

Django Templates



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

Table of Contents

1. Templates and Django Template Language
2. Custom Filters
3. Custom Tags
4. Template Inheritance
5. Static Files



sli.do

#python-web



Templates and DTL

Generate HTML Dynamically

Django Template



- It is a **text document** (usually a .html file) in which you use the Django template language
- Django defines a standard API for **loading** and **rendering** templates:
 - Loading consists of **finding** the template and **preprocessing** it
 - Rendering means **interpolating** the template with **context data** and **returning the resulting string**

Django Template Language

- It is Django's **own template system**
- Use to **express presentation**, not program logic
- The syntax involves **four** constructs
 - Variables
 - Filters
 - Tags
 - Comments



DTL Variables

- Outputs a **value** from the view **context** (dict-like object)
- Variables are surrounded by **{{** and **}}**
- The name of a variable:
 - Cannot have **spaces** or **punctuation** characters
 - May not **have a dot in it**
 - May not be a **number**
- Variable attributes that begin with an **underscore** may **not be accessed**



Example: DTL Variables

- Displaying an 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"}
```


- Access dictionary values by using the dot notation

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

- Call functions by using them like regular variables

- Without the parenthesis

```
{{ some_function }}
```

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

Filters

- **Modifies variables** for display
- Use a pipe " | " to apply a filter to a variable
- Filters can be "**chained**"
 - The output of one filter is applied to the next
- Some filters **take arguments**
 - Use colon " : " to mark arguments
- Django provides about **sixty built-in template filters**



Commonly Used Filters (1)

- 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:"Y-m-d" }}
```

Click here for
more date
format strings

Commonly Used Filters (2)

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

- 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 **Nth decimal place**

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

Tags

- A template tag is a **function** which **returns a value** to be displayed
- Provide **custom logic** in the rendering process
- Surrounded by **{% and %}**
- Most tags accept **arguments**
- Some tags require **beginning** and **ending tags**



- **Evaluates a variable**, and if that variable is "**true**" (exists, not empty or not false)
- Requires **beginning** and **ending** 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 %}
```

- In it could be used "**and**", "**or**" or "**not**"
 - Use of **both** "**and**" and "**or**" clauses within the same tag is allowed, with "**and**" **having higher precedence** than "**or**"
 - Use of **parentheses** in the if-tag is **invalid** syntax

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

- Use operators
 - `"=="`, `"!="`, `"<"`, `">"`, `"<="`, `">="`
 - `"in"`, `"not in"`, `"is"`, and `"is not"`
- You can use **filters** in the if-expression

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


- Requires **beginning** and **ending** tags
- It can take an **optional** `{% empty %}` clause whose text is displayed if the given array is empty or could not be found

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

- Returns a URL, matching a given view and optional parameters

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

- You can use the URL as variable

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

- Cross-site Request Forgery protection
- Used **inside** the **<form>** element
- Cross-site request forgeries:
 - Type of **malicious exploit**
 - **Unauthorized commands** are performed on behalf of an **authenticated users**
- More about **Cross-site Request Forgery**

- 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
```

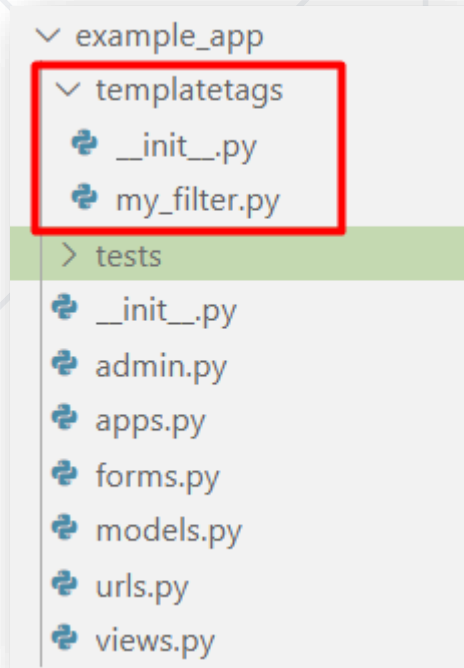
```
{% endcomment %}
```



Custom Filters

Create templatetags Folder

- In your application create a **templatetags** module with your **custom filter file**
- Write your own **filter function**



```
example_app > templatetags > my_filter.py > ...  
1  from django import template  
2  
3  register = template.Library()  
4  
5  @register.filter(name="odd")  
6  def odd(nums):  
7      """ My custom filter """  
8      return [x for x in nums if x % 2 == 1]
```

