Серверная разработка ПО

Лаба 3

Шаблоны

Из этой темы мы узнаем:

□ Что такое шаблоны и для чего они нужны;

□ как использовать функцию render () и класс TemplateResponse для загрузки шаблонов;

□ как передать в шаблоны простые и сложные данные;

□ что такое статичные файлы, файлы CSS и как использовать статичные файлы в приложениях на Django;

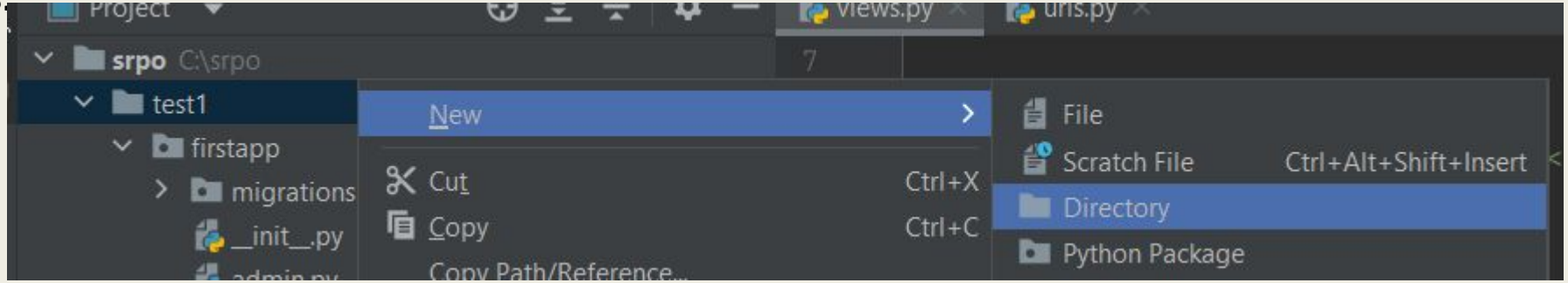
□ как можно изменить конфигурацию шаблонов HTML-страниц;

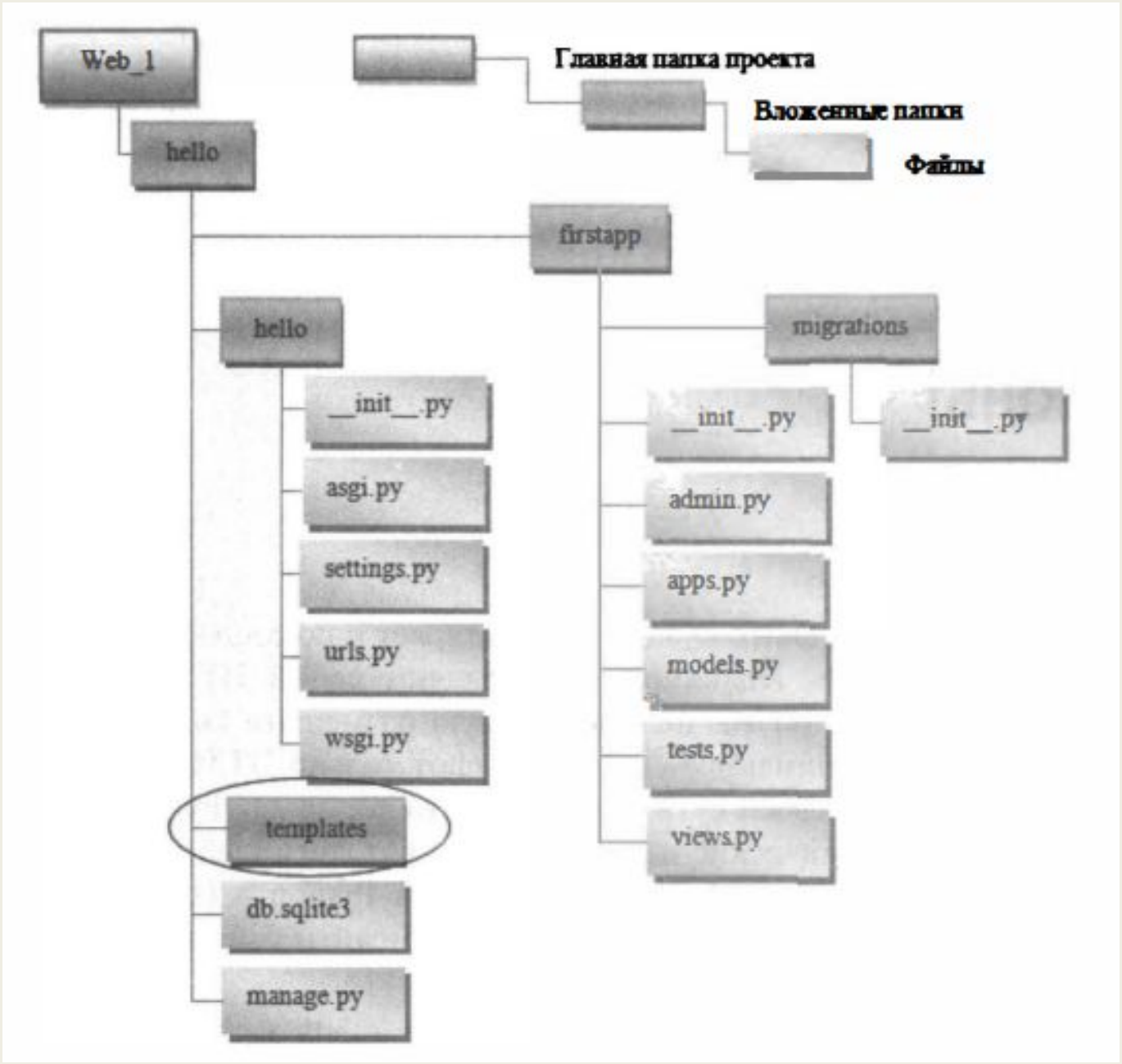
□ как можно расширить шаблоны HTML-страниц на основе базового шаблона;

