Django Templates Basics

SoftUni Team

Technical Trainers

Software University

Software University

sli.do

#python-web

Table of Contents

1. **Templates** in Django

    ■ Django Template Language (DTL)

2. Built-in **Filters**

3. Built-in **Tags**

4. **Static Files**

3

Templates and DTL

Generate HTML Dynamically

Django Template

- A **text document** (typically an HTML file)
  - where you use the **Django Template Language** to generate content **dynamically**
- Django provides a standard API for **loading** and **rendering** templates:
  - **Loading** involves finding the template file and **performing** any necessary **preprocessing**
  - **Rendering** involves **interpolating** the template with **context** data (variables) and **returning** the resulting string

Django Template Language

- Django features its own **template system**

- Primarily used for expressing **presentation logic**

- This **template system** employs four key **elements**

  - **Variables**

  - **Filters**

  - **Tags**

  - **Comments**

DTL Variables

- Outputs a **value** from the view **context** (dictionary-like object)

- Variables are enclosed within **double curly braces** **{{**variable_name**}}**

- There are certain **rules** for naming **variables**:

  - Names **cannot** contain **spaces** or **punctuation** characters

  - A **dot** may **not** be **included** in the name

  - They **cannot** be solely comprised of a **number**

  - Variable attributes that **start** with an **underscore** (_) are **inaccessible**

Example: DTL Variables

- Displaying information for an employee

```
Employee names: {{ first_name }} {{ last_name }}
Department: {{ department }}
Email Adress: {{ email_address }}
```

- Example context

```
context = {
    "first_name": "John",
    "last_name": "Smith",
    "department": "Marketing",
    "email_address": "john.smith@company.com"}
```

Specific DTL Syntax

- Access **dictionary values** by using the **dot** **notation**

```
{{ some_dictionary.some_key }}
```

- **Call** functions by using them like **regular** variables

  - **No** parenthesis

```
{{ some_function }}
```

```
{{ some_dictionary.items }}
```

Built-in Filters

Filters

- Used to **modify variables before** they are **displayed**

- To **apply** a **filter** to a **variable**

  - use the **pipe symbol** (**|**) followed by the **filter's name**

  - **{{** variable_name**|**filter_name **}}**

- Filters can also be **chained**

  - The **output** of one filter is **passed** as **input** to the next

Filters

- Certain **filters** may require **arguments**
  - Use a **colon** (**:**) followed by the **argument values**
  - **{{** variable_name**|**filter_name**:**argument **}}**
- Django provides a **comprehensive set** of about **sixty built-in** template **filters**
  - to cater to various display **modification needs**

Commonly Used Filters

- Display the **first N chars** of a string (string ends with "…")

```
{{ value|truncatechars:N }}
```

- Display the **first N words** of a string (string ends with "…")

```
{{ value|truncatewords:N }}
```

- **Join** list elements

```
{{ list|join:", " }}
```

- **Format a date** according to the given format

```
{{ my_date|date:"D d M Y" }}
```

Click here for more date format strings

Commonly Used Filters

- If a variable is **false** or **empty**, use the given **default value**

  - Otherwise, use the value of the variable

```
{{ value|default:"nothing" }}
```

- Returns the **length** of the value (string or list)

```
{{ value|length }}
```

- Formats a float value to the **N^{th} decimal place**

```
{{ value|floatformat:N }}
```

More built-in filters: *https://docs.djangoproject.com/en/5.0/ref/templates/builtins/#built-in-filter-reference*

14

Built-in Tags

Tags

- A **template tag** in Django is a **function** that
  - yields a **value** for **display**
- Can also incorporate **custom logic**
  - into the **rendering** process
- Tags are **enclosed** within **{%** and **%}** delimiters

Tags

- Most **tags** can accept **arguments**

  - to further **customize** their **behavior**

- Some **tags require**

  - both **opening** and **closing tags**

  - to function **properly**

if Tag

- If-tag **evaluates** a variable

  - checks if it is "**true**" (exists, not empty, or not false)

- It requires **opening** and **closing** tags

```
{% if employees_list %}
    Number of employees: {{ employees_list|length }}
{% elif selected_candidates %}
    Number of candidates {{ selected_candidates|length }}
{% else %}
    No employees or candidates!
{% endif %}
```

if Tag

- When constructing conditions, use **Boolean** operators such as "**and**", "**or**" or "**not**"

  - Use of **both** **"and"** and **"or"** clauses within the **same tag** is allowed

  - Note that **"and"** takes **precedence over "or"** when used together

  - Use of **parentheses** in the if-tag is an **invalid** syntax

```
{% if employees_list and deparments or selected_candidates %}
    ...
{% endif %}
```

if Tag

- Use operators like:

  - "**==**", "**!=**", "**<**", "**>**", "**<=**", "**>=**"

  - "**in**", "**not in**", "**is**", and "**is not**"

- You can apply **filters** in the if-expressions

```
{% if employees_list|length > 10 %}
    ...
{% endif %}
```