Use Your Custom Filter

- Load the filter in your template and use it

```
templates > <> home_page.html > ...  
1 {% extends 'index.html' %}  
2 {% load my_filter %}  
3  
4 {% block content %}  
5  
6     {% for num in nums|odd %}  
7         <p>{{ num }}</p>  
8     {% endfor %}  
9  
10 {% endblock %}
```

Use the name of
your file



`{% my_tag %}`

Custom Tags

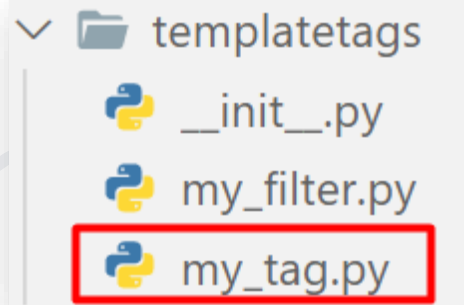
Template Tags Helper Functions

- Django provides us with **helper functions** that allow us to create custom template tags
 - **simple_tag**
 - Processes the data and **returns a string**
 - **inclusion_tag**
 - Processes the data and **returns a rendered template**



Creating Custom Template Tags

- To create a custom template tag, we need again the created **templatetags** package
- Here is an example of **an inclusion custom tag**



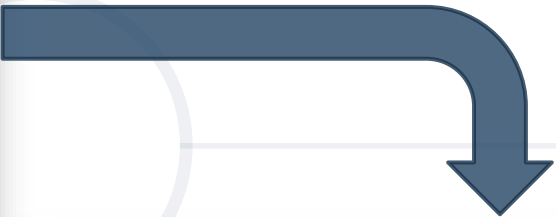
```
1 from django import template
2 from example_app.models import Article
3
4 register = template.Library()
5
6 @register.inclusion_tag('articles.html')
7 def show_articles():
8     articles = Article.objects.all()
9     return { 'articles': articles }
```

Using the Template Tag

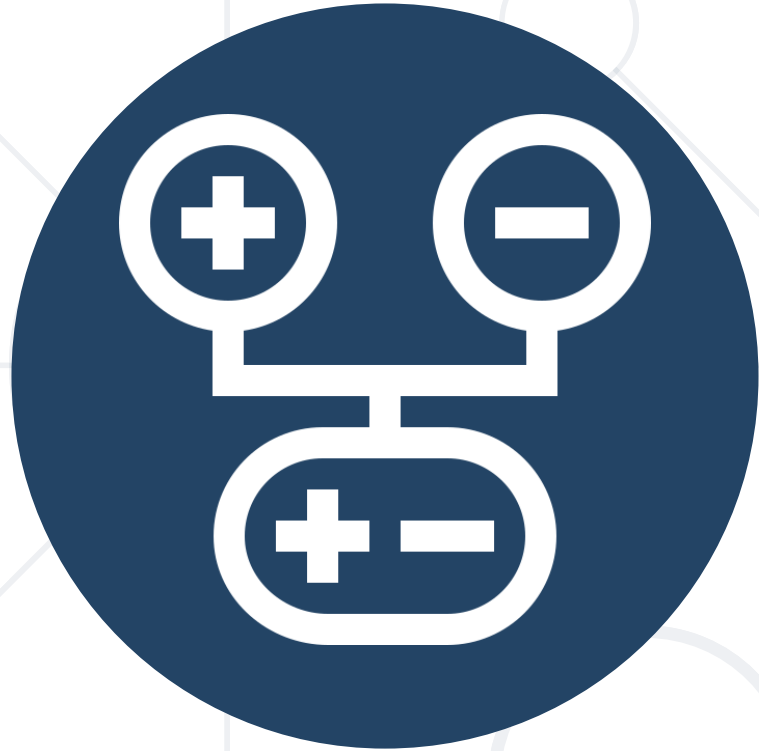
- Create the **article.html** template and make a **loop** through the articles
- After that, use your tag in your **main template**



```
articles.html X
templates > articles.html
1  <ul>
2  {% for article in articles %}
3  |   <li>{{ article.title }}</li>
4  {% endfor %}
5  </ul>
```



```
index.html X
templates > index.html
1  <h1>Home Page</h1>
2  {% show_articles articles %}
```



