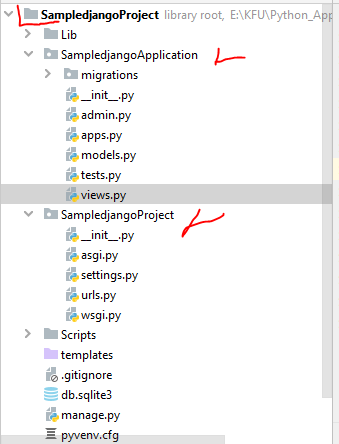
# Lecture 2. Шаблоны

## Создание и использование шаблонов

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

Итак, в прошлых темах был создан следующий проект:



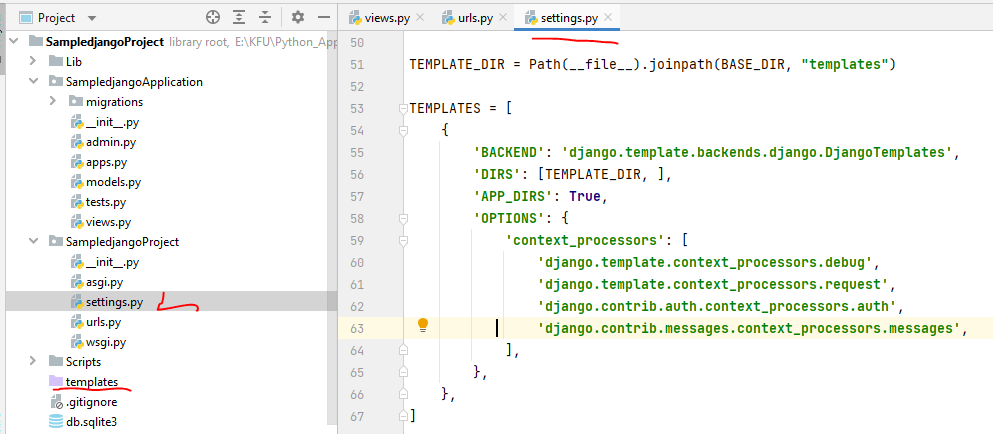
Проект называется SampledjanoApplication. И в нем определено одно приложение - SampledjangoProject.

Теперь добавим шаблоны. Для этого определим в корневой папке проекта новый каталог templates. Вообще само имя каталога может быть любым, но, как правило, это именно templates. Теперь нам надо указать, что этот каталог будет использоваться в качестве хранилища шаблонов. Для этого откроем файл settings.py. В этом файле настройка шаблонов произодится с помощью переменной TEMPLATES:

|  |
| --- |
| TEMPLATES = [  {  **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  **'DIRS'**: [BASE\_DIR / **'templates'**],  **'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'**,  ],  },  }, ] |

Параметр DIRS задает набор каталогов, которые хранят шаблоны. Но по умолчанию он пуст. Теперь изменим данный кусок кода следующим образом:

|  |
| --- |
| TEMPLATE\_DIR = Path(\_\_file\_\_).joinpath(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'**,  ],  },  }, ] |



Затем в папке templates определим новый файл index.html со следующим кодом:

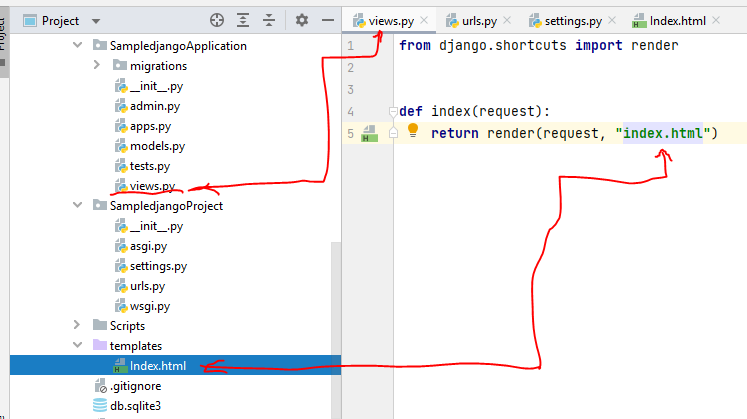
|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**>  <**h1**>Welcome to KFU!</**h1**> </**body**> </**html**> |

По сути это обычная веб-страница, которая содержит код html. Теперь используем эту страницу для отправки ответа пользователю. И для этого перейдем в приложении SampledjangoApplication к файлу views.py, который определяет функции для обработки запроса. Изменим этот файл следующим образом:

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  **return** render(request, **"index.html"**) |

Из модуля django.shortcuts импортируется функция render.