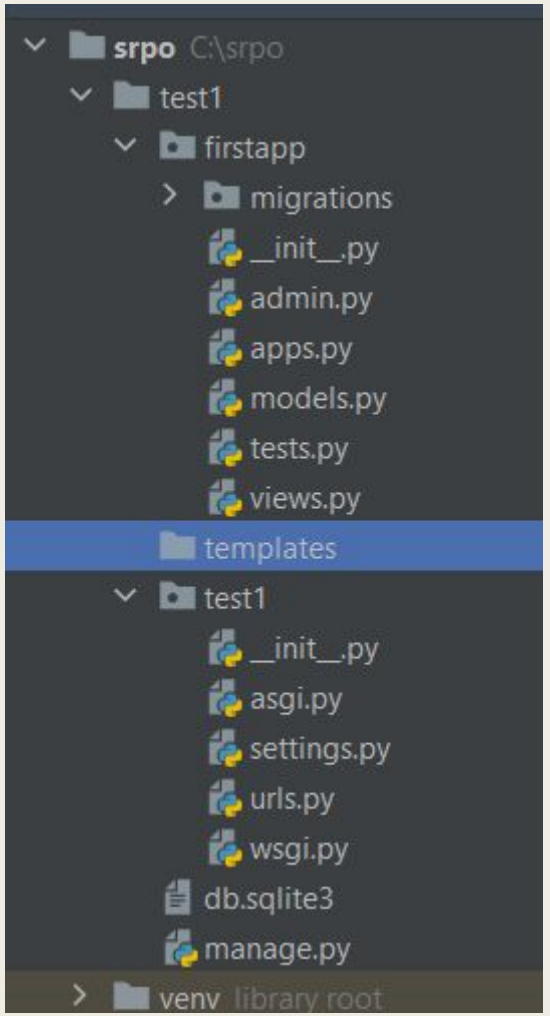
□ как можно использовать специальные теги в шаблонах HTML-страниц.

Шаблоны (templates) отвечают за формирование внешнего вида приложения. Они предоставляют специальный синтаксис, который позволяет внедрять данные в код HTML.

В предыдущих темах мы создали и доработали проект test1 с приложением firstapp. Теперь добавим в проект шаблоны. Для этого определим в корневой папке проекта новый каталог с именем templates. Вообще-то имя папки с шаблонами может быть любым, но, как правило, для лучшего восприятия структуры проекта этой папке все же лучше присвоить значащее имя templates.



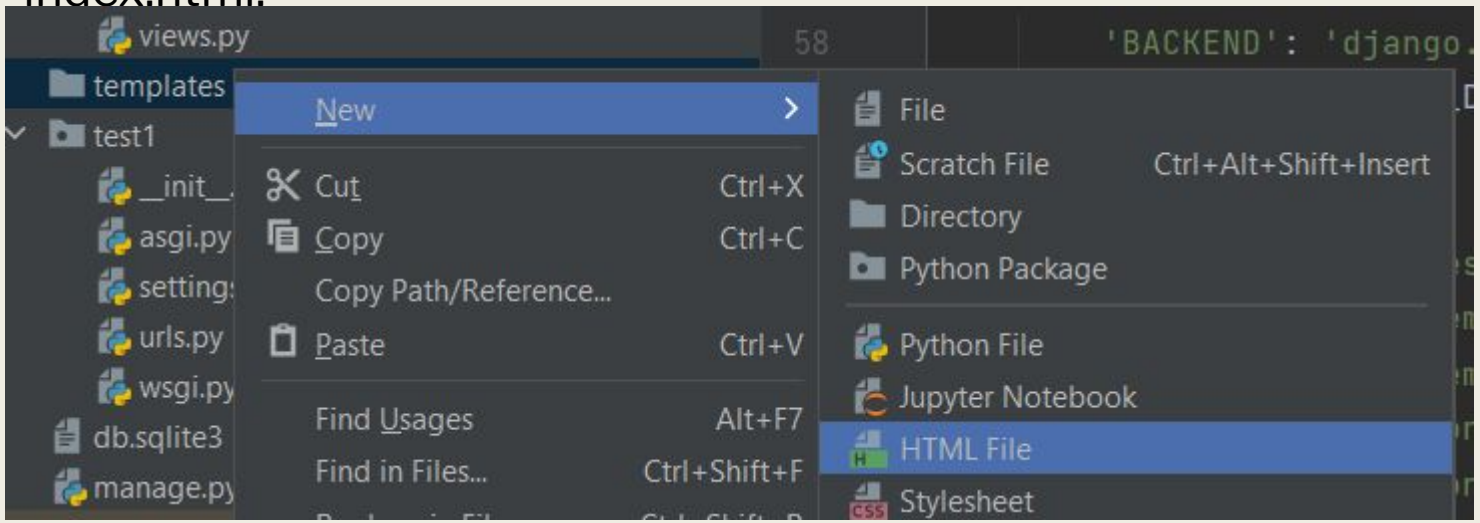

Web_1
Главная папка проекта
Вложенные папки
Файлы
hello
firstapp
hello
migrations
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
templates
db.sqlite3
manage.py
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
__init__.py
__init__.py

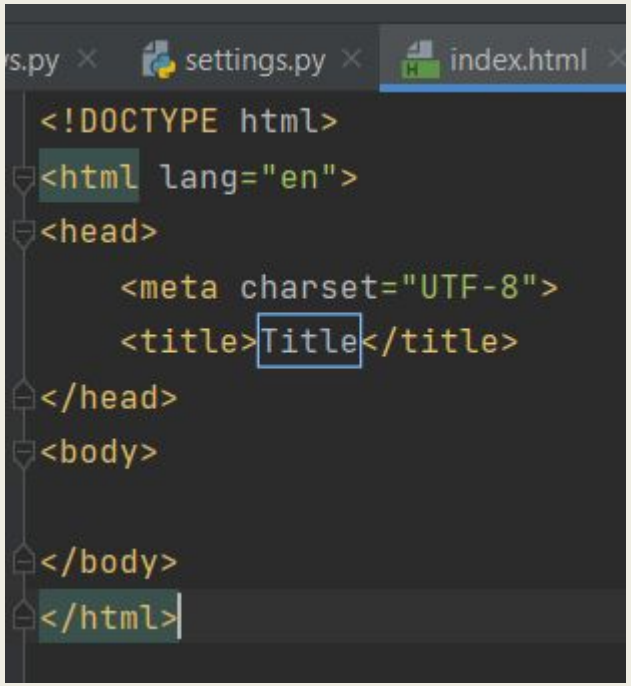

srpo C:\srpo
test1
firstapp
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
templates
test1
__init__.py
asgi.py
settings.py
urls.py
wsgi.py
db.sqlite3
manage.py
venv library root

Теперь нам надо указать, что этот каталог будет использоваться в качестве хранилища шаблонов. Для этого откроем файл settings.py. В этом файле настройка шаблонов производится с помощью переменной TEMPLATES. Здесь параметр DIRS задает набор каталогов, которые хранят шаблоны. Но по умолчанию он пуст. Изменим этот фрагмент кода следующим образом:

```
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',
            ],
        }
```

Теперь создадим в папке templates новый файл index.html.