Template Inheritance

block, endblock, extends

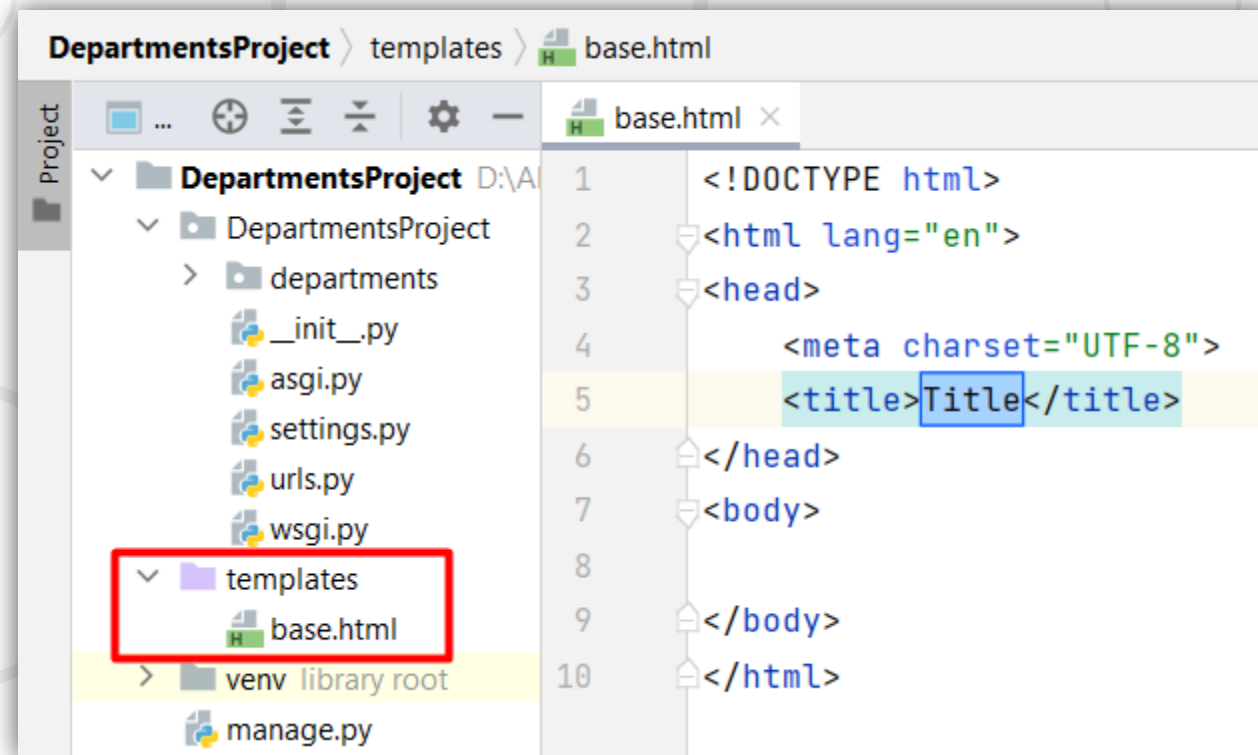
Template Inheritance

- Template inheritance allows us to build a **base** skeleton template
- The base template contains all the **common** elements and defines **blocks** that **child** templates can **override**
- Typically, the header and the footer remain the same in the whole app
- Using template inheritance, we can **reuse** the common parts of our app