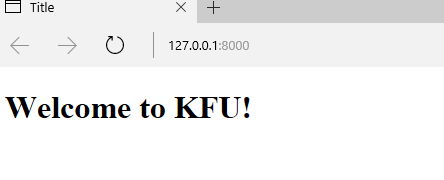
Функция index вызывает функцию render, которой передаются объект запроса request и путь к файлу шаблона в рамках папки templates - "index.html".



В файле urls.py в главном проекте пропишем сопоставление функции index с запросом к корню веб-приложения:

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path **from** SampledjangoApplication **import** views urlpatterns = [  path(**''**, views.index), ] |

И запустим проект на выполнение и перейдем к приложению в браузере:



Таким образом, мы можем передавать в шаблоны данные и возвращать пользователю динамически генерируемые веб-страницы.

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 определять для шаблонов каждого приложения отдельный каталог. Например, в нашем случае у нас одно приложение - firstapp. Для него определим в папке templates каталог firstapp (по имени приложения). И в этом каталоге определим также файл home.html:

|  |
| --- |
| <!DOCTYPE **html**> <**html lang="en"**> <**head**>  <**meta charset="UTF-8"**>  <**title**>Title</**title**> </**head**> <**body**> <**h1**>Hello from Home.html!</**h1**> </**body**> </**html**> |

Теперь изменим файл views.py в приложении:

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  **return** render(request, **"firstapp/home.html"**) |

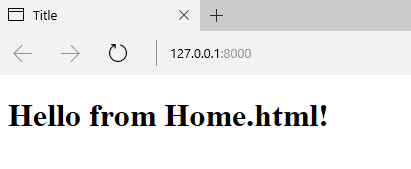
Стоит отметить, что теперь в пути к шаблону также указывается папка, в которой он находится: "firstapp/index.html".

**TemplateResponse**

Выше для генерации шаблона применялась функция render(), которая является наиболее распространенным вариантом. Однако также мы можем использовать класс TemplateResponse:

|  |
| --- |
| **from** django.template.response **import** TemplateResponse  **def** index(request):  **return** TemplateResponse(request, **"firstapp/home.html"**) |

Результат будет тот же самый.



## Передача данных в шаблоны

Одним из преимуществ шаблонов является то, что мы можем передать в них динамически из представлений различные данные. Для вывода данных в шаблоне могут использоваться различные способы. Для вывода самых простых данных применяется двойная пара фигурных скобок:

|  |
| --- |
| {{ название\_объекта }} |

Например, пусть в проекте у нас есть папка templates, в которой содержится шаблон index.html со следующим кодом:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h2**>{{ **header** }}</**h2**>  <**p**>{{ **message** }}</**p**> </**body**> </**html**> |

Здесь используется две переменных: message и header. Они будут передаваться из представления.

Чтобы из функции-представления передать данные в шаблон применяется третий параметр функции render, который еще называется контекст. Например, изменим файл views.py следующим образом:

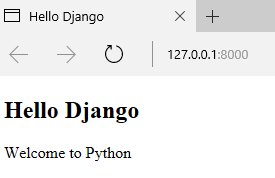
|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  data = {**"header"**: **"Hello Django"**, **"message"**: **"Welcome to Python"**}  **return** render(request, **"index.html"**, context=data) |

В шаблоне используются две переменных, соответственно словарь, который передается в функцию render через параметр context, теперь содержит два значения с ключами header и message.

Изменим файла urls.py следующим образом:

|  |
| --- |
| **from** django.contrib **import** admin **from** django.urls **import** path **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, views.index), ] |

В результате при обращении к корню веб-приложения мы увидим следующий вывод в браузере:



**Передача сложных данных**

Рассмотрим передачу более сложных данных. Допустим, в представлении передаются следующие данные (Файл Views.py):

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  header = **"Personal Data"** *# обычная переменная* langs = [**"English"**, **"German"**, **"Spanish"**] *# массив* user = {**"name"**: **"Tom"**, **"age"**: 23} *# словарь* addr = (**"Абрикосовая"**, 23, 45) *# кортеж* data = {**"header"**: header, **"langs"**: langs, **"user"**: user, **"address"**: addr}  **return** render(request, **"index.html"**, context=data) |

В качестве третьего параметра в функцию render нам надо передать словарь, поэтому все данные оборачиваются в словарь и в таком виде передаются в шаблон.

