Theory: Django template language

25 minutes 0 / 5 problems solved

A typical HTML page consists of hundreds or even thousands of lines: it's a lot of repetitive work. But what if you need to change a tag or a class of several similar elements? If you're working on a site, this little problem can totally spoil the evening. Yet there is a way to avoid unnecessary copy-pasting: you can use the power of **Django Template Language (DTL)**.

§1. DTL

DTL is a language that allows you to manipulate the elements of an HTML page on the server side. **Templates** are files that consist of special syntax constructions and HTML layout. These constructions allow you to modify the page content before sending it to the client. So templates help you personalize a page directly: for each user, with love.

The work with DTL starts with four basic pillars: **variables**, **control expressions**, **tags**, and **filters**. All of that comes with the framework. You can also define your own filters and tags, but before you get to that, let's figure out what can you get from the box, starting with variables and control expressions.

> Remember, templates have tools for programming right on the page, but you should not overuse them: for that purpose, views handlers are more appropriate. If you only need the length of some expression and not the expression itself, you should calculate the length before sending it to a template. But if you need both, DTL will come in handy.

§2. Prepare a project

Suppose there is a John Doe, a guy who's working on creating some genius blog. Every blog starts with the first post and the first impression, so the content better be good; we'll leave that to John though, since our job here is code.

To launch examples, create a Django project with any name you like and add the application *blog* in it. Define a simple handler to render a template in the *blog/views.py* module:

```
from django.views import View
from django.shortcuts import render

blog_name = "John Doe's blog"
post = {
    "text": "My first post",
    "theme": "Easy talk",
    "comments": [
        "My congratulations!!",

        "Looking forward to the second one",

    ],

}




class MainView(View):

    def get(self, request, *args, **kwargs):

        context = {"post": post, "blog_name": blog_name}

        return render(request, "blog/index.html", context=context)
```

Current topic:
Django template language Stage 2

Topic depends on:
Inline Elements Stage 2
For loop 12 Stage 1
Datetime module Stage 2
Launching web server Stage 1

Topic is required for:
Using models with templates
Forms and validation
Rendering templates
Template tags Stage 3
Template filters Stage 2

Table of contents:
↑ Django template language
§1. DTL
§2. Prepare a project
§3. Variables
§4. Conditions
§5. Loops
§6. Conclusion
Feedback & Comments

Do not forget to add `"blog"` to your `INSTALLED_APPS` in the *settings.py* module, and then add this code to your *urls.py* module:

```
from blog.views import MainView

urlpatterns = [
    path("", MainView.as_view()),
]
```

You don't need to understand how this code works now: just look through the content of the `post` and `blog_name` variables.

Finally, paste this template to the *blog/templates/blog/index.html* file:

```
<html>
  <body>
    <h2>{{ blog_name }}</h2>
    <div>{{ post.text }}</div>
  </body>
</html>
```

You can keep using this file later on, just replace its content when you need!

§3. Variables

Variables are used to deliver the data to the templates. Usually, all data comes from the controller, but you can define your own variables right in the template, too.

Here is what you can pass from the Python code to the template:

- primitive types like integers and strings
- common Python structures such as dict, list, set, or tuple
- functions with no extra arguments
- an instance of a custom class

> Actually, you can pass any Python object in templates, but for any other type it makes no sense because you cannot use them properly.

To render the variable or its attributes, you should use special syntax to distinguish DTL from HTML. Put the variable in double curly brackets `{{` and `}}`:

```
<!-- Accessing a variable -->
<h2>{{ blog_name }}</h2>
```

If you want to access the attributes of the variable, use the dot operator:

```
<!-- Accessing the value of a dict by key -->
<div>{{ post.text }}</div>
```

It's possible to call the variable's method without extra arguments, the difference is that you don't need round brackets after the call. If you want to capitalize all words in the blog name, call the title method of a string variable `{{ blog_name.title }}`. Here is a nice simple example for you:

```
<h2>{{ blog_name.title }}</h2>
```

§4. Conditions

With controlling statements, we can choose what we show or do on the page depending on the conditions.

Just like in Python, branching in DTL consists of `if`, `elif`, and `else` statements. Curly brackets with the percent sign `{%` and `%}` embrace them and all the other tags: it differs tags from variables syntactically.

All branches should start with `{% if %}` and end with `{% endif %}` statements.

Let's get back to our example: we've notified John that we used his first post as an example for our topic, so he makes the next entry with the link to our site. The text of the post speaks for itself, so John does not add a theme. He wants to retain the layout of a page, and if there's no theme, the header will be *"No theme"* as well:

```
<html>
  <body>
    <h3>
      {% if post.theme %} Theme: {{ post.theme }}
      {% else %} No theme
      <!-- Without the closing tag the whole expression is not correct -->
      {% endif %}
    </h3>

<a href="https://hyperskill.org">How to make a post entry with Django</a>

  </body>

</html>
```

If you don't have a variable at all when you get access to a value, it's not a grave mistake. The rule is, if we do not pass a variable in the context, the value of this variable is by default `None`.

> Be cautious if you want to access attributes and methods of `None`: this will cause an error.

§5. Loops

Sometimes we don't know how many items we've got in a list, but we still want to show all of them one by one on a page. Loops are helpful when you have to iterate over a lot of similar elements.

The template loops are similar to Python *for* expressions. Start one with `{% for %}` and end it with `{% endfor %}` statements.

For example, look at the comment section under John's first post: he's getting somewhere! Let's render all comments to the post one after another:

```
<html>
  <body>
    {% for comment in post.comments %}
      <div>Comment #{{ forloop.counter }}: {{ comment }}</div>
    {% endfor %}
  </body>
</html>
```

To access the index of the element, we use `{{ forloop.counter0 }}` for zero-based iteration and `{{ forloop.counter }}` for one-based .

§6. Conclusion

Let's summarize! Templates provide tools to:

- Pass variables to the HTML layout
- Control the data you render
- Iterate over multiple similar elements

Template language helps you work with the HTML layout and make your life as a developer much easier. And not a single John will get hurt when creating a blog!

Report a typo

88 users liked this theory. **11** didn't like it. **What about you?**

Start practicing

This content was created over 1 year ago and updated 12 days ago. Share your feedback below in comments to help us improve it!

Comments (5) Hints (0) Useful links (2) Show discussion