

Лекция 3. Шаблоны

Часть 2



Статичные файлы

Для оформления веб-страниц можно использовать статичные файлы.

Это файлы изображений, скриптов, каскадных таблиц стилей.

Статичные файлы не изменяются динамически и их содержание не зависит от контекста запроса.

Каскадные таблицы стилей (Cascading Style Sheets, CSS) —

это стандарт, определяющий представление данных в браузере.

Если HTML предоставляет информацию о структуре документа, то **таблицы стилей** сообщают, как он должен выглядеть.



Объявление CSS-стиля состоит из двух частей: селектора и объявления.

Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) перечисляются формирующие команды — свойства и их значения.

В HTML имена элементов нечувствительны к регистру, поэтому «h1» работает так же, как и «H1».



По методам добавления стилей в документ различают три вида стилей CSS:

- Внутренние стили
- Глобальные стили
- Внешние (связанные) стили



Внутренние стили

Определяются атрибутом **style** конкретных тегов.

Внутренний стиль действует только на определенные такими тегами элементы.

Например:

```
<p style="color:blue">Абзац с текстом синего цвета</p>
```

```
<p style="color:red">Абзац с текстом красного цвета</p>
```



```
<html>
```

```
<head>
```

```
<style type="text/css">
```

```
  p {color:#808080;}
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<p>Серый цвет текста во всех абзацах Web-страницы</p>
```

```
<p>Серый цвет текста во всех абзацах Web-страницы</p>
```

```
</body>
```

```
</html>
```

Глобальные стили CSS

Располагаются в контейнере `<style>...</style>`, который расположен в свою очередь в контейнере `<head>...</head>`.

Gray / Серый / #808080 Шестнадцатеричный Код Цветов



Внешние (связанные) стили

Определяются в отдельном файле с расширением .css.

Внешние стили позволяют всем страницам сайта выглядеть единообразно.

Для связи с файлом, в котором описаны стили, используется тег **<link>**, расположенный в контейнере **<head>...</head>**.

Например:



Например: два атрибута: `rel="stylesheet"` и `href` определяют адрес файла стилей.

```
<html>
```

```
<head>
```

```
.....
```

```
<link rel="stylesheet" href="style.css">
```

```
.....
```

```
</head>
```

```
<body>
```

```
.....
```

```
</body>
```

```
</html>
```



Подключение глобальных и внешних стилей

Правило состоит из селектора и объявлений стиля.

Селектор, расположенный в левой части правила, определяет элемент (элементы), для которых установлено правило.

Далее, в фигурных скобках перечисляются объявления стиля, разделенные точкой с запятой.

Например:

```
p {  
  
    text-indent: 30px;  
  
    font-size: 14px;  
  
    color: #666;  
  
}
```

Объявление стиля – это пара: свойство CSS: значение CSS.

Например: color: red

color	свойство CSS, определяющее цвет текста;
red	значение CSS, определяющее красный цвет.



Внутреннее подключение стиля

При внутреннем подключении стиля правило CSS, которое является значением атрибута **style**, состоит из объявлений стиля, разделенных точкой с запятой. Например:

```
<p style="text-indent: 30px; font-size: 14px; color: #666;">...</p>
```

Подробнее:

[Каскадные таблицы стилей CSS \(htmlweb.ru\)](http://htmlweb.ru)

[Каскадные таблицы стилей](#)

[Основы CSS \(html5book.ru\)](http://html5book.ru)