В этом случае шаблон мог бы выглядеть, например, следующим образом:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h1**>{{ **header** }}</**h1**>  <**p**>Name: {{ **user**.**name**}} Age: {{**user**.**age**}}</**p**>  <**p**>Address: ул. {{**address**.0}}, д. {{**address**.1}}, кв.  {{**address**.2}}</**p**>  <**p**>Langs: {{**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 бы работали бы с ними в коде на Python, например, первый элемент кортежа address: address.0.

Подобным образом, поскольку объект user представляет словарь, то мы можем обратиться к его элементам по ключам name и age: {{ user.name}} {{user.age}}.

В итоге мы получим следующий вывод в веб-браузере:

**TemplateResponse**

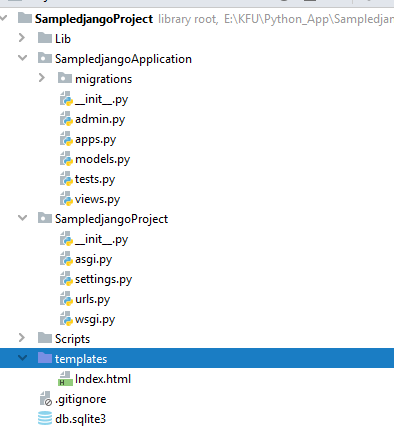
Если для генерации шаблона применяется класс TemplateResponse, то в его конструктор также через третий параметр можно передать данные для шаблона:

|  |
| --- |
| **from** django.template.response **import** TemplateResponse  **def** index(request):  header = **"Personal Data"** *# обычная переменная* langs = [**"English"**, **"German"**, **"Spanish"**] *# массив* user = {**"name"**: **"Tom"**, **"age"**: 23} *# словарь* addr = (**"Абрикосовая"**, 23, 45) *# кортеж* data = {**"header"**: header, **"langs"**: langs, **"user"**: user,  **"address"**: addr}  **return** TemplateResponse(request, **"index.html"**, data) |

## Статические файлы

Веб-приложение, как правило, использует различные статические файлы - изображения, файлы стилей css, скриптов javascript и так далее. Рассмотрим, как мы можем использовать подобые файлы.

Возьмем проект из прошлой темы, где обсуждалось создание шаблонов.



Добавим в корневую папку проекта новый каталог static. Чтобы не сваливать все статические файлы в кучу, определим для каждого типа файлов отдельные папки. В частности, создадим в папке static для изображений каталог images, а для стилей - каталог css. Подобным образом можно создавать папки и для других типов файлов.

В папку static/images добавим какое-нибудь изображение, а в папке static/css определим новый файл styles.css.

Определим в файле styles.css следующий код:

|  |
| --- |
| **body**{ **font-family**: **Verdana**;} **h1**{**color**:**navy**;} **img**{**width**:350**px**;} |

Теперь используем эти файлы в шаблоне. Для этого в начале файла шаблона необходимо определить инструкцию

|  |
| --- |
| {% **load static** %} |

При этом данный код должен идти после тега DOCTYPE.

Для определения пути к статическим файлам используются выражения типа

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

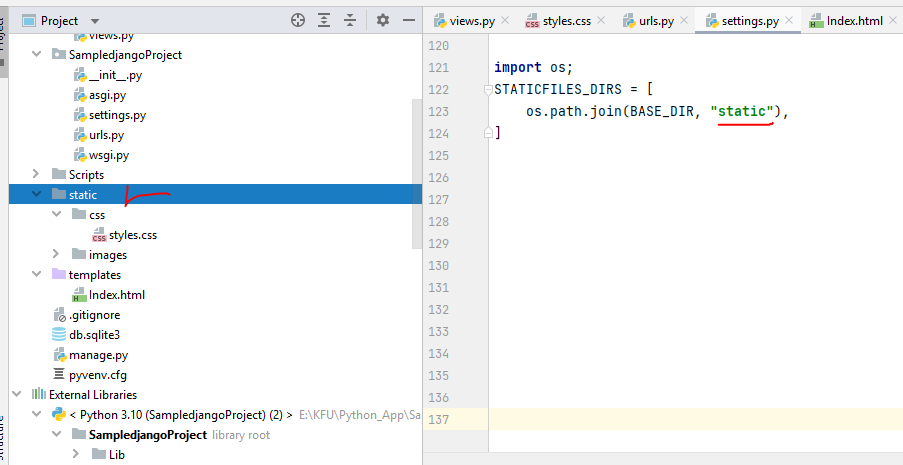
Так, возьмем шаблон index.html из папки templates и изменим его следующим образом:

|  |
| --- |
| <!DOCTYPE **html**> {% **load static** %} <**html**> <**head**>  <**meta charset="utf-8"** />  <**link rel="stylesheet" href=  "**{% **static "css/styles.css"** %}**"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h1**>Another image</**h1**>  <**img src="**{% **static "images/navruz.jpg"** %}**"  alt="Stones along the river"** > </**body**> </**html**> |

