Django Template Language (DTL) Notes

1. Introduction to Django Template Language (DTL)

Django Template Language (DTL) is used to separate the presentation logic from the business logic in Django. Templates define the structure of the HTML but use special syntax to insert dynamic data.

- **Loading templates**: In views, you can load and render a template:

```
from django.shortcuts import render

def my_view(request):
    return render(request, 'my_template.html', {'key': 'value'})
```

2. Basic Syntax and Structure

- **Template Tags**: Django uses `{% %}` for template logic (tags).
- **Variables**: Variables passed from views are rendered with `{{ variable_name }}`.

Example:

```
<h1>Welcome {{ user.username }}</h1>
```

3. Rendering Variables in Templates

You can pass variables (e.g., from a view) to templates, and use them to dynamically render content:

```
<p>My name is {{ person.name }}</p>
<p>I am {{ person.age }} years old.</p>
```

4. Filters in Django Templates

Filters allow manipulation of variables before rendering. They are added with the `|` symbol after a variable.

Common Filters

- `{{ name|lower }}` – Converts a string to lowercase.
- `{{ list|length }}` – Returns the length of a list.
- `{{ price|floatformat:2 }}` – Formats floats to a specified number of decimal places.
- `{{ date|date:"Y-m-d" }}` – Formats a datetime object.

Advanced Filtering Examples

Date Filters:

```
<p>Published on: {{ post.published_at|date:"F d, Y" }}</p>
```

Float Filters:

```
<p>Price: ${{ product.price|floatformat:2 }}</p>
```

String Filters:

```
<p>{{ text|truncatechars:30 }}</p>  <!-- Truncate text to 30 characters -->
```

5. Control Structures

if Statements

Control logic using `{% if %}` tag:

```
{% if user.is_authenticated %}
  <p>Welcome back, {{ user.username }}!</p>
{% else %}
  <p>Hello, Guest!</p>
{% endif %}
```

for Loops

Loop through lists or querysets with `{% for %}`:

```
<ul>
  {% for item in items %}
    <li>{{ item.name }}</li>
  {% endfor %}
</ul>
```

Loop with `forloop` variables:

```
<p>Item number {{ forloop.counter }}: {{ item.name }}</p>
```

6. Template Inheritance

Template inheritance allows creating a base structure for templates and reusing it.

- **Base template** (`base.html`):

```
<!DOCTYPE html>
<html>
  <head>
    <title>{% block title %}My Website{% endblock %}</title>
  </head>
  <body>
    <header>{% block header %}{% endblock %}</header>
    <main>{% block content %}{% endblock %}</main>
  </body>
</html>
```

- **Child template** (`home.html`):

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

{% block title %}Homepage{% endblock %}

{% block content %}
  <h1>Welcome to the homepage!</h1>
{% endblock %}
```

7. Template Comments

Add comments inside templates with `{# #}`:

```
{# This is a comment, it won't appear in the rendered HTML #}
```

8. Using Static Files in Templates

To serve static files (e.g., images, CSS, JS), load the static template tag:

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

9. Including Other Templates

The `{% include %}` tag allows you to include another template within the current template:

```
{% include 'navbar.html' %}
```

10. Template Context and Context Processors

Context processors allow adding common data (e.g., user info) globally to all templates.

- **Using context processors**:

```
<p>Logged in as: {{ user.username }}</p>
```

11. Custom Template Filters and Tags

You can create custom filters and tags using Django's template system.

- **Custom filter** (`templatetags/custom_filters.py`):

```
from django import template

register = template.Library()

@register.filter
def add(value, arg):
    return value + arg
```

- **Using custom filters** in templates:

```
<p>{{ number|add:5 }}</p>  <!-- Adds 5 to number -->
```

12. Working with Forms in Templates

Django forms can be rendered easily in templates:

- **Rendering form fields**:

```
<form method="POST">
  {% csrf_token %}
  {{ form.as_p }}  <!-- Renders the form as paragraphs -->
  <button type="submit">Submit</button>
</form>
```

13. Using Template Built-in Tags

csrf_token

Used to prevent Cross-Site Request Forgery attacks:

```
<form method="POST">
  {% csrf_token %}
</form>
```

url

Generates a URL for a given view:

```
<a href="{% url 'post_detail' post.id %}">Read more</a>
```

with

Used to alias variables inside a block:

```
{% with total=price|floatformat:2 %}
  <p>Total: ${{ total }}</p>
{% endwith %}
```

14. Template Debugging

Enable `DEBUG = True` in your `settings.py` file for better error messages in templates. Use the **Django Debug Toolbar** to monitor template performance and queries.

15. Template Performance Optimization

Use the `{% cache %}` template tag to cache parts of your template for improved performance:

```
{% load cache %}
{% cache 500 sidebar %}
  <!-- Sidebar code goes here -->
{% endcache %}
```

Advanced Filtering Options

Date Filter

- Formats a datetime object:

```
{{ value|date:"Y-m-d" }}
```

String Filter

- **truncatechars**: Shortens the string to a specific number of characters:

```
{{ long_text|truncatechars:50 }}
```

Float Filter

- **floatformat**: Rounds the float to a given number of decimal places:

```
{{ number|floatformat:2 }}
```

Default Filter

- **default**: Provides a default value if the variable is not present:

```
{{ value|default:"N/A" }}
```

Length Filter

- **length**: Returns the length of a list or string:

```
{{ items|length }}
```

This concludes the comprehensive notes on Django Template Language, including basic and advanced concepts. You can now refer to these sections as you explore DTL step by step.