, которая его запускает) – это встроенный лёгкий веб-сервер, который вы можете использовать в процессе разработки вашего сайта. Он включен в Django для того, чтобы вы могли быстро приступить к разработке вашего сайта без траты времени на конфигурирование вашего боевого веб-сервера (например, Apache) раньше времени. Этот сервер разработки отслеживает изменения в вашем коде и автоматически перезагружает его, помогая видеть вносимые вами изменения без перезагрузки веб-сервера.

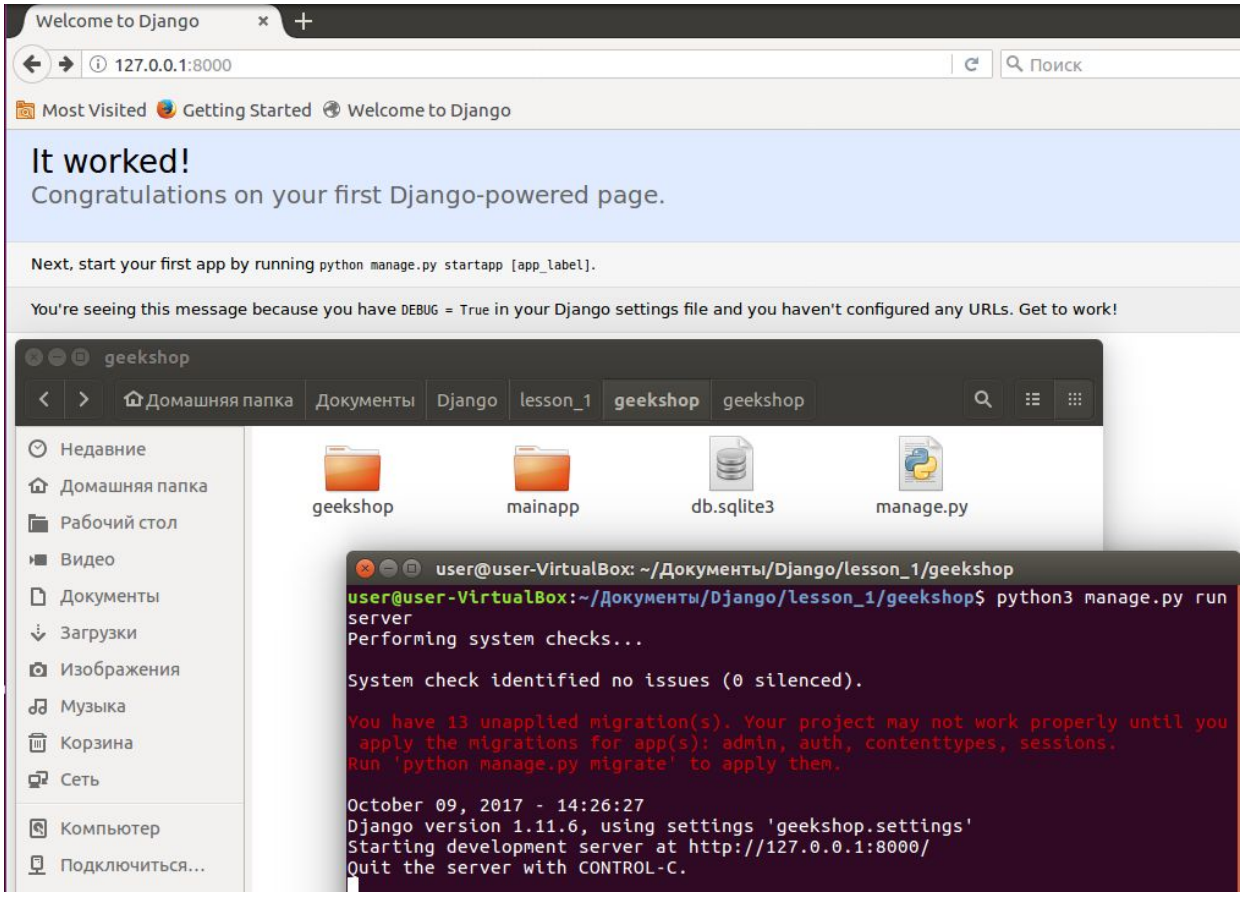
Для запуска сервера необходимо запустить командную строку в **корне проекта** (вы сейчас должны там находиться) и выполнить команду:

```
python 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.

Несмотря на то, что этот сервер очень удобен во время разработки, сопротивляйтесь искушению использовать его в боевом режиме. Этот сервер может обрабатывать только один запрос в единицу времени, и он не проходил никакого аудита на предмет безопасности. Вопросы развертывания сайта на боевом сервере будут рассмотрены 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:

```
python manage.py runserver
pause
```

Если вы сделали все правильно – файл будет иметь значок с шестеренкой. С его помощью можно быстро запускать сервер Django двойным кликом мыши.

Общие сведения о структуре проекта

Давайте кратко рассмотрим назначение файлов и папок проекта на данный момент:

```
…/step_2/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, т. е., как группу модулей. Это пустой файл, и обычно вам не требуется добавлять что-либо в него.
>
> **settings.py**: настройки проекта Django.
>
> **url.py**: диспетчер url-адресов проекта.
>
> **wsgi.py**: необходимый в будущем для развертывания на боевом сервере файл.

manage.py: утилита командной строки, которая позволяет вам взаимодействовать с проектом различными методами. Наберите `python manage.py help` (в Ubuntu вместо `python` всегда необходимо писать – `python3`) для получения информации о возможностях утилиты. Вы не должны изменять содержимое данного файла, он создан в данном каталоге в целях 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, необходимый 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 3 контроллера для основных страниц – index, products и contacts).