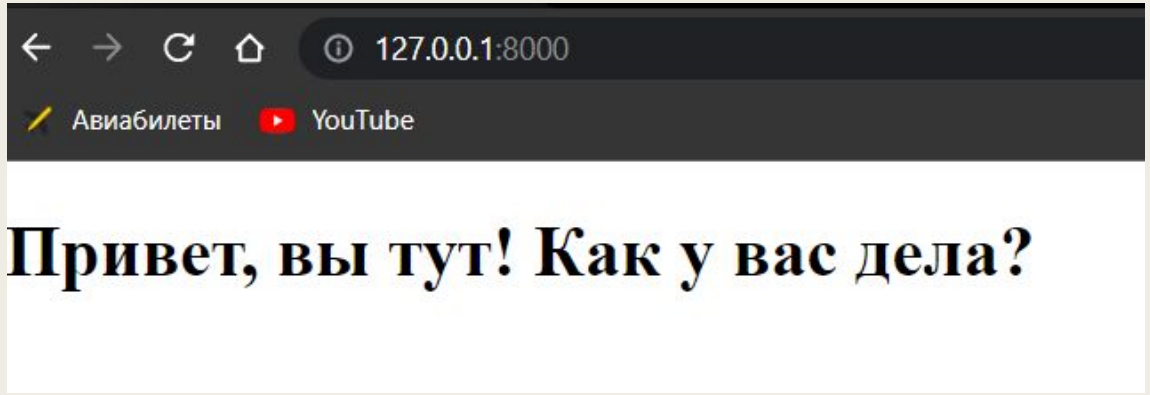
По сути, это обычная веб-страница, содержащая код HTML. Теперь используем эту страницу для отправки ответа пользователю.

Для этого перейдем в приложении firstapp к файлу views.py, который определяет функции для обработки запроса пользователя, и изменим этот файл следующим образом:

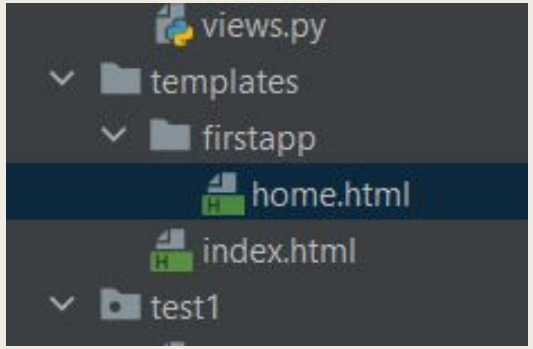
```python
from django.shortcuts import render
from django.http import HttpResponseNotFound
from django.http import HttpResponse, HttpRespo

def index(request):
    return render(request, 'index.html')
```

```python
urlpatterns = [
    path('', views.index),
```

Что мы здесь сделали? Первым шагом мы из модуля django.shortcuts импортировали функцию render (предоставлять). Вторым шагом изменили функцию def index (request). Теперь функция index (request) вызывает функцию render, которой передаются объект запроса пользователя request и файл шаблона index.html, который находится в папке templates.



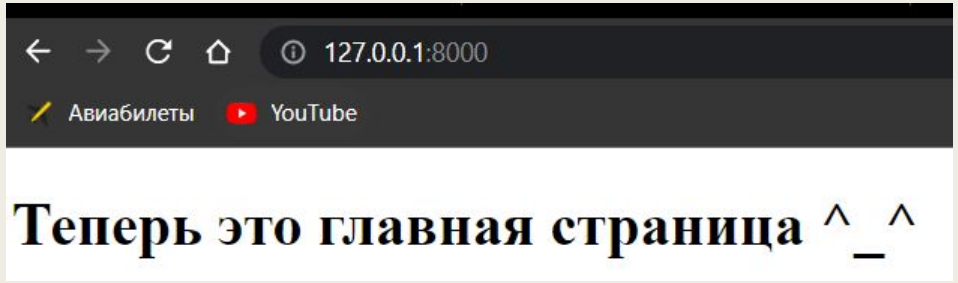
Однако в проекте Django нередко бывает несколько приложений. И каждое из этих приложений может иметь свой набор шаблонов. Чтобы разграничить шаблоны для отдельных проектов, можно создавать для шаблонов каждого приложения отдельный каталог. Например, в нашем случае у нас одно приложение - firstapp. Создадим для него в папке templates каталог firstapp (по имени приложения). И в этом каталоге определим файл home.html.



Теперь изменим файл views.py нашего приложения, указав в нем новый путь к странице сайта, которую нужно выдать пользователю при его обращении к домашней странице сайта - home.html

```
def index(request):
    return render(request, "firstapp\home.html")
```

Стоит отметить, что теперь в пути к шаблону страницы показывается и папка, в которой он находится: firstapp/home.html.