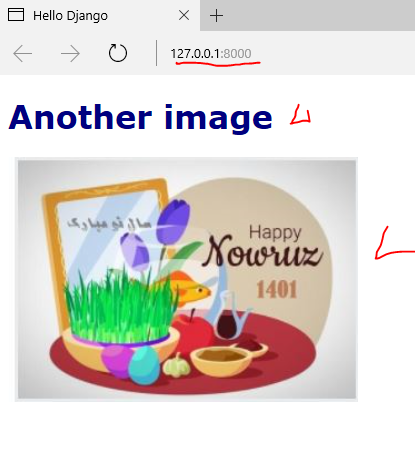
Чтобы файлы из папки static могли использоваться в приложениях, надо указать путь к данной папке в файле settings.py, добавив в конец файла следующий код:

|  |
| --- |
| **import** os; STATICFILES\_DIRS = [  os.path.join(BASE\_DIR, **"static"**), ] |

Переменная STATICFILES\_DIRS указывает, в каких папках искать статические файлы.



При запуске приложения шаблон index.html будет генерироваться в следующую веб-страницу, которая будет использовать изображение и применять стили:



**TemplateView**

В предыдущих темах, когда приходил запрос система маршрутизации выбирала нужное представление, и то уже в свою очередь использовало шаблон для генерации ответа. Hо в принципе если в ответ на запрос нам просто надо возвратить пользователю **содержимое шаблона**, то для этого необязательно определять функцию-представления. И можно воспользоваться встроенным классом TemplateView.

Так, определим несколько простейших шаблонов. Пусть это будет файл about.html со следующим кодом:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h1**>About</**h1**> </**body**> </**html**> |

И также определим шаблон contact.html со следующим кодом:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h1**>Contact</**h1**> </**body**> </**html**> |

Изменим файл urls.py следующим образом:

|  |
| --- |
| **from** django.urls **import** path **from** django.views.generic **import** TemplateView  urlpatterns = [  path(**'about/'**, TemplateView.as\_view(template\_name=**"about.html"**)),  path(**'contact/'**, TemplateView.as\_view(template\_name=**"contact.html"**)), ] |

Фактически TemplateView сам по себе предоставляет функционалность представления. С помощью метода as\_view() через параметр template\_name устанавливается путь к шаблону, который будет использоваться в качестве ответа.

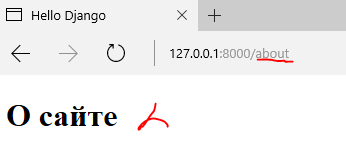
С помощью параметра extra\_context в метод as\_view можно передать данные для шаблона. Данные должны представлять словарь. Например:

|  |
| --- |
| **from** django.urls **import** path **from** django.views.generic **import** TemplateView  urlpatterns = [  path(**'about/'**, TemplateView.as\_view(template\_name=**"about.html"**,  extra\_context={**"header"**: **"О сайте"**})),  path(**'contact/'**, TemplateView.as\_view(template\_name=**"contact.html"**)), ] |

Здесь в шаблон "about.html" передается объект header, который представляет строку "О сайте". И в этом случае мы можем использовать этот объект в шаблоне:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**h1**>{{ **header** }}</**h1**> </**body**> </**html**> |

Результат



## Конфигурация шаблонов

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

|  |
| --- |
| TEMPLATES = [  {  **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  **'DIRS'**: [],  **'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, мы можем создать для каждого приложения подкаталог, в который помещаются шаблоны для конкретного приложения.

В этом случае в файле setting.py мы можем определить путь к шаблонам следующим образом:

|  |
| --- |
| TEMPLATE\_DIR = Path(\_\_file\_\_).joinpath(BASE\_DIR, **"templates"**)  TEMPLATES = [  {  **'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,  **'DIRS'**: [TEMPLATE\_DIR, ],  **'APP\_DIRS'**: **True**,  ……………. |

В этом случае берется определенная в начале файла settings.py переменная BASE\_DIR:

|  |
| --- |
| BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent |

которая представляет путь к проекту, и к этому пути добавляется папка templates.

