Django Templates

Last Updated : 17 Apr, 2024

Templates are the third and most important part of Django's MVT Structure. A template in Django is basically written in HTML, CSS, and Javascript in a .html file. Django framework efficiently handles and generates dynamic HTML web pages that are visible to the end-user. Django mainly functions with a backend so, in order to provide a frontend and provide a layout to our website, we use templates. There are two methods of adding the template to our website depending on our needs.
We can use a single template directory which will be spread over the entire project. For each app of our project, we can create a different template directory.

Templates in Django

For our current project, we will create a single template directory that will be spread over the entire project for simplicity. App-level templates are generally used in big projects or in case we want to provide a different layout to each component of our webpage.

Configuration

settings.py: Django Templates can be configured in app_name/settings.py,

We will create our Templates folder in "BASE_DIR/'templates' ".

Python3

```
TEMPLATES = [
    {
        # Template backend to be used, For example Jinja
        'BACKEND':
'django.template.backends.django.DjangoTemplates',

        ## Path definition of templates folder .
        'DIRS': [BASE_DIR/'templates'],
```

```
        'APP_DIRS': True,
        # options to configure
        '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',
            ],
        },
    },
]
```

Using Django Templates

Illustration of How to use templates in Django using an Example Project. Templates not only show static data but also the data from different databases connected to the application through a context dictionary. Consider a project named geeksforgeeks having an app named geeks.

> *Refer to the following articles to check how to create a project and an app in Django.*

- *How to Create a Basic Project using MVT in Django?*
- *How to Create an App in Django ?*

views.py: To render a template one needs a view and a URL mapped to that view. Let's begin by creating a view in geeks/views.py,

- `simple_view`**:** Renders the "geeks.html" template with the data **"Gfg is the best."**
- `check_age`**:** Handles a form submission, checking the user's age and rendering the **"check_age.html"** template with the age.

- `loop`: Sends a list of numbers and the string **"Gfg is the best"** to the **"loop.html"** template.

Python3

```python
from django.shortcuts import render
from .forms import AgeForm


# create a function
def simple_view(request):
    data = {"content": "Gfg is the best"}
    return render(request, "geeks.html", data)


def check_age(request):
    if request.method == 'POST':
      # Get the age from the form
        age = int(request.POST.get('age', 0))
        return render(request, 'check_age.html', {'age': age})
    return render(request, 'check_age.html')


def loop(request):
    data = "Gfg is the best"
    number_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    context = {
        "data": data,
        "list": number_list}

    return render(request, "loop.html", context)
```

urls.py: Here we are defining all the URL path for the different views.

Python3

```python
from django.urls import path

# importing views from views..py
```

```
from mini import views

urlpatterns = [
    path('simple',views.simple_view),
    path('condition', views.check_age),
    path('loop', views.loop),
]
```

geeks.html: Here we are printing the data which Is being passed by view function.

Created this on the path: app_name/templates/geeks.html .

HTML

```
<!-- app_name/templates/geeks.html  -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Homepage</title>
</head>
<body>
    <h1>Welcome to Geeksforgeeks.</h1>
<p> {{  data }}</p>



    <ul>
</body>
</html>
```

check_age.html: Here first we are accepting age from the user and then checking it with the jinja if it is greater than certain age or not.

Created this on the path: app_name/templates/check_age.html .

HTML

```html
<!-- app_name/templates/check_age.html -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Age Checker</title>
</head>
<body>
    <h1>Welcome to the Age Checker</h1>

    <form method="post">
        {% csrf_token %}
        <label for="age">Enter your age:</label>
        <input type="number" id="age" name="age">
        <button type="submit">Check Age</button>
    </form>

    {% if age %}
        <p>Your age is: {{ age }}</p>
        {% if age >= 20 %}
            <p>You are an adult.</p>
        {% else %}
            <p>You are not an adult.</p>
        {% endif %}
    {% endif %}
</body>
</html>
```

loop.html: Here we are printing the even numbers of the list with the help of loop and condition in Jinja.