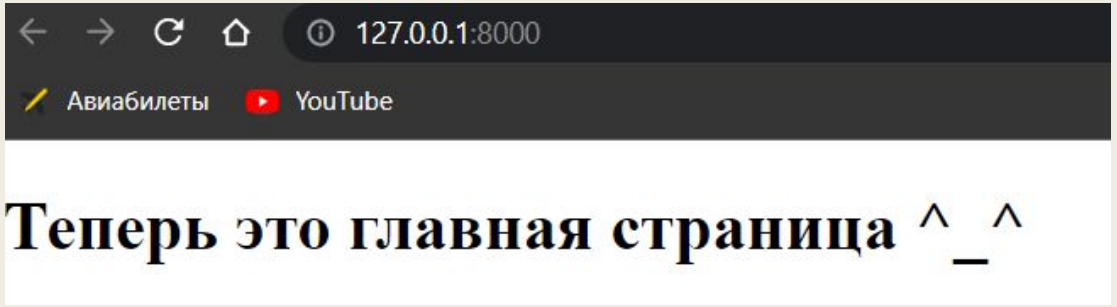


Класс TemplateResponse

Ранее для загрузки (вызова) шаблона применялась функция render (), что является наиболее распространенным вариантом. Однако мы также можем использовать класс TemplateResponse (шаблонный ответ). Функция def index (request) при использовании класса TemplateResponse будет выглядеть следующим образом.

```python
from django.template.response import TemplateResponse

def index(request):
    return TemplateResponse(request, "firstapp\home.html")
```

Результат работы приложения с использованием класса TemplateResponse будет таким же, как и при использовании функции render()



Передача данных в шаблоны

Одним из преимуществ шаблонов является то, что мы можем передать в них пользователю различные данные, которые будут динамически подгружены из базы данных через представления (views). Для вывода данных в шаблоне могут использоваться различные способы. Так, вывод самых простых данных может осуществляться с помощью двойной пары фигурных скобок:

{{название_объекта}}

Вернемся к нашему проекту test1, содержащему приложение firstapp. Добавим в папку templates\firstapp еще один шаблон страницы - index_app1.html

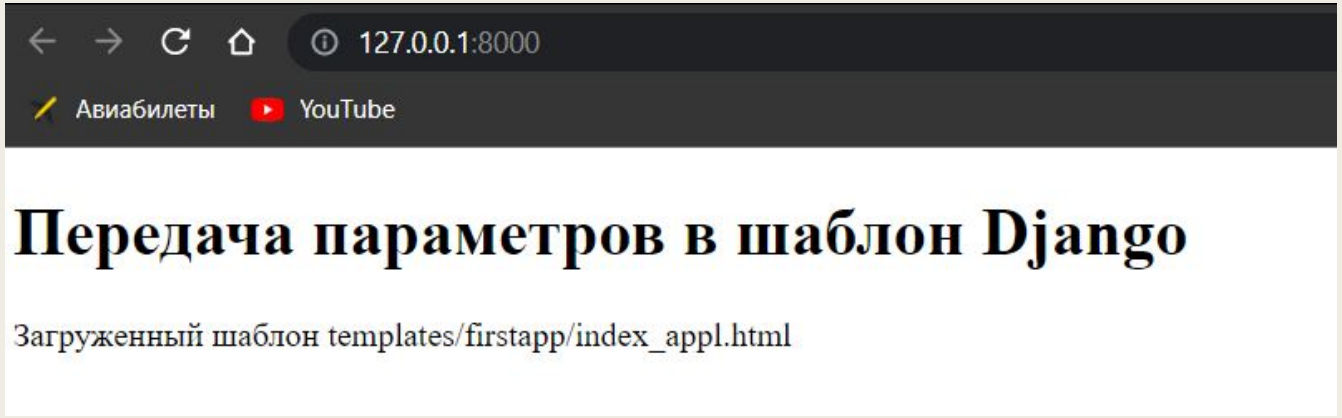
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет</title>
</head>
<body>
    <h1>{{header}}</h1>
    <p>{{message}}</p>
</body>
</html>
```

Как можно видеть, мы ввели здесь две новые переменные: header и message. Эти переменные и будут получать значения из представления (view).

Чтобы из функции-представления передать данные в шаблон применяется еще один (третий) параметр в функции render, который называется контекст (context). В качестве примера изменим в файле views.py функцию def index () следующим образом:

```
def index(request):
    # return render(request, "firstapp/home.html")
    data = {"header": "Передача параметров в шаблон Django",
            "message": "Загружен шаблон templates/firstapp/index_app1.html"}
    return render(request, "firstapp/index_app1.html", context=data)
```

В нашем шаблоне мы использовали две переменные: header и message, соответственно, словарь data, который передается в функцию render через параметр context, теперь содержит два значения с ключами: header и message.



Передача в шаблон сложных данных

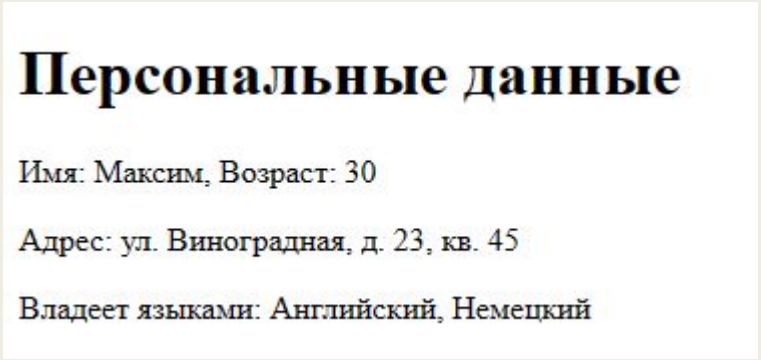
Рассмотрим теперь передачу пользователю через шаблон более сложных данных.