**Расширение шаблонов**

Нередко шаблоны должны иметь **одинаковую базовую структуру**, одни и те же блоки, при этом определять для отдельных блоков различное содержимое. Это позволяет сформировать единообразный стиль сайта, когда веб-страницы имеют одни и те же структурные элементы - меню, хедер, футер, сайдбары и так далее.

В этом случае мы можем определять все шаблоны по отдельности. Однако если возникнет необходимость изменить какой-то блок, например, добавить в общее меню еще один пункт, тогда придется менять все шаблоны, коих может быть довольно много. И в этом случае оптимальное повторно использовать один базовый шаблон, который определяет все основные блоки.

Например, определим шаблон, который назовем **base.html**:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>{% **block title** %}Default title{% **endblock title** %}</**title**> </**head**> <**body**>  <**h1**>{% **block header** %}{% **endblock header** %}</**h1**>  <**div**>{% **block content**%}{% **endblock content** %}</**div**>  <**div**>Footer</**div**> </**body**> </**html**> |

С помощью элементов {% block название\_блока %}{% endblock название\_блока %} определяются отдельные блоки шаблонов. При этом для каждого блока определяется открывающий элемент {% block название\_блока %} и закрывающий элемент {% endblock название\_блока %}.

Например, блок **title**:

|  |
| --- |
| {% **block title** %}Default title{% **endblock title** %} |

Когда другие шаблоны будут применять данный шаблон, то они могут определить для блока title какое-то свое содержимое.

Для каждого блока можно определить содержимое по умолчанию. Так, для блока title это строка "Default title". И если другие шаблоны, которые будут использовать данный шаблон, не определяет содержимое для блока title, то данный блок будет использовать строку "Default title"

Подобным образом здесь определены блоки header и content. Содержимое по умолчанию для блоков определять не обязательно. Самих блоков при необходимости можно определить сколько угодно.

Кроме того, в базовом шаблоне определен футер, однако покольку, допустим, мы хотим сделать его общим для всех страниц, то для него не определен отдельный блок.

Теперь применим этот базовый шаблон. Например, создадим новый шаблон **index.html**:

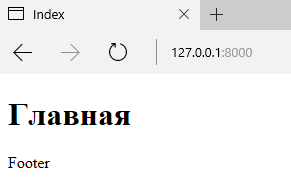
|  |
| --- |
| {% **extends "base.html"** %} {% **block title** %}Index{% **endblock title** %} {% **block header** %}Главная{% **endblock header** %} |

С помощью выражения {% extends "base.html" %} определяем, какой базовый шаблон будет расширяться. Затем определяется содержимое для блоков title и header. Стоит отметить, что необязательно указывать содержимое для всех блоков базового шаблона.

В файле urls.py в главном проекте пропишем сопоставление функции index с запросом к корню веб-приложения:

|  |
| --- |
| **from** django.urls **import** path **from** django.views.generic **import** TemplateView **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, TemplateView.as\_view(template\_name=**"Index.html"**)),  path(**'about/'**, TemplateView.as\_view(template\_name=**"about.html"**)),  ] |

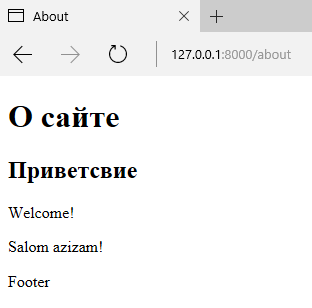
В итоге по шаблону index.html будет создана следующая веб-страница:



Создадим также шаблон **about.html**:

|  |
| --- |
| {% **extends "base.html"** %} {% **block title** %}About{% **endblock title** %} {% **block header** %}О сайте{% **endblock header** %} {% **block content** %}  <**H2**>Приветсвие</**H2**>  <**p**>Welcome!</**p**>  <**p**>Salom azizam!</**p**> {% **endblock content** %} |

Этот шаблон также расширяет базовый шаблон base.html. В отличие от index.html здесь также определяется содержимое для блока content. В результате мы получим следующую веб-страницу:



В итоге если нам потребуется изменить структуру всех веб-страниц сайта, добавить новые элементы или убрать старые, то достаточно будет изменить один базовый шаблон.