Created this on the path: app_name/templates/loop.html .

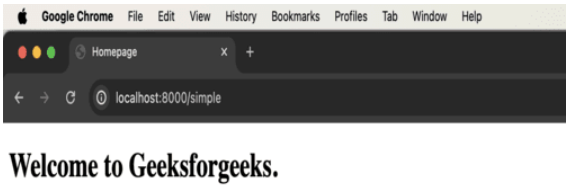
HTML

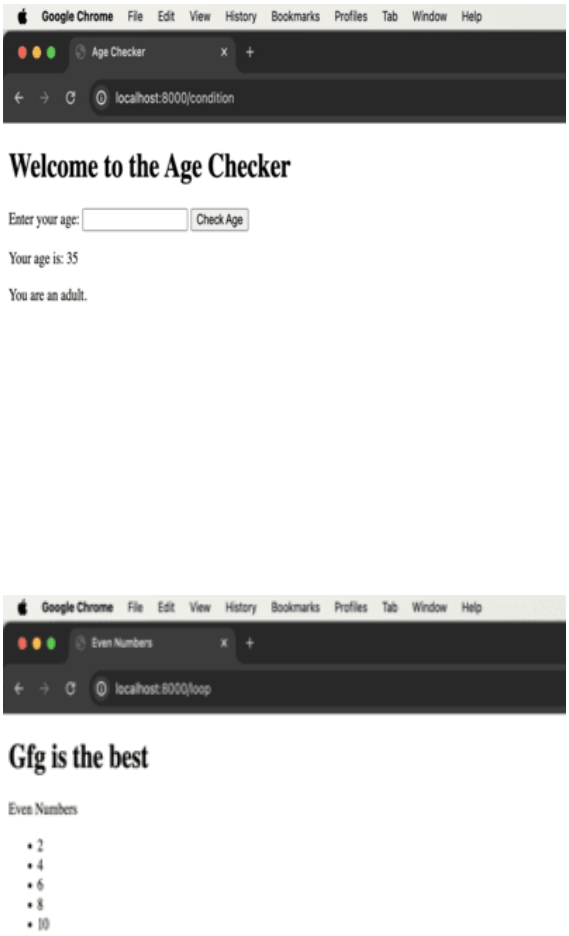
```html
<!-- app_name/templates/loop.html -->

<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Even Numbers</title>
</head>
<body>
    <h1>{{ data }}</h1>
    Even Numbers
    <ul>
        {% for number in list %}
            {% if number|divisibleby:2 %}
                <li>{{ number }}</li>
            {% endif %}
        {% endfor %}
    </ul>
</body>
</html>
```

Output:





You can also keep the path as app_name/templates/app_name/html_file.html, in this case you will have to define the path in app_name/views.py as: “app_name/html_file.html” this is the traditional and conventional way to do it. The former method is mostly used.

The Django template language

This is one of the most important facilities provided by Django Templates. A Django template is a text document or a Python string marked-up using the Django template language. Some constructs are recognized and interpreted by the template engine. The main ones are variables and tags. As we used for the loop in the above example, we used it as a tag. similarly, we can use various other conditions such as if, else, if-else, empty, etc. The main characteristics of Django Template language are Variables, Tags, Filters, and Comments.