```
def index(request):
    header = "Персональные данные"                          # обычная переменная
    langs = ["Английский", "Немецкий", "Испанский"]         # массив
    user = {"name": "Максим,", "age": 30}                   # словарь
    addr = ("Виноградная", 23, 45)                          # кортеж
    data = {"header": header, "langs": langs, "user": user, "address": addr}
    return render(request, "index.html", context=data)
```

Здесь мы создали несколько переменных: header, langs, user и addr и все эти переменные обернули в словарь data. Затем в функции render () передали этот словарь третьему параметру - context.

Теперь нам нужно изменить сам шаблон templates\index.html, чтобы он смог принять новые данные.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Передача сложных данных</title>
</head>
<body>
    <h1>{{header}}</h1>
    <p>Имя: {{user.name}} Возраст: {{user.age}}</p>
    <p>Адрес: ул. {{address.0}}, д. {{address.1}}, кв. {{address.2}}</p>
    <p>Владеет языками: {{langs.0}}, {{langs.1}}</p>
</body>
</html>
```



Поскольку объекты langs и address представляют, соответственно, массив и кортеж, то мы можем обратиться к их элементам через индексы, как мы это делали с ними в любом программном коде на Python. Например, первый элемент кортежа адреса (address) мы можем получить следующим образом: address.0. Соответственно, к элементам массива, содержащего языки (langs) можно обращать по номеру элемента в массиве: langs.0, langs.1. Поскольку объект с информацией о пользователе (user) представляет словарь, то аналогичным образом мы можем обратиться к его элементам по ключам словаря name и age следующим образом: user. name, user. age

Статичные файлы

Пришло время ближе познакомиться со статическими файлами. Статическими файлами называются все файлы каскадных таблиц стилей (Cascading Style Sheets, CSS), изображений, а также скриптов javascript, - т. е. файлы, которые не изменяются динамически и содержание которых не зависит от контекста запроса и одинаково для всех пользователей. Можно воспринимать их как своего рода «макияж») для веб-страниц. Для придания привлекательности информации, выводимой на HTML-страницах, используются различные стили форматирования текстов, оформленные в виде каскадных таблиц ctw,ей (CSS) с помощью их специального языка. Объявление стиля состоит из двух частей: селектора и объявления. В HTML имена элементов нечувствительны к регистру, поэтому в селекторе значение h1 работает так же, как и ю. Объявление состоит из двух частей: имени свойства (например, color) и значения свойства (например, grey). Селектор сообщает браузеру, какой именно элемент форматировать, а в блоке объявления (код в фигурных скобках) указываются форматирующие команды - свойства и их значения.

Стили могут быть следующих видов:
□ встроенные;
□ внутренние;
□ внешняя таблица стилей.

Внутренние стили имеют приоритет над внешними таблицами стилей, но уступают встроенным стилям, заданным через атрибут style. При использовании встроенных стилей CSS-код располагается в HTML-файле, непосредственно внутри тега элемента с помощью атрибута style:

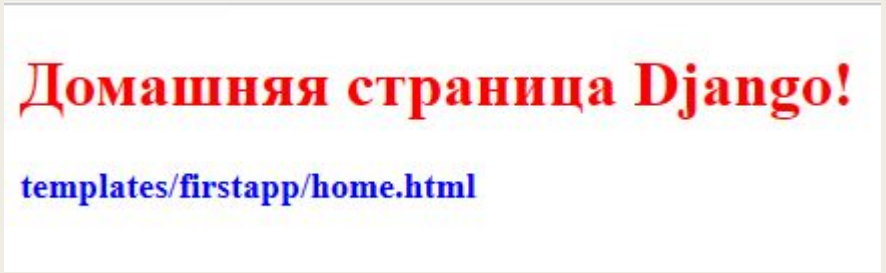
 Этот текст будет красного цвета!

Рассмотрим использование стилей на нескольких примерах. Для этого вернемся к нашему проекту hello и перейдем к файлу test1\templates\firstapp\home.html.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>home</title>
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
</body>
</html>
```

Изменим стиль выводимого текста: первую строку выведем красным цветом, а вторую - синим. Для этого изменим текст файла home.html следующим образом.

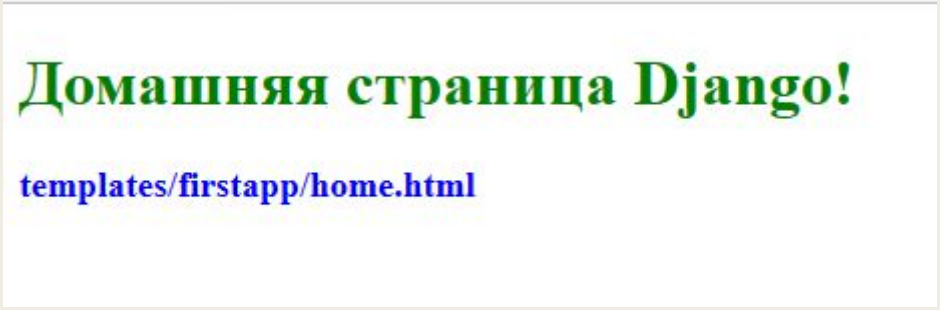
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>home</title>
</head>
<body>
    <font style="color: red">
    <h1>Домашняя страница Django!</h1>
    </font>
    <font style="color: blue; font-size: 12px">
    <h2>templates/firstapp/home.html</h2>
    </font>
</body>
</html>
```



Использовать встроенные стили достаточно просто, но не совсем удобно. Если через какое-то время потребуется изменить цвет всех заголовков на всех страницах, то нужно будет менять код на каждой из страниц.

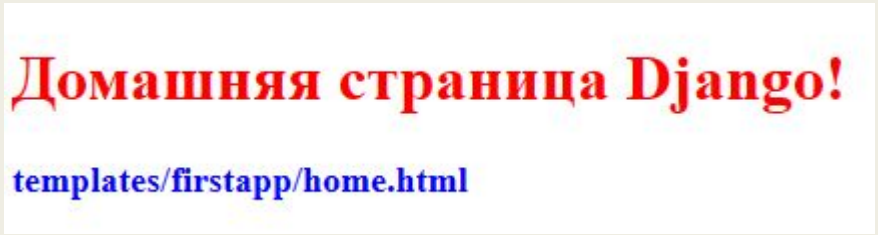
Внутренние стили отличаются от встроенных стилей тем, что они встраиваются в раздел <head> </head> HTML-документа. Изменим текст файла home.html следующим образом:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <style>h1{color: green;}</style>
    <meta charset="UTF-8">
    <title>home</title>
</head>
<body>
```



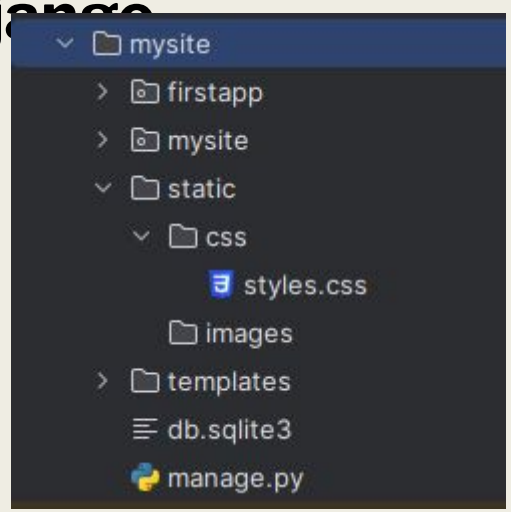
Несмотря на то что перед тегом h1 во встроенном стиле задан красный цвет текста, выведен он в зеленом. То есть здесь задействован тот стиль (внутренний), который был указан в заголовке страницы. Проверим теперь в работе приоритетность внутренних стилей над встроенными.

