# The Template Layer

Web Applications and Services

Spring Term

Naercio Magaia

UNIVERSITY
OF SUSSEX

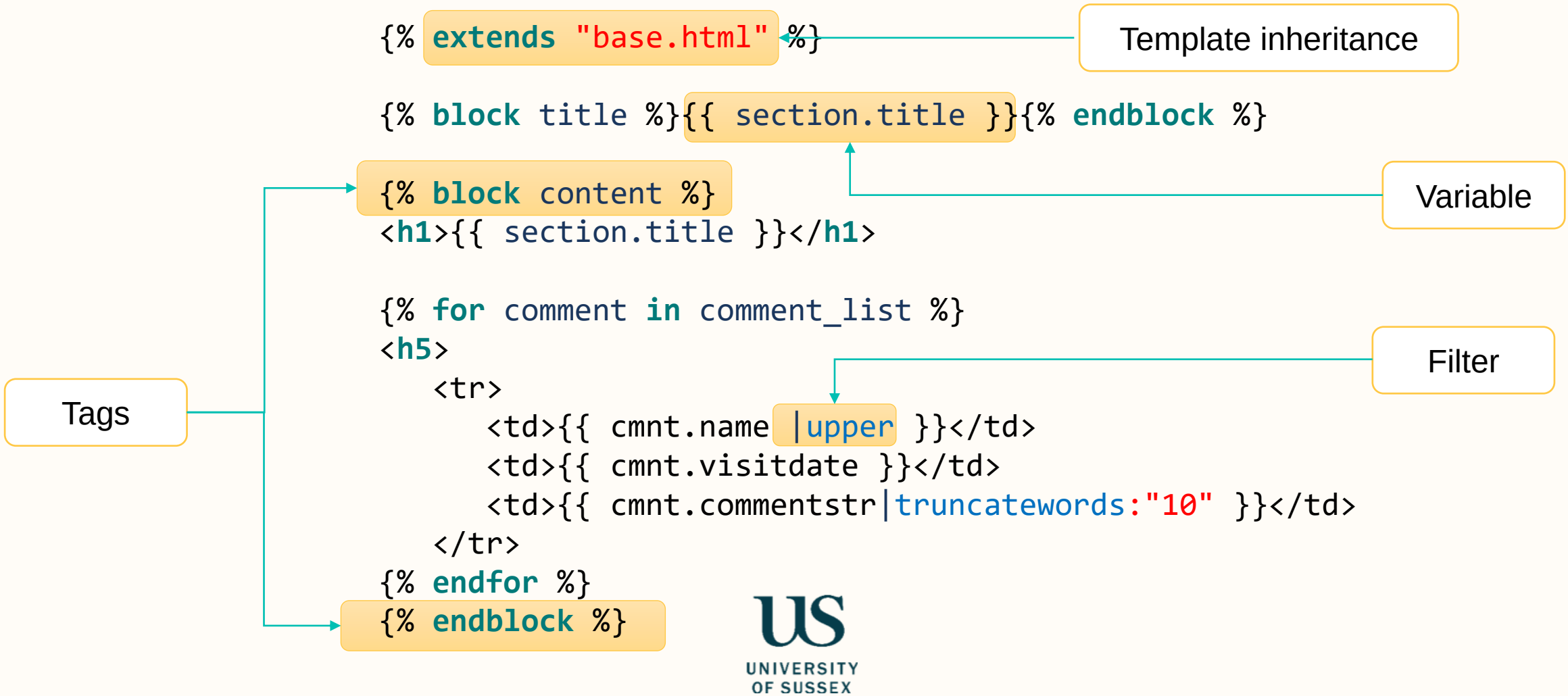
# Contents

- Templates
- Variables
- Filters
- Tags
- Comments
- Template inheritance
- Automatic HTML escaping
- Custom tag and filter libraries

# Templates

- A template is a text file, and can generate any text-based format (i.e., HTML, XML, CSV, etc.)
- It contains *variables* that are replaced with values when the template is evaluated
- It contains *tags* that control its logic

