جلسه ۴۸ دوره پایتون
Django Template

‘

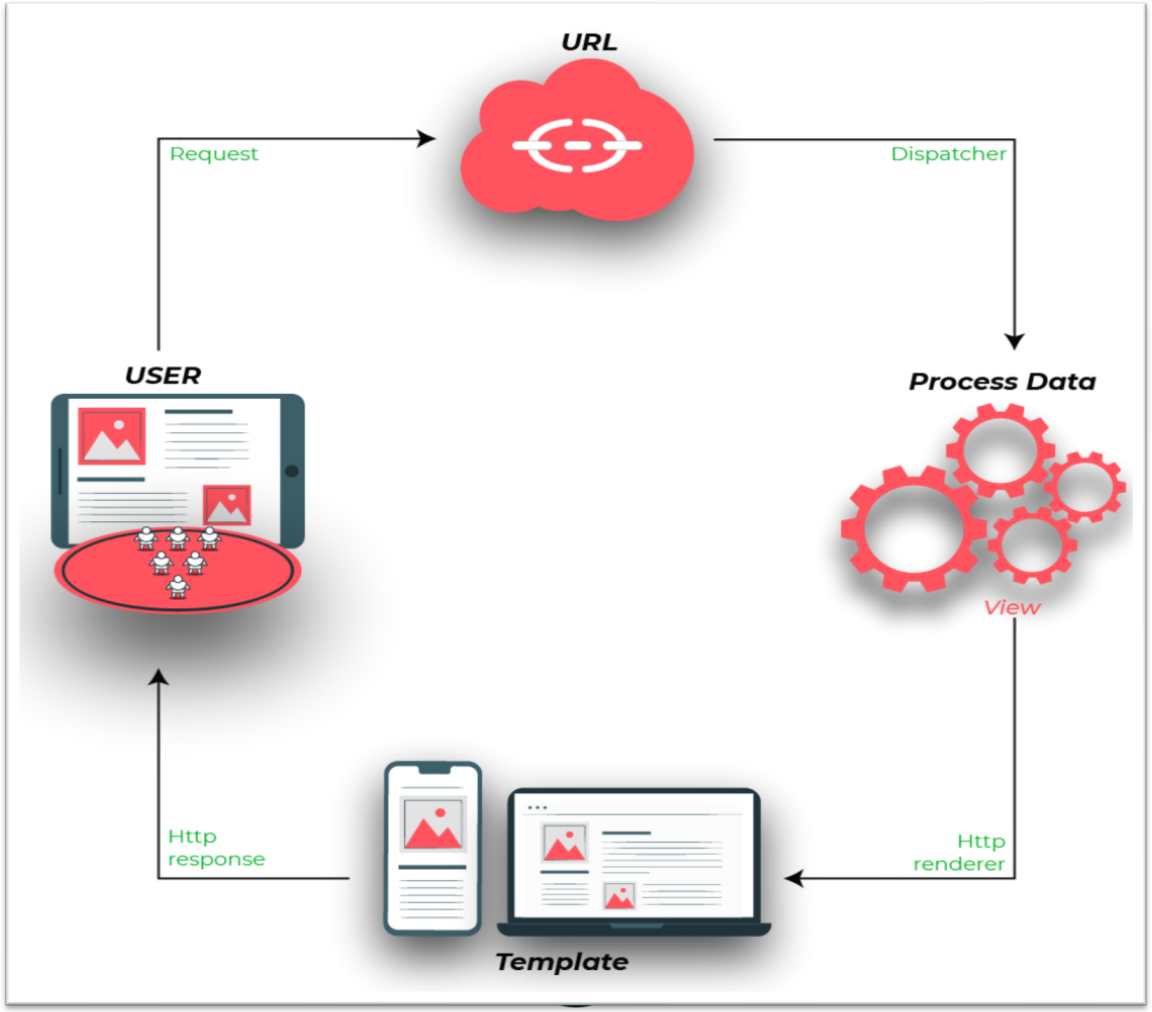
Contents

- Templates
- DTL
- Custom Tags and Filters
- Inheritance
- Exercise

Templates

- embedding the HTML code of the page into the view's code:
  - Violates the separation between UI and business logic
  - Requires Python knowledge, alongside HTML
  - Tedious and error-prone task.
- we'd better separate Django views from HTML code: Template system
  - Instead of embedding HTML code in the view, you store it in a separate file called a template.
  - This template may contain placeholders for dynamic sections that are generated in the view.
  - When generating a page, the view loads the template and passes dynamic values to it.
  - In turn, the template replaces the placeholders with these values and generates the page.