Example: Template Inheritance (1)

- Create the base template in the **project/template** directory
 - The base template is typically called **base.html**



Example: Template Inheritance (2)

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>{% block title%}{% endblock %}</title>
</head>

<body>
  {% block content%}
    # default code if no content is passed in
  {% endblock %}
</body>
</html>
```

You can have many blocks in one template

Each block should have its own unique name

Example: Template Inheritance (3)

index.html

```
{% extends "base.html" %}

{% block page_title %}
    Home Page
{% endblock %}

{% block content %}
    {% for dep in departments.items %}
        <h2>{{ dep.title }}</h2>
        <p>{{ dep.description }}</p>
    {% endfor %}
{% endblock %}
```

Always use the extends tag in the "child" templates

"Home Page" will be injected in the page_title block in the base.html

The code will be injected into the content block in the base.html

Including Template Snippets

- It is a way of **including templates** within another template

```
{% include template-name %}
```

- The template name can be:

- A variable

```
{% include template_name %}
```

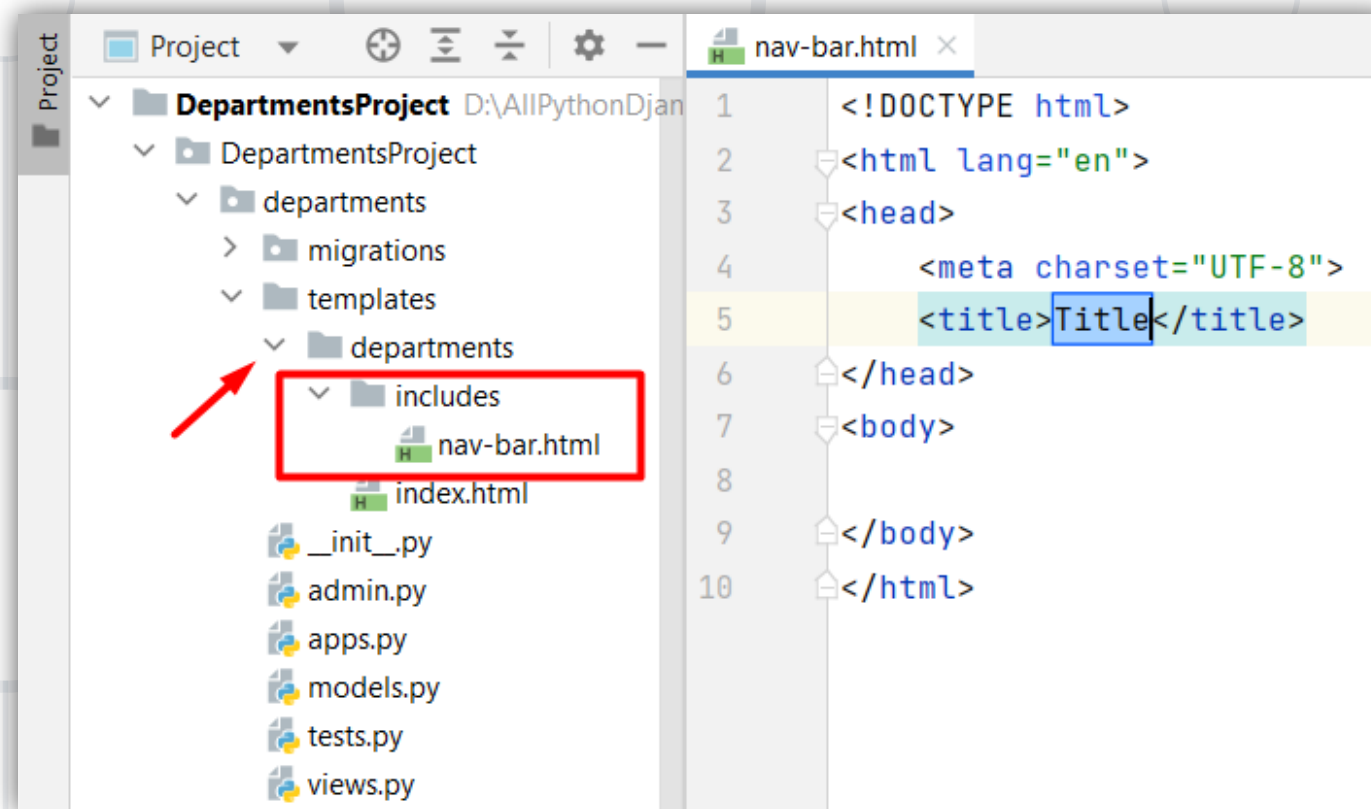
- A quoted string in single or double quotes