# A minimal template




UNIVERSITY OF SUSSEX

# Variables

- A variable looks like `{{ variable }}`
- The template engine evaluates and replaces it with the result
- Its name consist of any combination of alphanumeric characters and the underscore ("_")
  - It may not start with an underscore and may not be a number
- The dot (.) is used to access its attributes
  - `{{ comment.name }}` will be replaced with the name attribute of the `comment` object

UNIVERSITY OF SUSSEX

# Variables

- If an invalid (e.g., misspelled) variable is used, the template system inserts the value of the *string_if_invalid* option
  - It corresponds to setting by default to `''` (i.e., the empty string).
- If the variable "`bar`" exists in the template context, template expression like `{{ foo.bar }}` will be interpreted as a literal string and not using the value of the variable.
- Variable attributes starting with an underscore possibly will not be accessed
  - Normally, they're considered private

# Filters

- Allow modifying variables for display
- A filter looks like `{{ comment_string|length }}`
  - It is necessary to use a pipe (`|`) to apply a filter.
  - It displays the length of the `{{ comment_string }}` variable after being filtered through the length filter.
- Filters can be “chained”, that is, the output of one filter is applied to the next.
  - `{{ text|escape|linebreaks }}` is a common way for escaping text contents
- Some filters take arguments.
  - `{{ comment_string|truncatewords:10 }}`
    - It displays the first 10 words of the `comment_string` variable.

UNIVERSITY OF SUSSEX

# Tags

- A tag looks like `{% tag %}`
- They are more complex than variables, i.e., some
  - create text in the output.
  - control flow by performing loops or logic, and
  - load external information into the template to be used by later variables.
- Some tags require beginning and ending tags

```
{% tag %}
   ... tag contents ...
{% endtag %}).
```

- Django provides many built-in template tags

UNIVERSITY OF SUSSEX

# Tags

- for

```
<ul>
{% for comment in comment_list %}
    <li>{{ comment.name }}</li>
{% endfor %}
</ul>
```

- if, elif, and else

```
{% if comment_list %}
    Number of comments: {{ comment_list|length }}
{% elif comment_in_the_last_hour_list %}
    All comments made in the last hour!
{% else %}
    No comments.
{% endif %}
```

UNIVERSITY OF SUSSEX

# Comments

- Ares used to comment-out part of a line in a template
- The comment syntax is `{# #}`
- The template below will render as ‘webapps':
  - `{# greeting #}webapps`
- A comment can contain any template code, invalid or not.
  - `{# {% if web %}apps{% else %} #}`

# Template inheritance

- Is the most powerful and complex part of Django's template engine
- Allows building a base "skeleton" template that contains all the common elements of the site and defines blocks that child templates can override
- Allow using as many levels of inheritance as needed. The three-level approach is a common way of using inheritance:
  - Create a *base.html* template that holds the main look-and-feel of the site.
  - Create a *base_SECTIONNAME.html* template for each "section" of your site.
    - These templates all extend base.html and include section-specific styles/design.
  - Create individual templates for each type of page
    - These templates extend the appropriate section template.

UNIVERSITY OF SUSSEX

# Template inheritance example

- *base.html*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %}My web site{% endblock %}</title>
</head>

<body>
  <div id="content">
      {% block content %}{% endblock %}
  </div>
</body>
</html>
```

block tags

# Template inheritance example

- A child template might look like:

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

The *extends* tag tells the template engine that this template “extends” the *base.html* template.