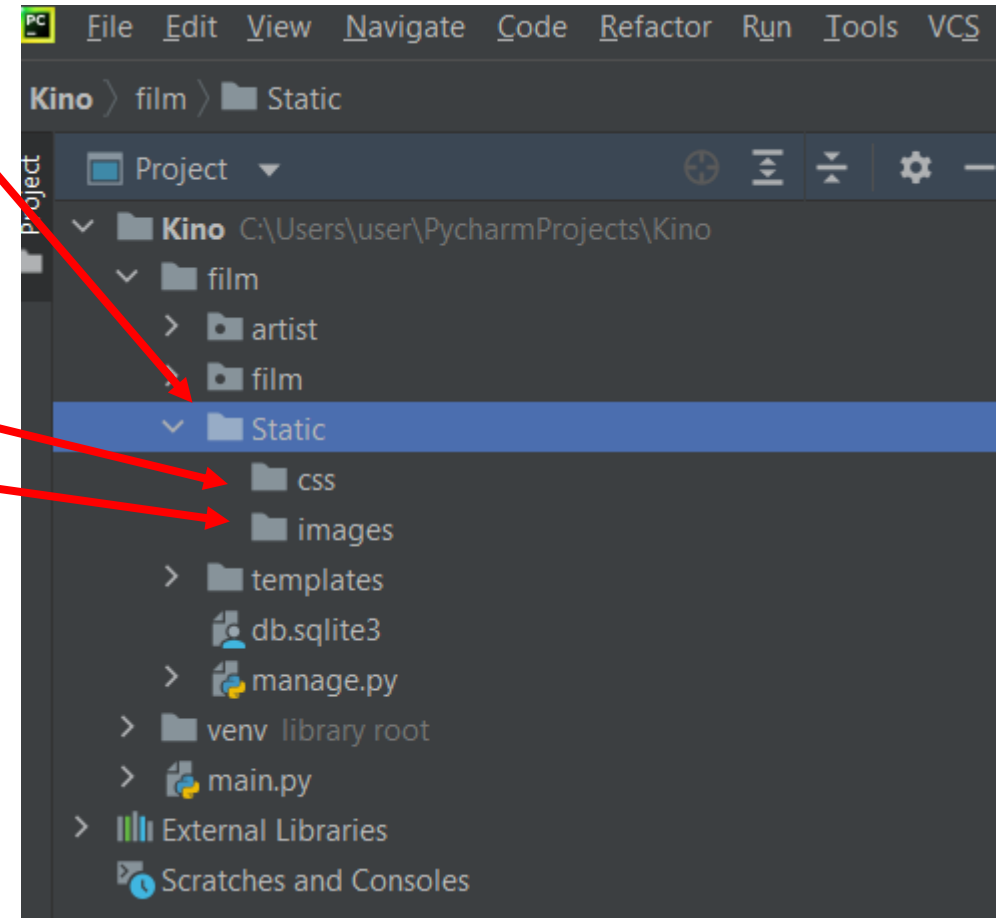


Использование статичных файлов в приложениях Django

В корневую папку проекта **film** добавим папку **static**.

В папке Static создадим 2 папки:

- папка **css** для таблиц стилей
- папка **images** для изображений



В папке **css** создадим файл **styles.css**

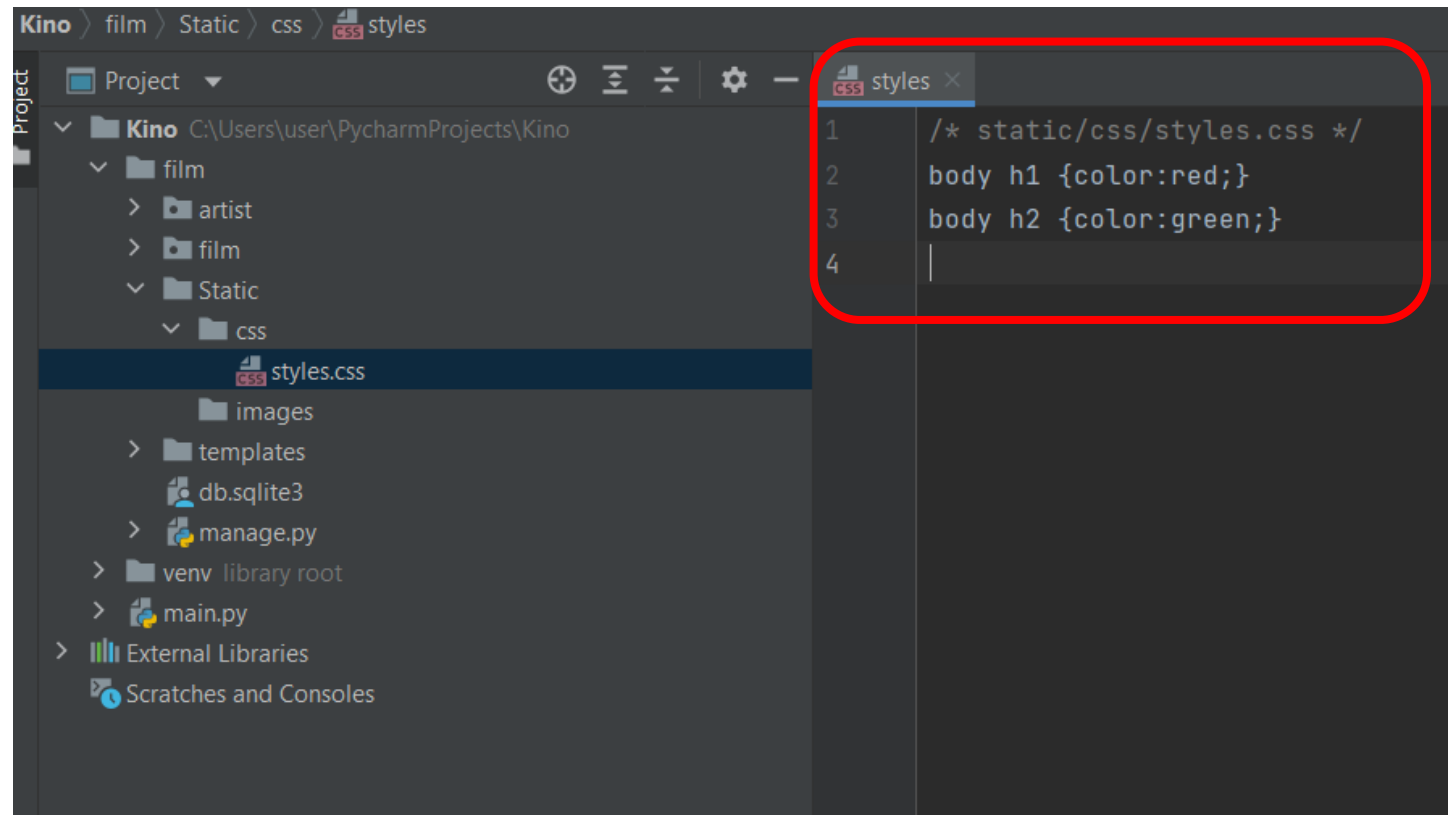
(по щелчку правой кнопки мыши на папке **css** надо выбрать **New/File** и ввести имя **styles.css**).

В файле **styles.css** записать код:

```
/* static/css/styles.css */
```

```
body h1 {color:red;}
```

```
body h2 {color:green;}
```



Включим в файл шаблона инструкцию:

```
{% load static %}
```

Для указания пути к статическим файлам используется выражение:

```
{% static 'Путь к файлу внутри папки static' %}
```



Изменим шаблон **home.html** (папка templates/artist/):

```
home x
1  <!DOCTYPE html>
2  {% load static %}
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <title>Привет!!! </title>
7      <link href="{% static 'css/styles.css' %}" rel="stylesheet">
8  </head>
9  <body>
10     <h1>Домашняя страница Django</h1>
11     <h2>templates/artist/home.html</h2>
12 </body>
13 </html>
14
```



Путь к папке static пропишем в файле settings.py:

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

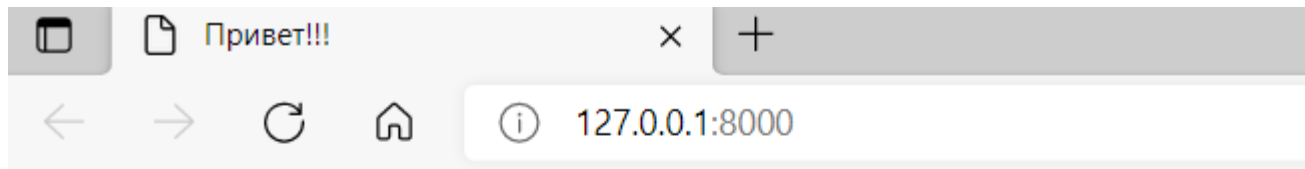
```
STATIC_URL = '/static/'  
STATICFILES_DIRS = [  
    os.path.join(BASE_DIR, "static"),  
]
```



В файле **views.py** изменим функцию **index**:

```
def index(request):  
    return render(request, "artist/home.html")
```

В результате этих изменений страница `home.html` откроется с выделением тега `h1` красным цветом, а тега `h2` - зеленым цветом:



Домашняя страница Django

`templates/artist/home.html`



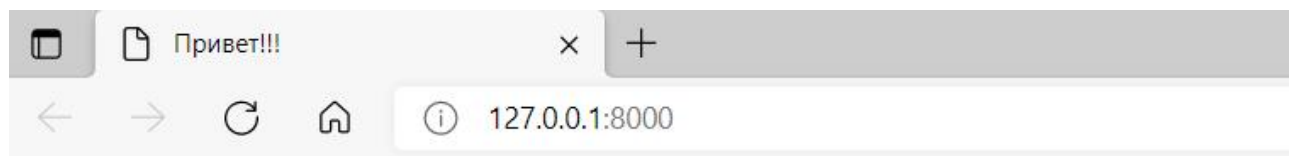
Вывод на странице `home.html` изображения

Поместим в папку `images` любое изображение, например, с именем **`ship.jpg`**.

Для вывода этого изображения на странице **`home.html`** отредактируем шаблон:

```
home x
1  <!DOCTYPE html>
2  {% load static %}
3  <html lang="en">
4  <head>
5      <meta charset="UTF-8">
6      <title>Привет!!! </title>
7      <link href="{% static 'css/styles.css' %}" rel="stylesheet">
8  </head>
9  <body>
10     <h1>Домашняя страница Django</h1>
11     <h2>templates/artist/home.html</h2>
12     
13 </body>
14 </html>
15
```





Домашняя страница Django

[templates/artist/home.html](#)



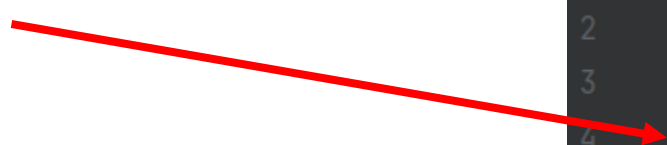
Для того, чтобы задать стиль отображения для рисунка, например, указать ширину 270 пикселей, надо в файл **styles.css** внести:

```
/* static/css/styles.css */
```

```
body h1 {color:red;}
```

```
body h2 {color:green;}
```

```
img{width:270px;}
```



```
css styles x
1  /* static/css/styles.css */
2  body h1 {color:red;}
3  body h2 {color:green;}
4  img{width:270px;}
5  |
```



Использование встроенного класса **TemplateView**

Если в ответ на запрос надо просто вернуть пользователю содержимое шаблона, то для этого необязательно определять функцию-представление.

Можно воспользоваться встроенным классом **TemplateView**.



В папке **film\templates\artist** определим 2 файла-шаблона.

