Тема 2.4.6. Шаблоны. Передача данных в шаблоны.

Статичные файлы. Класс TemplateView. Базовый шаблон. Специальные теги.

Шаблоны служат для отображения данных на сайте.

С помощью шаблонов оформляется внешний вид приложения.

Шаблоны Django – это HTML-страницы, но тэги шаблонов Django позволяют вставлять в эти HTML-страницы результаты работы программ на Python.

Обычные же браузеры воспринимают только HTML.

Для хранения шаблонов в корневой папке проекта hello создадим папку **templates** и в файл **settings.py** внесем изменения:

В начале файле добавим **import os**

выше переменной TEMPLATES добавим

TEMPLATE_DIR = os.path.join(BASE_DIR, “templates”)

И в переменную внесем (выделено красным):

TEMPLATES = [

{

‘BACKEND’: ‘django.template.backend.django.DjangoTemplates’,

‘DIR’:[TEMPLATE_DIR],

…….

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

В папке templates создадим файл index.html (выполним команду template/File/HTML File/New HTML File).

По умолчанию он содержит код:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

</body>
</html>
```

Изменим код по умолчанию следующим образом:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ура! Мы работаем с Django! </title>
</head>
<body>
    <h1>Это главная страница django</h1>
   <h2>templates/index.html</h2>
</body>
</html>
```

Функция render().

Отредактируем в файле views.py приложения firstapp функцию **index** для обработки запроса пользователя:

def index(request)):

    return render(request, ”index.html”)

```
from django.shortcuts import render
from django.http import HttpResponse


def index(request):
    """
    :param request:
    :return:
    """
    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 проекте hello проверим наличие сопоставления функции index с запросом пользователя:

urlpatterns = [

path(‘’, views.index),

…….

]

Для обращения к корню веб-приложения прописан маршрут

path(‘’, views.index)

```
from django.contrib import admin
from django.urls import path, re_path
from firstapp import views


urlpatterns = [
    path('', views.index),
    re_path(r'^about', views.about),
    re_path(r'^contact', views.contact),
    path('products/<int:productid>/', views.products),
```

```
    path('users/', views.users),
]
```

Запустим сервер: python manage.py runserver

В браузере перейдем по адресу http://127/0/0/1:800 на главную страницу сайта.

На странице отобразится:

Это главная страница django

templates/index.html

Каждое из приложений проекта django может иметь свой набор шаблонов.

Для шаблонов каждого приложения надо создавать отдельную папку.

Для приложения firstapp в папке templates создаем каталог firstapp и в этом каталоге создадим файл home.html:

<!DOCTYPE html>

<html lang = ”en”>

<head>

<meta charset = ”UTF-8”>

<title>”Это файл Hello!” </title>

</head>

<body>

<h1>Домашняя страница Django</h1>

<h2>templates/firstapp/home.html</h2>

</body>

</html>

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

</body>
</html>
```

В файле views.py приложения firstapp изменим функцию index():

def index(request):

    return render(request, ”firstapp/home.html”)

Запустим сервер: python manage.py runserver

В браузере перейдем по адресу http://127/0/0/1:800 на новую домашнюю страницу сайта.

На странице отобразится текст и путь к шаблону страницы:

Домашняя страница Django

templates/firstapp/home.html

Класс TemplateResponse (шаблонный ответ) - альтернатива функции render()

from django.shortcuts import **render**

def index(request):

    return **render**(request, “firstapp/home.html”)

from django.template.response import **TemplateResponse**

def index(request):

    return **TemplateResponse**(request, “firstapp/home.html”)

Передача данных в шаблоны

Для вывода самых простых данных используется двойная пара фигурных скобок: **{{название объекта }}**

В папке templates/firstapp создадим новый шаблон страницы – **index_app1.html**

Отредактируем код страницы и введем переменные **header** и **message**. Они будут получать значения из представления (view):

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

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

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

На странице http://127.0.0.1:8000/ отобразится:

Передача параметров в шаблон Django

Загружен шаблон templates/firstapp/index_app1.html

Передача в шаблон сложных данных

Для передачи пользователю через шаблон данных более сложных, чем простой текст, выполняем следующее.

Изменим функцию index в файле представлений (view).

```
def index(request):
    """
    :param request:
    :return:
    """
    header = "Персональные данные"      # обычная переменная
    avto = ["Мазда", "Вольво", "Круз"]  # массив
    user = {"name": "Иван", "age": 25}  # словарь
    addr = ("Русская", 25, 17)          # кортеж
    data = {"header": header, "avto": avto, "user": user, "addr": addr}
    return render(request, "firstapp/index.html", context=data)
```

Изменим шаблон templates/firstapp/index.html, чтобы он мог принять новые данные:

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

На странице сайта отобразится:

Персональные данные

Имя: Иван, Возраст: 25

Адрес: Русская, д.: 25, кв.17

Владеет машинами: Мазда, Вольво, Круз

В случае использования в функции index класса TemplateResponse в программный код нужно добавить:

from django.template.response import TemplateResponse

и в операторе return вместо render написать TemplateResponse

Статичные файлы

Для оформления веб-страниц можно использовать статичные файлы. Это файлы изображений, скриптов, каскадных таблиц стилей. Они 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.