```
<head>
    <style>h1{color: green;}</style>
    <meta charset="UTF-8">
    <title>home</title>
</head>
<body>
    <style>h1{color: red;}</style>
    <h1>Домашняя страница Django!</h1>
    <font style="color: blue; font-size: 12px">
    <h2>templates/firstapp/home.html</h2>
    </font>
</body>
</html>
```



Как можно видеть (при выводе цветного изображения на экран компьютера), заголовок страницы имеет красный цвет, что подтверждает приоритет встроенного стиля над внутренним.

Использование статичных файлов в приложениях на Django



Добавим в корневую папку проекта новую вложенную папку static.

А чтобы не смешивать в одной папке различные типы статичных файлов, создадим для каждого типа файлов отдельную папку. В частности, создадим в папке ststic папку images - для изображений, и папку css - для таблиц стилей. Подобным образом можно создавать папки и для других типов файлов. Теперь структура нашего проекта будет выглядеть так:

Для определения пути к статическим файлам используются выражения такого типа:

{% static "путь к файлу внутри папки static" %}

Так что возьмем шаблон home.html из папки templates\firstapp и изменим его следующим образом:

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>home</title>
    <link href="{% static 'css/styles.css' %}" rel="stylesheet">
</head>
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
</body>
</html>
```

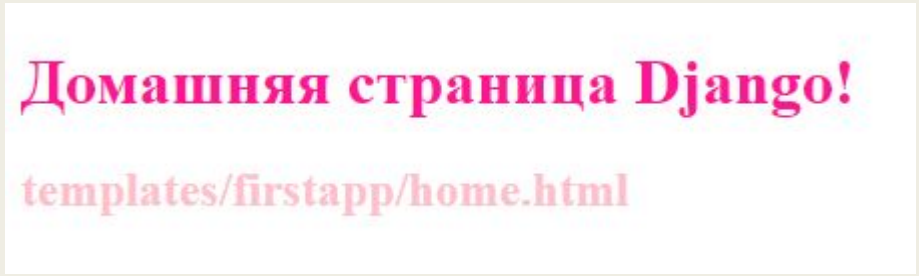
Чтобы файлы из папки static могли использоваться в приложениях, надо указать путь к этой папке в файле settings.py, добавив в конец файла следующий код:

```
STATIC_URL = 'static/'

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

Далее добавим стилей в файл syles.css:

```
body h1 {color: deeppink;}
body h2 {color: pink;}
```

После этих изменений обратимся к странице home.html и получим следующий результат:



Изображения тоже являются статическими файлами. Для хранения изображений мы ранее создали папку images. Разместим в этой папке файл с любым изображением и дадим ему имя image1 .jpg. Теперь выведем его на нашей странице home.html, для чего модифицируем код страницы следующим образом:

```
<body>
    <h1>Домашняя страница Django!</h1>
    <h2>templates/firstapp/home.html</h2>
    <img src="{%static 'images/img1.jpg'%}">
</body>
</html>
```

Однако, если мы так запустим сервер, то мы увидим такую картинку:

Для того, чтобы изображение появилось, нам необходимо поместить на компьютере в указанную папку указанное изображение.

Использование класса TemplateView для вызова шаблонов HTML-страниц

В примерах предыдущих разделов, когда приходил запрос от браузера пользователя, система маршрутизации выбирала нужное представление (view), и уже оно вызывало шаблон для генерации ответа. Однако если нужно просто возвратить пользователю содержимое шаблона, то для этого необязательно определять функцию в представлении и обращаться к ней. Можно воспользоваться встроенным классом TemplateView и вызвать нужный шаблон, минуя обращение к представлению. Проверим, как это работает на простых примерах, для чего определим два шаблона about и contact в папке hello\templates\firstapp\.

about

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Здесь может быть ваше название</title>
</head>
<body>
  <h1>Сведения о компании</h1>
  <h2>"Пипалапупа"</h2>
</body>
</html>
```