В перспективе появятся еще папки – их назначение будет поясняться в дальнейшем. Давайте 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. Может возникнуть вопрос: «Почему имя папки должно быть templates?» Ответ: «Django автоматически просматривает папки с таким именем в поисках шаблонов». В принципе 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 функционала это приведет к путанице в именах шаблонов. Поэтому сразу привыкнем делать правильно.

Посвятите **достаточное количество 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

Пришла пора настроить наш проект. Откройте 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. В принципе, можно настроить для этого отдельный сервер (`NGNIX`). Главное – результат: файлы должны быть доступны по url-адресу `127.0.0.1:8000/static/`.

```
# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/1.10/howto/static-files/

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

Потерпите, еще совсем немного, и мы приступим к созданию нашей первой страницы. Подробнее познакомимся с url-обработчиком и общим подходу Django к работе с красивыми URL.

Красивая, элегантная схема URL является важной составляющей высококачественного веб-приложения. Django поощряет создание красивых схем URL и не захламляет их мусором, подобным `.php` или же `.asp`.

Проектируя модель URL для своего приложения, вы создаёте модуль Python, называемый менеджером URL-ов. Для обработки адресов используются регулярные выражения (вспоминаем курс

«Python-1»), которые прописаны в файле `urls.py` (он в папке `geekshop` в корне проекта). Добавим в него четыре строки:

```
16  from django.conf.urls import url
17  from django.contrib import admin
18  import mainapp.views as mainapp
19
20  urlpatterns = [
21      url(r'^$', mainapp.main),
22      url(r'^products/', mainapp.products),
23      url(r'^contact/', mainapp.contact),
24      url(r'^admin/', admin.site.urls),
25  ]
```

Первая строка (18 на рисунке) – это импорт модуля `views.py` из приложения «mainapp». Остальные строки (с 21 по 23) – это записи соответствия регулярных выражений-ловушек url-адресов и функций-обработчиков из файла `mainapp.views` при помощи функции **`url()`**.

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 `url()` – первым аргументом принимает шаблон в виде регулярного выражения, вторым – ссылку на функцию-обработчик. Не забывайте импортировать модули с функциями-обработчиками!

На следующих уроках мы научимся использовать группировку в регулярных выражениях (скобки) для получения значений из url-адреса и передачи в функцию-обработчик (например, номер категории товара или **P**rimary**K**ey продукта).

На самом деле при работе с запросами в функцию-обработчик всегда по умолчанию передается еще некоторая 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`, словарь `OPTIONS`, ключ `context_processors` – список, один из элементов которого `django.template.context_processors.request`.

Необходимо учитывать, что Django сработает на **первое совпадение** запрашиваемого url-адреса с регулярным выражением 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. Также следует вспомнить роль «^» – признак начала строки и «$» – признак конца строки (важно, если адрес будет наращиваться в дальнейшем – например, `products/edit/15`). Буква **«r»** в начале выражения – означает: воспринимать их как сырые («**r**aw»), без учета экранирующих символов.

Первая страница и остальные

Мы уже сделали много подготовительных операций: создали структуру проекта, создали приложение, создали папку для шаблонов и разместили в ней три шаблона. Создали папку для статических

файлов и разместили их в ней. 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')
```

Как и было сказано ранее, 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 будет некрасивая – без стилей и шрифтов.

Далее, 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.

Причины:

- ошибка в имени папки с шаблонами;
- ошибки в именах шаблонов;
- ошибки в регулярных выражениях;
- ошибки в именах функций-обработчиков;
- ошибки в import.

Подробнее рассмотрим обработку URL с помощью регулярного выражения

На данном этапе важно вспомнить и понять принцип обработки строки регулярным выражением. Часть, связанная с адресом самого сервера, всегда отбрасывается (в нашем случае это **`127.0.0.1:8000/`**). С регулярным выражением Django будет сравнивать часть адреса, расположенную правее, ее называют URN.

Например, если мы используем URL: `http://127.0.0.1:8000/`, то его URN будет пустая строка, именно она подходит под r'^$'.

Финальный штрих

Пришло время скорректировать некоторые 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 регулярных выражений:

```
<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, у вас все должно корректно отображаться. Не забывайте нажимать **CTRL+F5** для очистки кеша в браузере (Chrome).

Домашнее задание

1. Подготовить исходники для проекта – 3 страницы из верстки магазина, и разместить их в одной папке.
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 как положено (стили и изображения грузятся, гиперссылки работают).
6. Сделать файл `run.bat` для быстрого запуска django-сервера.

Внимание!!! Решенное задание поместить ВНУТРЬ папки с именем вида «dz1_Ivanov_Ivan», затем ЭТУ папку сжать ЛЮБЫМ архиватором (например, zip или 7zip) и уже ПОТОМ прикреплять на сайт.

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

Дополнительные материалы

Все то, о чём сказано в методичке, но более подробно:

1. Установка Django
2. Создание нового проекта
3. Книга Джанго

Используемая литература

Для подготовки данного методического пособия были использованы следующие ресурсы:

1. Django 1.11
2. Регулярные выражения в Питоне (на английском)
3. WiKi – URN