More about Boolean operators: *https://docs.djangoproject.com/en/5.0/ref/templates/builtins/#boolean-operators*

for Tag

- Requires **both opening** and **closing** tags

- **Optionally**, it can include an **{% empty %}** tag

```
{% for employee in employees %}
    <li>{{ employee.first_name }}</li>
{% empty %}
    <li>No employees in this list.</li>
{% endfor %}
```

url Tag

- Returns a **URL** that **matches** a specified view **along** with **optional** parameters

```
{% url 'show-department-by-id' department.id %}
```

- This **URL** can then be used as a **variable** in your template

```
{% url 'some-view' as var_name %}
{% if var_name %}
  # use the URL
{% endif %}
```

csrf_token Tag

- Provides **Cross-Site Request Forgery** (**CSRF**) protection

- It should be placed **inside** the **<form>** element in your HTML code

- Cross-site request forgeries (**CSRF**) refer to:

  - a type of **malicious exploit**

  - where **unauthorized** commands are **executed** on behalf of **authenticated** users **without** their consent or knowledge
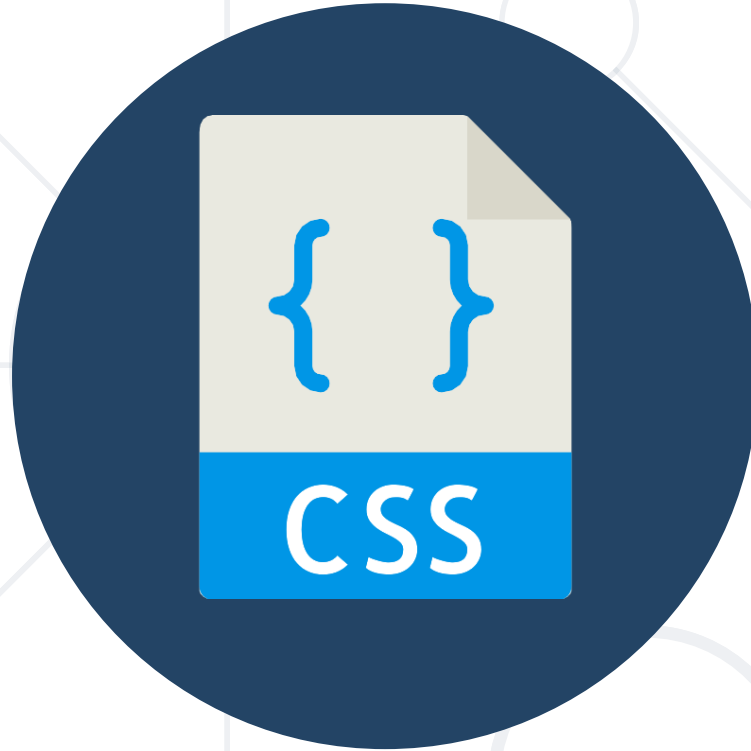
- More about **Cross-Site Request Forgery**

Comments

- Comments are surrounded by **{#** and **#}**

- A **multi-line comment** can be written using a **{% comment %}** tag

```
{# This is a comment #}

{% comment %}
This is a
multi-line
comment
{% endcomment %}
```
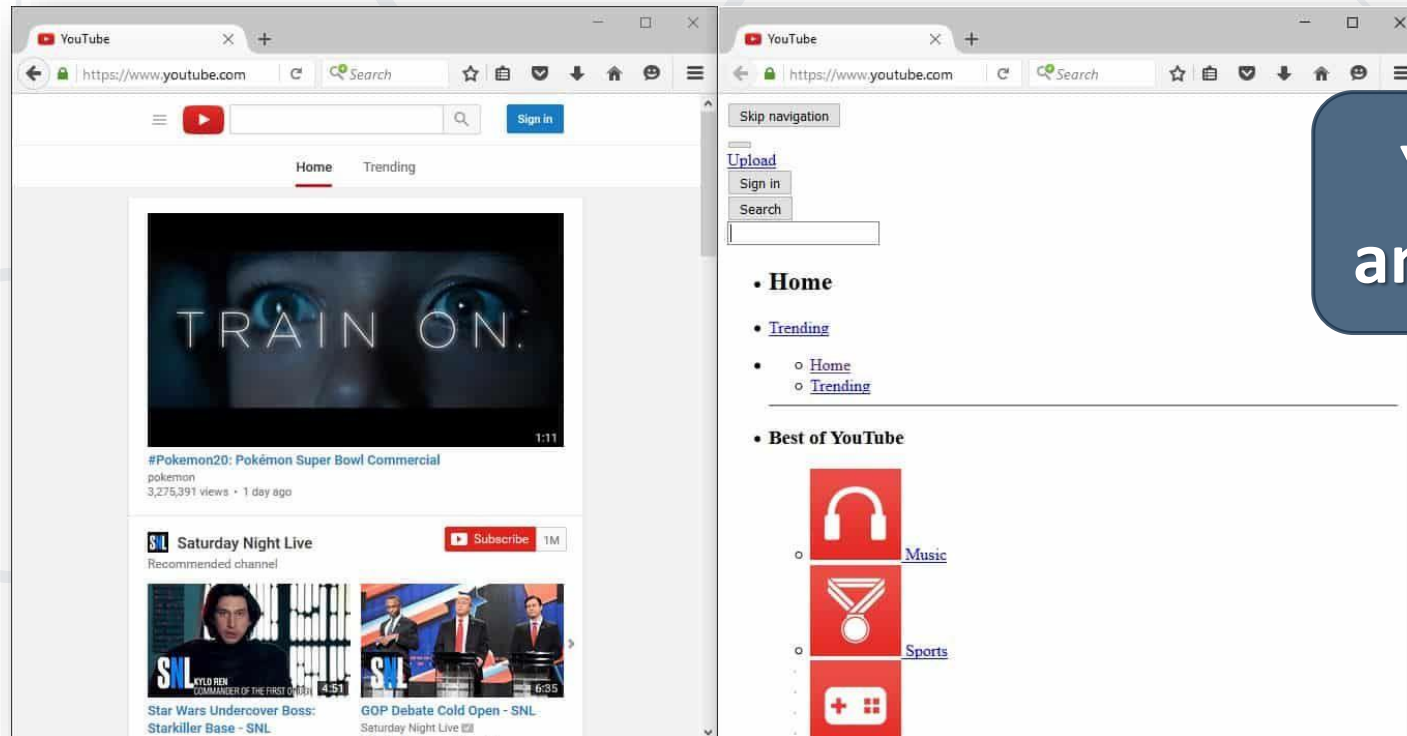
Static Files in Django