```
{% include "nav-bar.html" %}
```



Example: Including Template Snippets (1)

- Create a new directory in the **app/templates/departments**
 - Create a **.html** file called **nav-bar**



Example: Including Template Snippets (2)

- Create the navigation bar logic, which can be **reused** in all templates **where it is needed**

```
nav-bar.html

<header>
  <nav>
    {# some navigation bar logic #}
  </nav>
</header>
```

- If the code is used in all templates, it is better to put it in the base template

Example: Including Template Snippets (2)

- Include the template in the same HTML file, created above


index.html

```
{% extends "base.html" %}

{% block page_title %}Home Page{% endblock %}

{% block content %}
    {% include "departments/includes/nav-bar.html" %}
    {% for dep in departments.items %}
        <h2>{{ dep.title }}</h2>
        <p>{{ dep.description }}</p>
    {% endfor %}
{% endblock %}
```

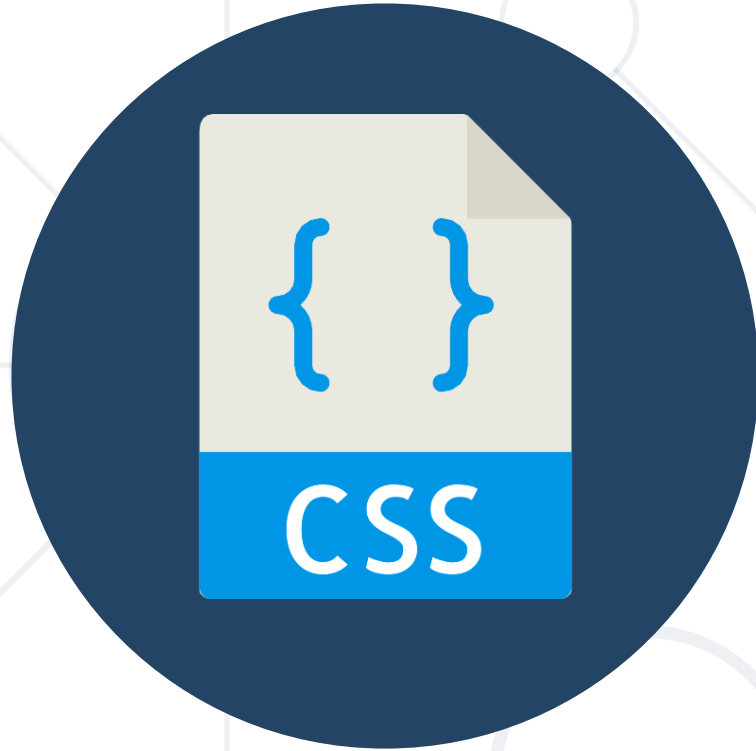
Include with Context

- 
- The included template `nav-bar.html` is **rendered within the context** of the template `index.html` that includes it
 - Also, an **additional context** can be passed to the template using keyword arguments

```
{% include template_name with user='user_name' %}
```

- Or, the context can be **rendered only** with the **variables provided**, or even **no variables** at all