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

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 форматирующие команды — свойства и их значения.

По методам добавления стилей в документ различают три вида стилей.

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>
```

Глобальные стили CSS

Располагаются в контейнере <style>...</style>, расположенном в свою очередь в контейнере <head>...</head>. Например:

```
<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>
```

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:

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:

...

Подробнее:

Каскадные таблицы стилей CSS (htmlweb.ru)

Каскадные таблицы стилей

Основы CSS (html5book.ru)

Использование статичных файлов в приложениях Django

В корневую папку проекта добавим папку **static**. В ней создадим папку **images** для изображений и папку **css** для таблиц стилей.

Создадим в папке css файл styles.css (по щелчку правой кнопки мыши на папке css выбрать New/File и ввести имя styles.css).

Код в файле styles.css:

```
/* 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/firstapp/):**

<!DOCTYPE html>

{% load static %}

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<title>Привет!!! </title>

<link href=”{% static ‘css/styles.css’ %}” rel=”stylesheet”>

</head>

<body>

<h1>Домашняя страница Django</h1>

<h2>templates/firstapp/home.html</h2>

</body>

</html>

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

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 изменим функцию index:

```
def index(request):
    """
    :param request:
    :return:
    """

    return render(request, "firstapp/home.html")
```

После этих изменений страница home.html откроется с выделением тега h1 красным цветом, а тега h2 - зеленым цветом.

Вывод на странице home.html изображения

Поместим в папку images любое изображение с именем image1.jpg.

Для вывода этого изображения на странице home.html отредактируем код:

<!DOCTYPE html>

{% load static %}

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<title>Привет! </title>

<link href=”{% static ‘css/styles.css’ %}” rel=”stylesheet”>

</head>

<body>

<h1>Домашняя страница Django</h1>

<h2>templates/firstapp/home.html</h2>

<img src=”{% static ‘images/image1.jpg’ %}”>

</body>

</html>

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет!!! </title>
    <link href="{% static 'css/styles.css' %}" rel="stylesheet">
</head>
<body>
    <h1>Домашняя страница Django</h1>
   <h2>templates/firstapp/home.html</h2>
    <img src="{% static 'images/image1.jpg' %}">
</body>
</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;}

```
/* static/css/styles.css */
body h1 {color:red}
body h2 {color:green}
img{width:270px;}
```

Домашняя страница Django

templates/firstapp/home.html

Использование встроенного класса TemplateView

Если в ответ на запрос надо просто возвратить пользователю содержимое шаблона, то для этого необязательно определять функцию-представление. Можно воспользоваться встроенным классом **TemplateView.**

В папке hello\templates\firstapp определим 2 файла-шаблона.

файл-шаблон about.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Привет!</title>
</head>
<body>
    <h1>Сведения об IT-компании </h1>
   <h2>Новые технологии </h2>
    <h3>templates/firstapp/about.html</h3>

</body>
</html>
```

файл-шаблон 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>Новосибирск, Технопарк, офис 115 </h3>
    <h4>templates/firstapp/contact.html</h4>

</body>
</html>
```

Внесем также изменения в файл **urls.py:**

from django.views.generic import TemplateView

```
from django.urls import path, re_path
from firstapp import views
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('products/<int:productid>/', views.products),
    path('users/', views.users),
]
```

Через класс TemplateView вызываются страницы firstapp/about.html и firstapp/contact.html.

Данные для их 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:**

```
from django.contrib import admin
from django.urls import path, re_path
from firstapp import views
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",
         extra_context={"work": "Проектирование ИС"})),
    path('products/<int:productid>/', views.products),
    path('users/', views.users),
]
```

Объект **work** передается и может быть использован 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>
</html>
```

Конфигурация шаблонов HTML-страниц

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.

Если в проекте Hello создано приложение firstapp, то в папке templates создается папка firstapp и в ней хранятся все шаблоны этого приложения.

В файле setting.py путь к шаблонам определяется так:

```
TEMPLATE_DIR = os.path.join(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:

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

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

Расширение шаблонов HTML-страниц на основе базового шаблона

Если веб-страницы имеют одни и те же структурные элементы, то для того, чтобы сформировать единообразный стиль сайта, шаблоны должны иметь одинаковую базовую структуру, одни и те же блоки, и при этом определять для отдельных блоков различное содержимое.

В этом случае можно не определять шаблоны по отдельности, а сформировать и повторно использовать один базовый шаблон, который определяет все основные блоки.