نحوه تعامل View و Template



How Django Finds Template

- Create a separate folder called *templates* in our project folder.
- Add the absolute path of your templates folder to *TEMPLATE -> DIRS* in *settings.py*.

```
TEMPLATES = [
        { 'BACKEND' : 'Django.template.backends.Django.DjangoTemplates',
        'DIRS' : [os.path.join(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, ]
                }
        }
]
```

DIR: defines a list of directories where the engine should look for template source files, in search order.

APP_DIRS: tells whether the engine should look for templates inside installed applications. Each backend defines a conventional name for the subdirectory inside applications where its templates should be stored.

Contents

- Templates
- DTL
- Custom Tags and Filters
- Inheritance
- Exercise

Django template language

- A Django template is **a text document** or **a Python string marked-up** using the **Django template language**.

- A template is **rendered** with a **context**.
- Rendering replaces variables with their values, which are looked up in the context, and executes tags.
  - Everything else is output as is.

Django template language ...

- The syntax of the Django template language involves four constructs.
  - Variables
  - Tags
  - Filters
  - Comments

Django template language- Variables

- A variable outputs a value from the context, which is a dict-like object mapping keys to values
- surrounded by {{ and }}
- passed to the template at the run time in the context
  - Simple Variable {{ title }}
  - Object Attribute {{ page.title }}
  - Dictionary Lookup {{ dict.key }}
  - List Index {{ list_items.0 }}
  - Method Call {{ var.upper }}

Django template language- Variables ...

- Example

```
My first name is {{ first_name }}. My last name is {{ last_name }}.
```

With a context of **{'first_name': 'John', 'last_name': 'Doe'}**, this template renders to:

```
My first name is John. My last name is Doe.
```


Django template language- Tags

- Tags provide arbitrary logic in the rendering process.
- This definition is deliberately vague.
- For example, a tag can
  - output content
  - serve as a control structure e.g. an “if” statement or a “for” loop
  - grab content from a database
  - enable access to other template tags
  - etc.

Django template language- Tags …

- A Template tag is surrounded by {% and %}
  - Display Logic {% if %}…{% endif %}
  - Loop Control {% for %}…{% endfor %}
  - Block Declaration {% block BLOCK_NAME %}… {% endblock %}
  - Content Import {% include “header.html” %}
  - Inheritance {% extends “base.html” %}
- It’s also possible to create custom template tags to extend the DTL.

Django template language- Filters

- Filters transform the values of variables and tag arguments for display.
- A filter is applied to a variable or tag using the (|) symbol.
- There are many built-in filters in django templates.
  - Change Case {{ name|title }} or {{ units|lower }}
  - Truncation {{ post_content|truncatewords:50 }}
  - Date Formatting {{ order_date|date:"D M Y" }}
  - List Slicing {{ list_items|slice:"3" }}
  - Default Values {{ item_total|default:"nil" }}

https://docs.djangoproject.com/en/3.1/ref/templates/builtins/#built-in-filter-reference

Django template language- Filters ...

- Example

```
You have {{ num_messages }} message{{ num_messages|pluralize }}.
```

If **num_messages** is **1**, the output will be **You have 1 message.** If **num_messages** is **2** the output will be **You have 2 messages.**

```
{{ value|random }}
```

If **value** is the list **['a', 'b', 'c', 'd']**, the output could be **"b"**.

```
{{ value|truncatewords:2 }}
```

If **value** is **"Joel is a slug"**, the output will be **"Joel is …"**.

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

If **value** is **""** (the empty string), the output will be **nothing**.

Django template language- Filters ...

- Example

If **books** is:

```
[
    {'title': '1984', 'author': {'name': 'George', 'age': 45}},
    {'title': 'Timequake', 'author': {'name': 'Kurt', 'age': 75}},
    {'title': 'Alice', 'author': {'name': 'Lewis', 'age': 33}},
]
```

```
{% for book in books|dictsort:"author.age" %}
    * {{ book.title }} ({{ book.author.name }})
{% endfor %}
```

```
* Alice (Lewis)
* 1984 (George)
* Timequake (Kurt)
```

Django template language- Comments

- Single line comment:

```
{# this won't be rendered #}
```

- Multi line comments:
  - Ignores everything between

    {% comment %} and {% endcomment %}

```
<p>Rendered text with {{ pub_date|date:"c" }}</p>
{% comment "Optional note" %}
    <p>Commented out text with {{ create_date|date:"c" }}</p>
{% endcomment %}
```