```
{% include template_name with user='user_name' only %}
```

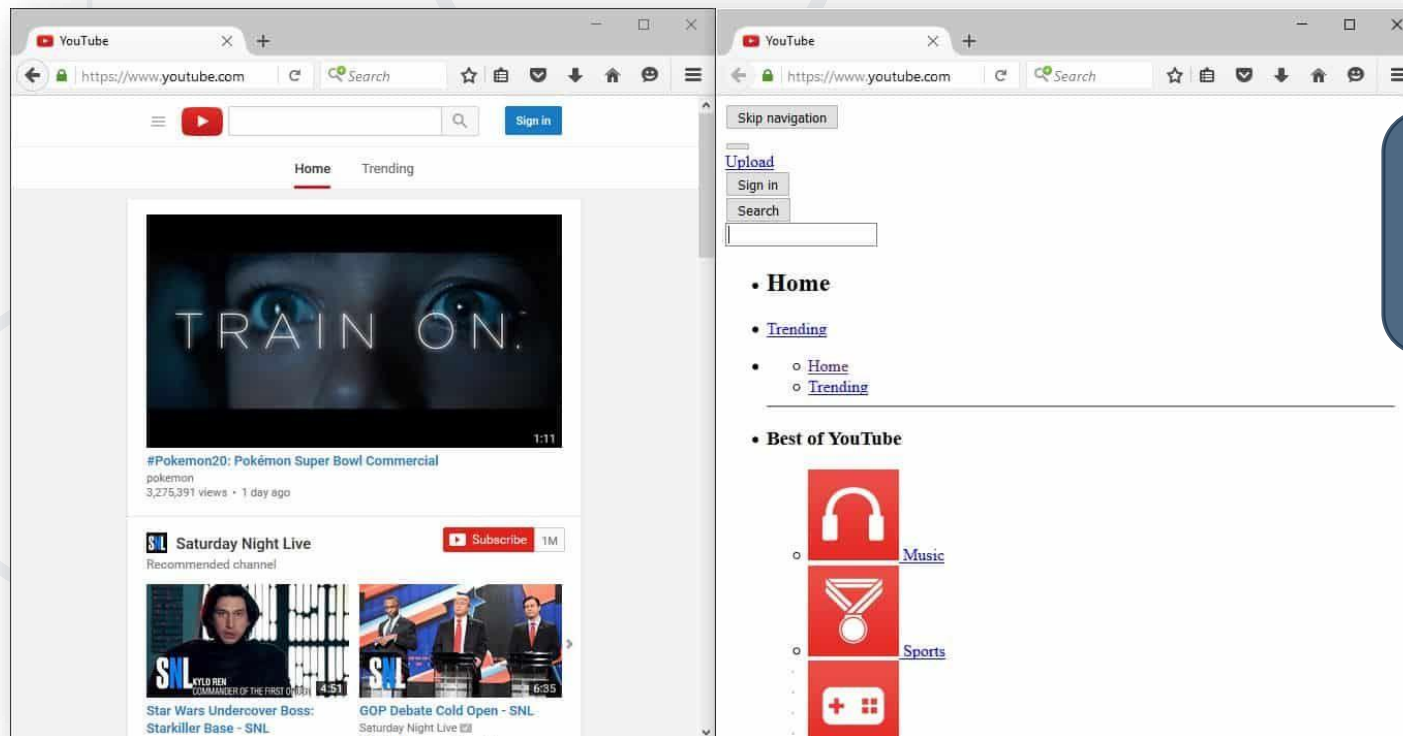


Static Files in Django

Managing Static Files

A Word About Static Files

- Most of the times, your application would need to **serve external files** such as **JavaScript**, **CSS**, etc.
- This type of files are called "**static files**"



Youtube with
and without CSS

Configuring Static Files in Django

- First, make sure that your **application** is in **INSTALLED_APPS**
- Make sure you have **STATIC_URL** variable

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

- Make sure you have **STATICFILES_DIRS** variable

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


- To include static files in the template, you will need to load it
 - To include the CSS file, set the style sheet **href** attribute

Path to the CSS file

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

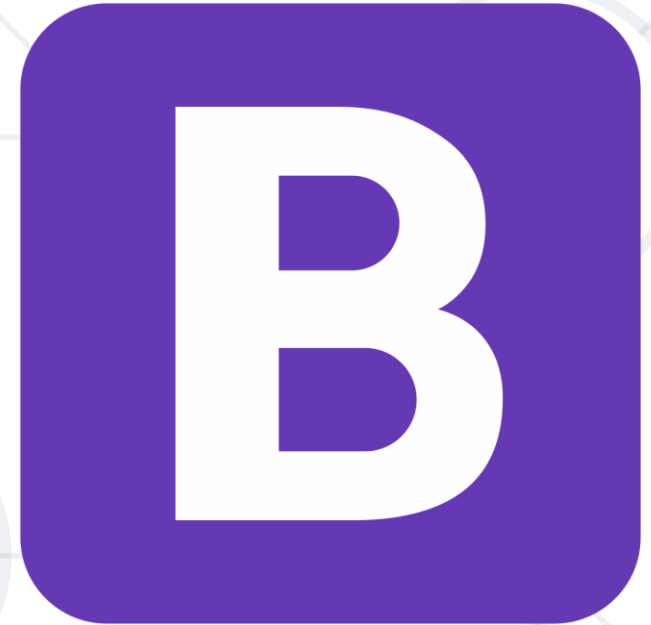
- To include an image, set the **src** attribute

Path to the image