Например, определим шаблон base.html:

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

Здесь 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 имеет структуру:

{% block title %} Заголовок {% 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 "Заголовок". И если другие шаблоны, которые будут использовать данный шаблон, не определяет содержимое для блока title, то данный блок будет использовать строку " Заголовок "

Подобным образом 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 Подвал страницы. Но если, допустим, мы хотим сделать его общим для всех страниц, то для него не определен отдельный блок.

Применим этот базовый шаблон.

Изменим в каталоге templates\firstapp файл-шаблон главной страницы сайта index.html:

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

{% extends " firstapp/base.html" %} – определяет, какой базовый шаблон будет расширяться.

Затем определяется содержимое для блоков title, header и content. Указывать содержимое для всех блоков базового шаблона необязательно.

Изменим код функции index() в представлении view:

```
def index(request):
    """
    :param request:
    :return:
    """

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

Результат на странице сайта:

Главная страница

Связка шаблонов index.html и base.html

Подвал страницы

Для формирования другой страницы сайта (about.html) используем тот же базовый шаблон:

```
{% extends "firstapp/base.html" %}
{% block title %}about{% endblock title %}
{% block header %} О компании {% endblock header %}
 {% block content %}
<p> Новые технологии </p>
<p> Связка шаблонов about.html и base.html </p>
{% endblock content %}
```

Изменим код функции about() в представлении view:

```python
def about(request):
    """
    :param request:
    :return:
    """
    return render(request, "firstapp/about.html")
```

Результат на странице сайта:

О компании

Новые технологии

Связка шаблонов about.html и base.html

Подвал страницы

Так, если потребуется изменить структуру всех веб-страниц сайта, добавить новые элементы или убрать старые, то достаточно будет изменить один базовый шаблон. На его основе можно будет создавать разные страницы сайта одинаковой формы, но с разной информацией.

Специальные теги в шаблонах HTML – страниц

Django предоставляет возможность использовать в шаблонах ряд специальных тегов, которые упрощают вывод некоторых данных.

Даты

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

Создадим файл-шаблон special.html:

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Специальные теги</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. Подробнее: Встроенные теги и фильтры шаблонов | | документации Django Джанго (djangoproject.com)

В файл views.py добавим функцию special:

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

В файле urls.py в переменной **urlpatterns** пропишем путь:

```
path('special/', views.special),
```

Перейдем на сайт. Страница special Специальные теги отобразит

2021

November 5 2021

Nov., 5, 2021

Nov. 5, 2021, 4:56 a.m.

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)
```

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

```
{% if n > 0 %}

    <p>Число положительное</p>

{% endif %}
```

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:

```
{% if n > 0 %}

    <p>Число положительное</p>

{% else %}

    <p>Число отрицательное или равно нулю</p>
```

{% endif %}

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

{% if n > 0 %}

Число положительное

{% elif n < 0 %}

Число отрицательное

{% else %}

Число равно нулю

{% endif %}

Еще пример:

Изменим файл special.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Специальные теги</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>
    {% if city == "Москва" %}
        <p>Москва - столица нашей Родины</p>
    {% endif %}

</body>
</html>
```

И функцию special в файле views.py

```
def special(request):
    """
    :param request:
    :return:
    """
    data = {"city": "Москва"}
    return render(request, "firstapp/special.html", context=data)
```

Страница отобразит:

2021

November 5 2021

Nov., 5, 2021

Nov. 5, 2021, 5:24 a.m.

Москва - столица нашей Родины

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:

```
def special(request):
    """
    :param request:
    :return:
    """

    cities = ["Москва", "Санкт-Петербург", "Киев", "Минск", "Новосибирск"]
    return render(request, "firstapp/special.html", context={"cities":
cities})
```

Выведем элементы массива cities в шаблоне с помощью тега for:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Специальные теги</title>
</head>
<body>
     <ul>
        {% for city in cities %}
        <li>{{ city }}</li>
        {% endfor %}
    </ul>


</body>
</html>
```

Тег **<ul>** устанавливает маркированный список. Каждый элемент списка должен начинаться с тега **<li>**.

Результат:

- Москва
- Санкт-Петербург
- Киев
- Минск
- Новосибирск

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

{% for city in cities %}

{{ city }}

{% empty %}

cities array is empty

{% endfor %}

Определение переменных

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

{% with name="Иван" age=17 %}

Name: {{ name }}

Age: {{ age }}

{% endwith %}

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Специальные теги</title>
</head>
<body>
     <ul>
        {% for city in cities %}
        <li>{{ city }}</li>
        {% endfor %}
    </ul>
        {% with name="Иван" age=17 %}
    <div>
        <p>Name: {{ name }}</p>
        <p>Age: {{ age }}</p>
    </div>
        {% endwith %}


</body>
</html>
```

Результат на странице сайта:

- Москва
- Санкт-Петербург
- Киев
- Минск

- Новосибирск

Name: Иван

Age: 17