файл-шаблон **about.html**:

```
about x
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Привет!</title>
6  </head>
7  <body>
8      <h1>Сведения об IT-компании </h1>
9      <h2>Новые технологии </h2>
10     <h3>templates/artist/about.html</h3>
11
12 </body>
13 </html>
14 |
```

и файл-шаблон **contact.html**:

```
contact x
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <title>Привет!!!</title>
6  </head>
7  <body>
8      <h1>Контакты IT-компании </h1>
9      <h2>Новые технологии </h2>
10     <h3>Новосибирск, Технопарк, офис 115 </h3>
11     <h4>templates/artist/contact.html</h4>
12
13 </body>
14 </html>
15
```



Внесем также изменения в файл `urls.py`:

включим `from django.views.generic import TemplateView` и пропишем путь:

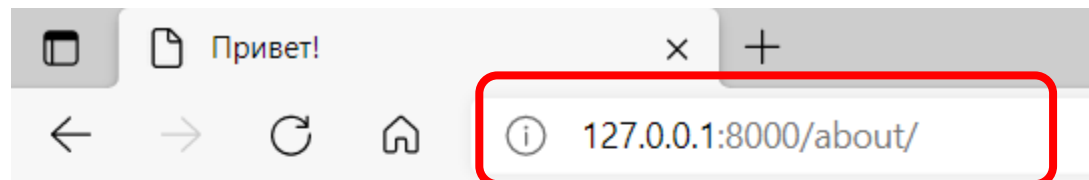
```
16 from django.contrib import admin
17 from django.urls import path, re_path
18 from artist import views
19 from django.views.generic import TemplateView
20
21 handler404 = views.pagenotfound
22
23 urlpatterns = [
24     path('', views.index, name='home'),
25     path('about/', TemplateView.as_view(template_name="artist/about.html")),
26     path('contact/', TemplateView.as_view(template_name="artist/contact.html")),
```

Через класс `TemplateView` вызываются страницы `artist/about.html` и `artist/contact.html`.



Переходим на сервер:

```
path('about/', TemplateView.as_view(template_name="artist/about.html")),  
path('contact/', TemplateView.as_view(template_name="artist/contact.html")),
```

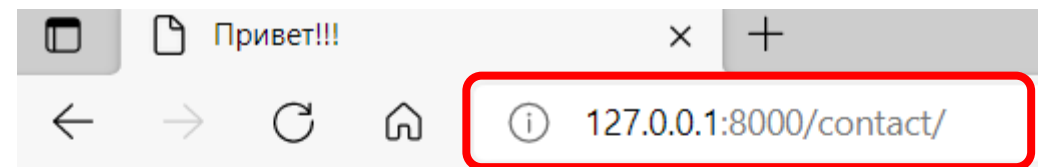


Сведения об IT-компании

Новые технологии

templates/artist/about.html

```
<body>  
  <h1>Сведения об IT-компании </h1>  
  <h2>Новые технологии </h2>  
  <h3>templates/artist/about.html</h3>  
</body>
```



Контакты IT-компании

Новые технологии

Новосибирск, Технопарк, офис 115

templates/artist/contact.html

```
<body>  
  <h1>Контакты IT-компании </h1>  
  <h2>Новые технологии </h2>  
  <h3>Новосибирск, Технопарк, офис 115 </h3>  
  <h4>templates/artist/contact.html</h4>  
</body>
```



Данные для отображения в шаблоне можно передать в метод `TemplateView.as_view` с помощью параметра `extra_context`. Данные при этом представляются в виде словаря.

Изменим файл **файл urls.py**:

```
urlpatterns = [  
  
    path("", views.index, name='home'),  
  
    path('about/', TemplateView.as_view(template_name="artist/about.html"),  
  
    path('contact/', TemplateView.as_view(template_name="artist/contact.html",  
        extra_context={"work": "Проектирование ИС"})),  
  
    ....  
  
]
```



Объект **work** прописываем в шаблоне contact.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Привет!!!</title>
</head>
<body>
  <h1>Контакты IT-компании </h1>
  <h2>Новые технологии </h2>
  <h3>{{work}}</h3>
  <h4>Новосибирск, Технопарк, офис 115 </h4>
  <h5>templates/firstapp/contact.html</h5>
</body>
```

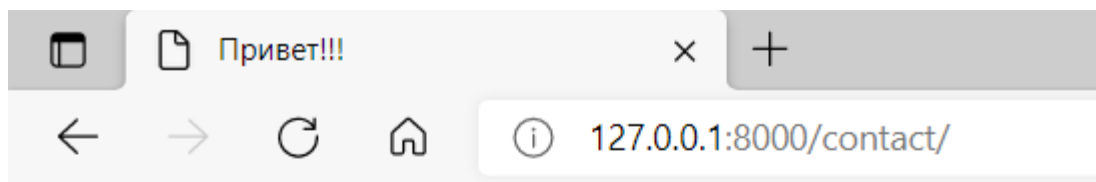