Handling Static Files

A Word About Static Files

- Your application will need to **serve** **external files** like **JavaScript**, **CSS**, and others

- These files are commonly referred to as **"static files"**

Youtube with and without CSS

Set Up a Static File Handling in Django

- Make sure your **application** is included in the **INSTALLED_APPS** in your project's **settings**

- Ensure you have **defined** the **STATIC_URL** variable in **settings**

```
STATIC_URL = '/static/'
```

- Additionally, make sure you have **specified** the **STATICFILES_DIRS** variable to indicate the **directories** where your static files are **located**

```
STATICFILES_DIRS = [BASE_DIR / 'static',]
```

Using Static Files

- To **include static files** in a Django template

  - you'll need to **load** them

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

- To include a **CSS file** in your HTML template

  - set the stylesheet link's **href attribute** to the **URL** of the **CSS file**

  ```
  {% load static %}
  <link rel="stylesheet" href="{% static 'css/style.css' %}"/>
  ```

  Path to the CSS file

Using Static Files

- To **include** an **image** in your HTML template

  - you'll need to **set** the **src attribute** of the **`<img>` tag**

  - to the **URL** of the **image file**

```
{% load static %}
<img src="{% static 'my_image.png' %}" alt="My image">
```
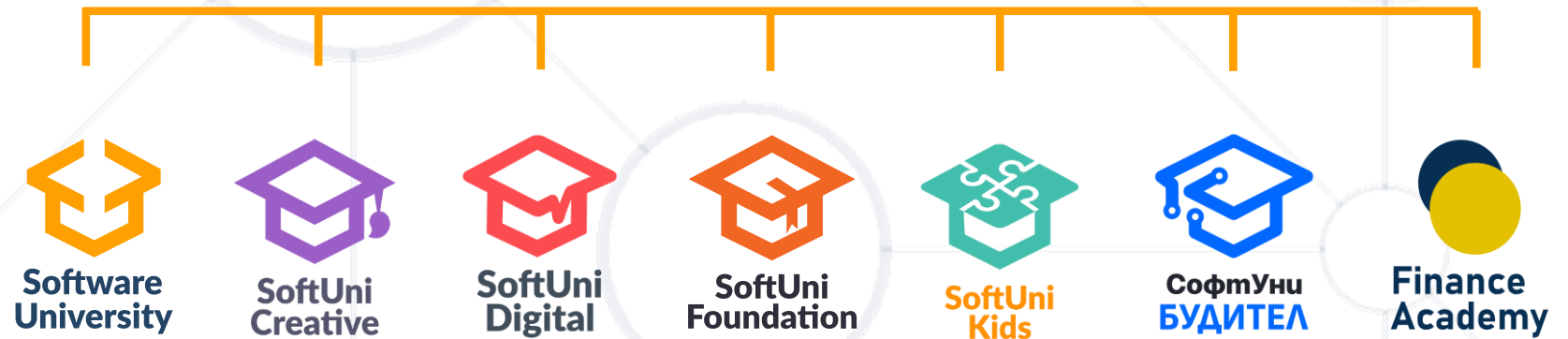
Path to the image

Summary

- **Templates** generate HTML **dynamically**

- **Filters** modify variables **before** displaying

- Template **tags** are functions

- **Static** files

  - CSS, JavaScript, images

Questions?

SoftUni Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers

  - softuni.bg, softuni.org

- Software University Foundation

  - softuni.foundation

- Software University @ Facebook

  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos, and other assets) is **copyrighted content**

- Unauthorized copy, reproduction, or use is illegal

- © SoftUni – https://softuni.org

- © Software University – https://softuni.bg