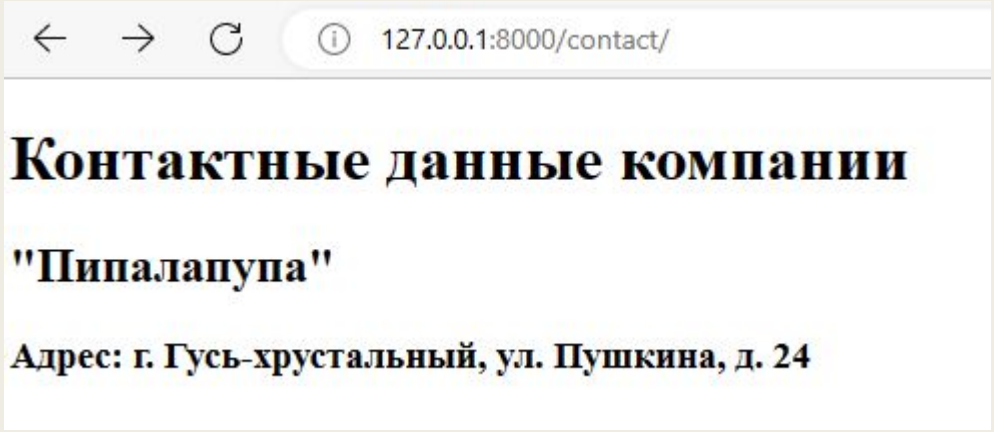
contact

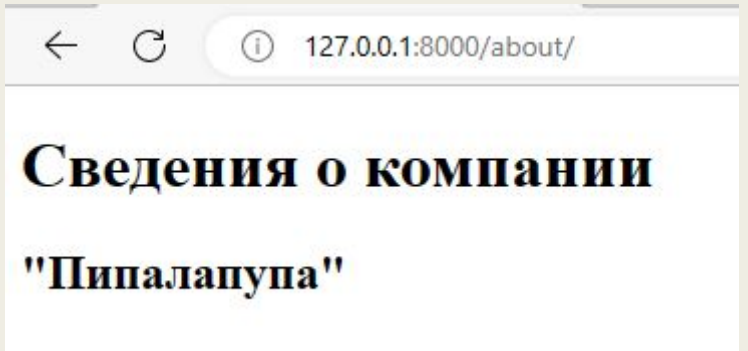
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Здесь может быть ваше название</title>
</head>
<body>
  <h1>Контактные данные компании</h1>
  <h2>"Пипалапупа"</h2>
  <h3>Адрес: г. Гусь-хрустальный, ул. Пушкина, д. 24</h3>
</body>
</html>
```

Осталось внести в файл urls.py следующие изменения

```python
from django.views.generic import TemplateView

urlpatterns = [
    path('', views.index),
    path('about/', TemplateView. as_view (template_name="firstapp/about .html")),
    path('contact/',TemplateView.as_view(template_name="firstapp/contact.html")),
    path('admin/', admin.site.urls),
]
```

В рассматриваемом программном коде сначала для вызова главной страницы сайта делается обращение к функции index () в представлении view, и уже функция index () вызывает главную страницу приложения firstapp/home.html. Затем в двух строках программного кода для вызова страниц firstapp\about.html и firstapp\contact.html используется класс TemplateView. По своей сути класс TemplateView предоставляет функциональность представления.



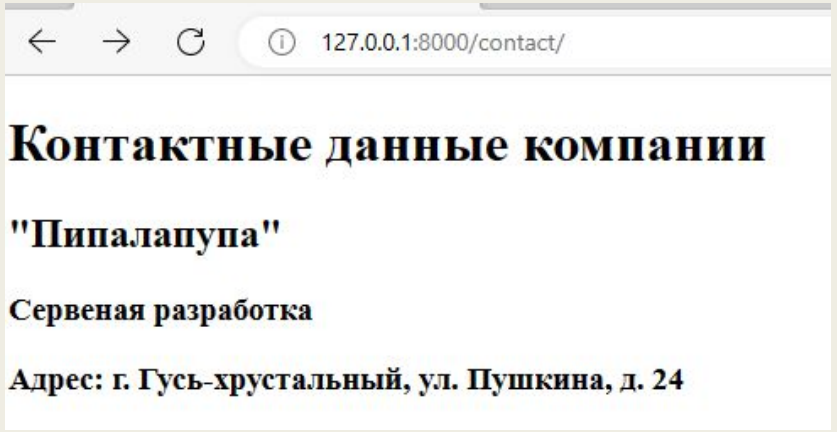


С помощью метода as_view() через параметр template_name задается путь к шаблону, который и будет использоваться в качестве ответа. С помощью параметра extra _ context в метод as _ view можно передать данные для их отображения в шаблоне. Данные должны представлять собой словарь. В качестве примера передадим в шаблон contact.html виды работ, которые выполняет компания. Для этого используем переменную work и следующим образом изменим код файла urls.py

```
urlpatterns = [
    path('', views.index),
    path('about/', TemplateView.as_view(template_name="firstapp/about.html")),
    path('contact/', TemplateView.as_view(template_name="firstapp/contact.html",
                                          extra_context={"work": "Сервеная разработка"})),
    path('admin/', admin.site.urls),
]
```

Здесь в шаблон contact.html передается объект work, который представляет строку со следующей информацией: "Разработка программных продуктов". И теперь мы можем использовать этот объект в шаблоне contact.html.

```
<body>
  <h1>Контактные данные компании</h1>
  <h2>"Пипалапупа"</h2>
  <h3>{{ work }}</h3>
  <h3>Адрес: г. Гусь-хрустальный, ул. Пушкина, д. 24</h3>
</body>
</html>
```



Конфигурация шаблонов HTML-страниц

За конфигурацию шаблонов проекта отвечает переменная TEMPLATES, расположенная в файле settings.py.

Здесь используются следующие переменные:

- BACKEND - указывает, что надо использовать шаблоны Django;
- DIRS - указывает на каталоги (папки) в проекте, которые будут содержать шаблоны;
- APP_DIRS- при значении true указывает, что поиск шаблонов будет производиться не только в каталогах (папках), указанных в параметре DIRS, но и в их подкаталогах (подпапках). Если такое поведение недопустимо, то можно установить значение False;
- OPTIONS - указывает, какие обработчики (процессоры) будут использоваться при обработке шаблонов

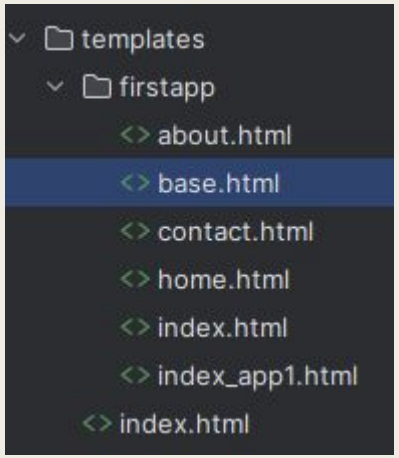
```
TEMPLATE_DIR = os.path.join(BASE_DIR, "templates")

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',
            ],
        },
    },
]
```

Расширение шаблонов HTML-страниц на основе базового шаблона

Хорошим тоном программирования считается формирование единообразного стиля сайта, когда веб-страницы имеют одни и те же структурные элементы: меню, шапку сайта (header), подвал (footer), боковую панель (sidebar) и т. д.

Конечно, можно сконфигурировать каждый шаблон по отдельности. Однако если возникнет необходимость изменить какой-то блок - например, добавить в общее меню еще один пункт, то придется менять все шаблоны, которых может быть довольно много. И в этом случае более рационально многократно использовать один базовый шаблон, который определяет все основные блоки страниц сайта. Попросту говоря, нужно создать некий базовый шаблон.

В качестве примера создадим базовый шаблон, который назовем base.html.