```
22
23 urlpatterns = [
24     path('', views.index, name='home'),
25     path('about/', TemplateView.as_view(template_name="artist/about.html"),
26     path('contact/', TemplateView.as_view(template_name="artist/contact.html",
27     |extra_context={"work": "Проектирование ИС"})),
```

{{work}}

```
contact x
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Привет!!!</title>
6 </head>
7 <body>
8     <h1>Контакты IT-компании </h1>
9     <h2>Новые технологии </h2>
10    <h3>{{work}}</h3>
11    <h4>Новосибирск, Технопарк, офис 115 </h4>
12    <h5>templates/artist/contact.html</h5>
13 </body>
14 |
```





Контакты IT-компаний

Новые технологии

Проектирование ИС

Новосибирск, Технопарк, офис 115

`templates/artist/contact.html`



Конфигурация шаблонов и пути к шаблонам в файле settings.py.

За конфигурацию шаблонов проекта отвечает переменная **TEMPLATES** в файле **settings.py**:

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [TEMPLATE_DIR],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]
```

- Переменная **BACKEND** указывает, что надо использовать шаблоны Django
- Переменная **DIRS** указывает на каталоги в проекте, которые будут содержать шаблоны
- Переменная **APP_DIRS** при значении **True** указывает, что поиск шаблонов будет производиться не только в самой папке, указанной в параметре **DIRS**, но и в ее подкаталогах. Если такое поведение недопустимо, то можно установить значение **False**.
- Переменная **OPTIONS** указывает, какие обработчики (процессоры) будут использоваться при обработке шаблонов.

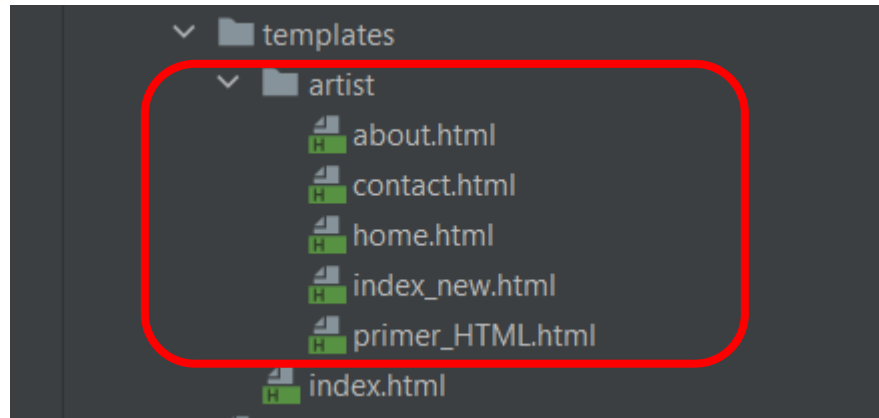


Пути к шаблонам

Как правило, шаблоны помещаются в общую папку в проекте, либо ее подкаталоги.

Например, можно определить папку **templates**. Если в проекте несколько приложений, которые должны использовать какие-то свои шаблоны, то, чтобы избежать проблем с именованием, можно создать для каждого приложения подкаталог, в который помещаются шаблоны для конкретного приложения.

Так, если в проекте **Film** создано приложение **artist**, то в папке **templates** создается папка **artist** и в ней хранятся все шаблоны этого приложения.



В файле **setting.py** путь к шаблонам определяется так:

Берется определенная в начале файла settings.py переменная **BASE_DIR**:
она представляет путь к проекту, и к этому пути добавляется папка **templates**:

```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")
```



Конец части 2. Продолжение следует..