```
{% block title %}My comment list{% endblock %}
```

```
{% block content %}
<table>
{% for cmnt in comment_list %}
    <tr>
        <td>{{ cmnt.name }}</td>
        <td>{{ cmnt.visitdate }}</td>
        <td>{{ cmnt.commentstr }}</td>
    </tr>
{% endfor %}
</table>
{% endblock %}
```

the template engine will notice and replace the two block tags in *base.html* with the contents of these blocks

# Template inheritance example

- Depending on the value of *comment_list*, the output might look like:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title My comment list </title>
</head>

<body>
  <div id="content">
    <table>
      <tr>
        <td>Naercio Magaia</td>
        <td>05/02/2023</td>
        <td>Django templates.</td>
      </tr>
    </table>
  </div>
</body>
</html>
```

UNIVERSITY OF SUSSEX

# Template inheritance tips

- The `{% extends %}` tag must be the first template tag in that template.
  - Otherwise, template inheritance won't work
- The more `{% block %}` tags in your base templates, the better as child templates don't have to define all parent blocks
- The `{{ block.super }}` variable enables getting the content of the block from the parent template
- The `{% extends %}` can be used to inherit a template at the same time as overriding it, if the same template name is used as the one inheriting from

UNIVERSITY OF SUSSEX

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

 {% block branding %}
     <img src="link/to/logo.png" alt="logo">
     {{ block.super }}
 {% endblock %}
```

# Template inheritance tips

- If a variable is created outside of a `{% block %}` using the template tag as syntax, it can't be used inside the block.

```
{% translate "Title" as title %}
{% block content %}{{ title }}{% endblock %}
```

this template doesn't render anything

- For extra readability, a *name* can be given to the `{% endblock %}` tag
- `{% block %}` tags are evaluated first.

```
{% if change_title %}
    {% block title %}Hello!{% endblock title %}
{% endif %}
```

this template will always override the content of the title block

UNIVERSITY OF SUSSEX

# Automatic HTML escaping

- Consider this template fragment

```
Hello, {{ name }}
```

- What would happen if the user entered their name as this

```
<script>alert('hello')</script>
```

- The template would be rendered as

```
Hello, <script>alert('hello')</script>
```

- the browser would pop-up a JavaScript alert box!

UNIVERSITY OF SUSSEX

# Automatic HTML escaping

- What if the name contained a '<' symbol?

```
<b>username
```

- What would happen if the user entered their name as this

```
Hello, <b>username
```

- would result in the remainder of the web page being in bold!

**User-submitted data shouldn’t be trusted blindly and inserted directly into your web pages!**

UNIVERSITY OF SUSSEX

# Automatic HTML escaping

- There are two options to avoid the Cross Site Scripting (XSS) attack described before:
  - ensure to run each untrusted variable through the escape filter, which converts potentially harmful HTML characters to unharmful ones
  - use Django's automatic HTML escaping
- In Django, templates automatically escapes the output of every variable tag by default
  - `<` is converted to `<`
  - `>` is converted to `>`
  - `'` (single quote) is converted to `&#x27;`
  - `"` (double quote) is converted to `&quot;`
  - `&` is converted to `&`

UNIVERSITY OF SUSSEX

# How to turn it off

- For individual variables
  - use the *safe* filter to disable auto-escaping for an individual variable
- Will be escaped?

1. `{{ data }}` ✓
2. `{{ data|safe }}` ✗

- For template blocks
  - wrap the template (or a particular section of the template) in the *autoescape* tag to control auto-escaping for a template

```
{% autoescape off %}
    Hello {{ name }}
{% endautoescape %}
```

- The auto-escaping tag passes its effect onto templates that extend the current one as well as templates included via the include tag

UNIVERSITY OF SUSSEX

# Custom tag and filter libraries

- To access custom tag and filter libraries provided by some applications in a template
  1. ensure the application is in INSTALLED_APPS at the *settings.py*, for example

```
'crispy_forms',
```

  2. use the *load* tag in a template

```
{% load crispy_forms_tags %}

 {{ form|crispy }}
```

- The child template is responsible for its own `{% load crispy_forms_tags %}`.

UNIVERSITY OF SUSSEX

# Next Lecture ...

- ✓ Introduction
- ✓ HTTP, Caching, and CDNs
- ✓ Views
- ✓ Templates
- ➢ **Forms**
- Models
- Security
- Transactions
- Remote Procedure Call
- Web Services
- Time
- Elections and Group Communication
- Coordination and Agreement

UNIVERSITY OF SUSSEX