```
{% load static %}  

```

- **CSS** Framework
- Developing Responsive
- Mobile-first websitesa
- Bootstrap 4 website





Demo

Live Exercises in Class

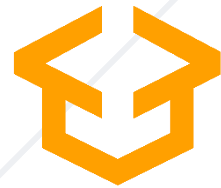
- Templates are used to generate HTML **dynamically**
- Filters allow us to **modify** variables before displaying them in the browser
- Template tags are **functions** that returns a value to be displayed
- Template inheritance allows us to build a **base** skeleton template



Questions?



SoftUni



**Software
University**



**SoftUni
Svetlina**



**SoftUni
Creative**



**SoftUni
Digital**



**SoftUni
Foundation**



**SoftUni
Kids**

SoftUni Diamond Partners



SCHWARZ



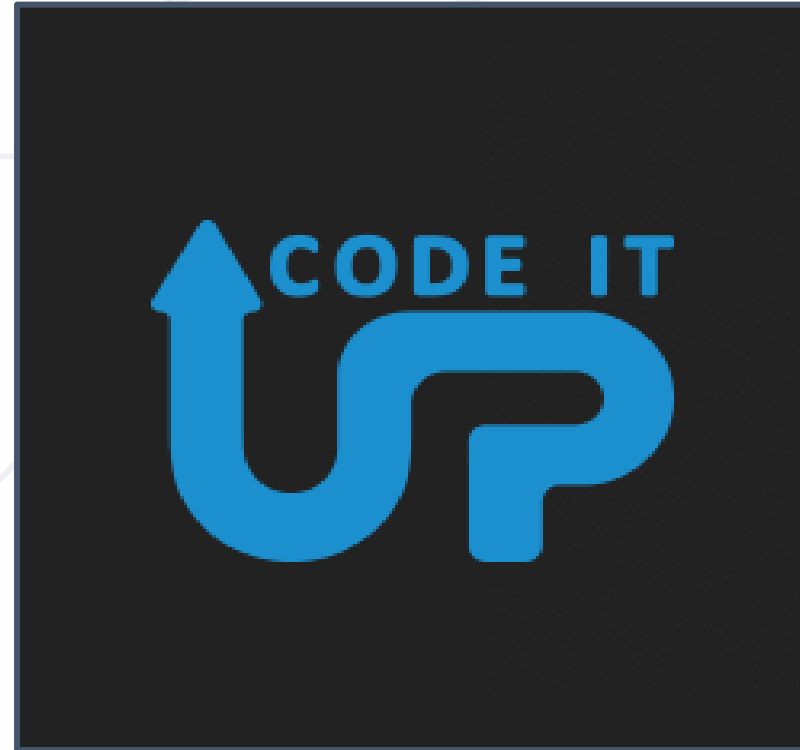
**SUPER
HOSTING
.BG**



INDEAVR
Serving the high achievers

Bosch.io





- 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

- Software University Forums

- forum.softuni.bg



Software University



- 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>