```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Заголовок{% endblok,title %}</title>
</head>
<body>
  <h1>{% block header %}{% endblok header %}</h1>
  <div>{% block content%}{% endblok content %}</div>
  <div>Подвал страницы</div>
</body>
</html>
```

Здесь с помощью элементов:
{% block название_блока %} {% endblok название_ блока %} определяются отдельные блоки шаблонов.

Когда другие шаблоны будут применять этот базовый шаблон, то они могут определить для блока ti tle какое-то свое содержимое. Подобным же образом здесь определены блоки header и content. Для каждого блока можно определить содержимое по умолчанию. Так, для блока title это строка заголовок. И если другие шаблоны, которые станут использовать этот шаблон, не определят содержимое для блока title, то этот блок будет использовать строку заголовок. Впрочем, содержимое по умолчанию для блоков определять не обязательно. Самих же блоков при необходимости можно определить сколько угодно. Как можно видеть, в базовом шаблоне также определен подвал страницы (footer).
Поскольку мы хотим сделать подвал общим для всех страниц, то мы его не определяем как отдельный блок, а задаем значение этой части страницы в виде некой константы. Теперь применим наш базовый шаблон.

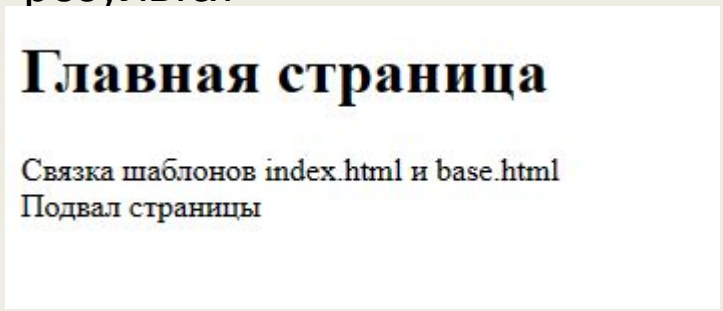
Теперь применим наш базовый шаблон. Создадим в каталоге templates\firstapp новый файл-шаблон главной страницы сайта с именем index.html и напишем в нем следующий код

```
{% extends "firstapp/base.html" %}
{% block title %}Index{% endblok title %}
{% block header %}Главная страница{% endblok header %}
{% block content%}Связка шаблонов index.html и base.html{% endblok content %}
```

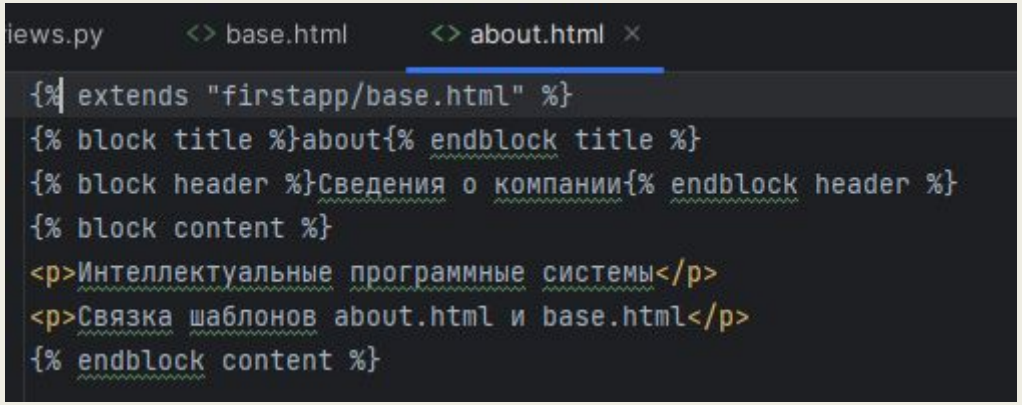
Здесь с помощью выражения {% extends "firstapp/base.html" %} мы определили, что эта страница будет формироваться (расширяться) на основе базового шаблона с именем base.html, который находится в каталоге шаблонов приложения firstapp (по пути: firstapp\Ьase.html). Затем задали содержимое для блоков title, header и content. Эти значения и будут переданы в базовый шаблон (base.html). Стоит отметить, что не обязательно указывать содержимое для всех блоков базового шаблона.
Изменим следующим образом код функции def index () в представлении view.

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

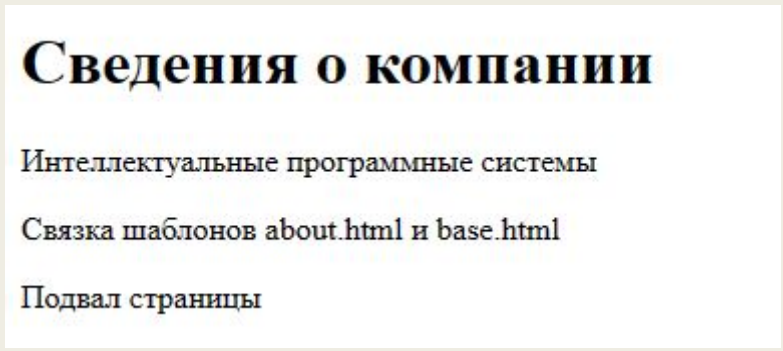
В итоге по связке шаблонов index.html и base.html в браузере будет выдан следующий результат



Теперь используем тот же базовый шаблон base.html для формирования другой страницы сайта - about.html.



Здесь с помощью выражения {% extends "firstapp/base.html" %} мы определили, что эта страница будет формироваться (расширяться) на основе базового шаблона с именем base.html, который находится в каталоге шаблонов приложения firstapp (по пути: firstapp\base.html). Затем задали содержимое для блоков title, header и content. Эти значения и будут переданы в базовый шаблон (base.html).



Теперь используйте все тот же базовый шаблон base.html для формирования другой страницы сайта - contact.html. Измените файл contact.html по своему усмотрению и желанию.


кажетс
я
конец
лекции