В этом случае файл urls.py имеет следующий вид.

|  |
| --- |
| **from** django.urls **import** path **from** django.views.generic **import** TemplateView **from** SampledjangoApplication **import** views  urlpatterns = [  path(**''**, TemplateView.as\_view(template\_name=**"Index.html"**)),  path(**'about/'**, TemplateView.as\_view(template\_name=**"about.html"**)),  path(**'contact/'**, TemplateView.as\_view(template\_name=**"contact.html"**)), ] |

## Встроенные теги

Django предоставляет возможность использовать в шаблонах ряд специальных тегов, которые упрощают вывод некоторых данных. Рассмотрим некоторые наиболее используемые теги.

Даты

Тег {% now "формат\_данных" %} позволяет вывести системное время. В качестве параметра тегу now передается формат данный, который указывает, как форматировать время и дату.

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  <**p**>{% **now "Y"** %}</**p**>  <**p**>{% **now "F j Y"** %}</**p**>  <**p**>{% **now "N, j, Y"** %}</**p**>  <**p**>{% **now "N j, Y, P"** %}</**p**> </**body**> </**html**> |

Символ "Y" передает год в виде четырех цифр, "y" - берет из года последние две цифры. "F" - полное название месяца, "N" -сокращенное название месяца. "j" - день месяца. "P" - время. Все возможные форматы для вывода даты и времени можно посмотреть в документации

**if..else**

Тег {% if %} {% endif %} позволяет выводить в шаблоне определенное содержимое в зависимости от некоторого условия. В качестве параметра тегу if передается выражение, которое должно возвращать True или False.

Например, пусть в представлении передаются в шаблон некоторые значения:

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  data = {**"n"**: 5}  **return** render(request, **"index.html"**, context=data) |

Изменим файла urls.py следующий образом:

|  |
| --- |
| **from** django.urls **import** path **from** firstApp **import** views  urlpatterns = [  path(**''**, views.index), ] |

В шаблоне в зависимости от значения переменной n мы можем выводить определенный контент:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**>  {% **if n** > 0 %}  <**p**>Число положительное</**p**>  {% **endif** %} </**body**> </**html**> |

То есть если n больше 0, то будет выводиться, что число положительное. Если n меньше или равно 0, ничего не будет выводиться.

С помощью дополнительного тега {% else %} можно вывести контент в случае, если условие после if равно False:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**> {% **if n** > 0 %}  <**p**>Число положительное</**p**> {% **else** %}  <**p**>Число отрицательное или равно нулю</**p**> {% **endif** %} </**body**> </**html**> |

С помощью тега {% elif %} можно проверить дополнительные условия, ели услове в if равно False:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**> {% **if n** > 0 %}  <**p**>Число положительное</**p**> {% **elif n** < 0 %}  <**p**>Число отрицательное</**p**> {% **else** %}  <**p**>Число равно нулю</**p**> {% **endif** %} </**body**> </**html**> |

**Циклы**

Тег for позволяет создавать циклы. Этот тег принимает в качестве параметра некоторую коллекцию и пробегается по этой коллекции, обрабатывая каждый ее элемент.

|  |
| --- |
| {% **for element in collection** %}  {% **endfor** %} |

Например, пусть из представления в шаблон передается некоторый массив:

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  langs = [**"English"**, **"German"**, **"French"**, **"Spanish"**, **"Chinese"**]  **return** render(request, **"index.html"**, context={**"langs"**: langs}) |

Выведем элементы массива langs в шаблоне с помощью тега for:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**> <**ul**>  {% **for lan in langs** %}  <**li**>{{ **lan** }}</**li**>  {% **endfor** %} </**ul**> </**body**> </**html**> |

Вполне возможно, что переданная из предствления в шаблон коллекция окажется пустой. На этот случай мы можем использовать тег {% empty %}:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**> <**ul**>  {% **for lan in langs** %}  <**li**>{{ **lan** }}</**li**>  {% **empty** %}  <**li**>Langs array is empty</**li**>  {% **endfor** %} </**ul**> </**body**> </**html**> |

**Определение переменных**

Тег {% with %} позволяет определить переменную и использовать ее внутри содержимого тега.

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Hello Django</**title**> </**head**> <**body**> {% **with name**=**"Tom" age**=29 %}  <**div**>  <**p**>Name: {{ **name** }}</**p**>  <**p**>Age: {{ **age** }}</**p**>  </**div**> {% **endwith** %} </**body**> </**html**> |

Тепер изменим файла view.py следующий образом:

|  |
| --- |
| **from** django.shortcuts **import** render  **def** index(request):  **return** render(request, **"index.html"**) |