Jinja Variables

Variables output a value from the context, which is a dict-like object mapping keys to values. The context object we sent from the view can be accessed in the template using variables of Django Template.

```
Syntax: {{ variable_name }}
```

Example: Variables are surrounded by {{ and }} like this:

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

With a context of {'first_name': 'Naveen', 'last_name': 'Arora'}, this template renders to:

```
My first name is Naveen. My last name is Arora.
```

To know more about Django Template Variables visit – variables – Django Templates

Jinja Tags

Tags provide arbitrary logic in the rendering process. 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, or even enable access to other template tags.

```
Syntax: {% tag_name %}
```

Example: Tags are surrounded by {% and %} like this:

```
{% csrf_token %}
```

Most tags accept arguments, for example :

```
{% cycle 'odd' 'even' %}
```

	Commonly used Tags	
Comment	cycle	extends
if	for loop	for ... empty loop
Boolean Operators	firstof	include
lorem	now	url

Filters

Django Template Engine provides filters that are used to transform the values of variables and tag arguments. We have already discussed major Django Template Tags. Tags can't modify the value of a variable whereas filters can be used for incrementing the value of a variable or modifying it to one's own need.

```
Syntax: {{ variable_name | filter_name }}
```

Filters can be "chained." The output of one filter is applied to the next. {{ text|escape|linebreaks }} is a common idiom for escaping text contents, then converting line breaks to <p> tags.

```
Example: {{ value | length }}
```

If value is **['a', 'b', 'c', 'd']**, the output will be **4**.

	Major Template Filters	
add	addslashes	capfirst
center	cut	date

	Major Template Filters	
default	dictsort	divisibleby
escape	filesizefodivisible byrmat	first
join	last	length
line numbers	lower	make_list
random	slice	slugify
time	timesince	title
unordered_list	upper	wordcount

Comments

Template ignores everything between {% comment %} and {% end comment %}. An optional note may be inserted in the first tag. For example, this is useful when commenting out code for documenting why the code was disabled.

```
Syntax: {% comment 'comment_name' %}
{% endcomment %}
```

Example :

```
{% comment "Optional note" %}
    Commented out text with {{ create_date|date:"c" }}
{% endcomment %}
```

To know more about using comments in Templates, visit comment – Django template tags

Template Inheritance

The most powerful and thus the most complex part of Django's template engine is template inheritance. Template inheritance allows you to build a base "skeleton" template that contains all the common elements of your site and defines blocks that child templates can override. extends tag is used for the inheritance of templates in Django. One needs to repeat the same code again and again. Using extends we can inherit templates as well as variables.

Syntax: `{% extends 'template_name.html' %}`

Example: Assume the following directory structure:

```
dir1/
    template.html
    base2.html
    my/
        base3.html
base1.html
```

In template.html, the following paths would be valid:

HTML

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./my/base3.html" %}
```

To know more about Template inheritance and extends, visit extends – Django Template Tags

Are you ready to elevate your web development skills from foundational knowledge to advanced expertise? Explore our Mastering Django Framework - Beginner to Advanced Course on GeeksforGeeks, designed for aspiring

developers and experienced programmers. This comprehensive course covers everything you need to know about Django, from the basics to advanced features. Gain practical experience through **hands-on projects** and real-world applications, mastering essential Django principles and techniques. Whether you're just starting or looking to refine your skills, this course will empower you to build sophisticated web applications efficiently. Ready to enhance your web development journey? Enroll now and unlock your potential with Django!

Next Article

variables - Django Templates

Similar Reads

Django Templates | Set - 1

There are two types of web pages - Static and Dynamic pages. Static webpages are those pages whose content is static i.e. they don't change with time. Every...

2 min read

Django Templates | Set - 2

Prerequisite: Django Templates | Set-1, Views In Django. Navigate to brand/views.py and add following code to brand/views.py from django.shortcuts...

2 min read

variables - Django Templates

A Django template is a text document or a Python string marked-up using the Django template language. Django being a powerful Batteries included...

2 min read

Exploring Lazy Loading Techniques for Django Templates

In this article, we'll explore how to implement lazy loading in Django templates using JavaScript and HTML. If you have no idea about how to create projects in...

5 min read

How to Add Data from Queryset into Templates in Django

In this article, we will read about how to add data from Queryset into Templates in Django Python. Data presentation logic is separated in Django MVT(Model...

3 min read

Access Constants in settings.py from Templates in Django

Django is a popular web framework known for its simplicity and powerful features, enabling developers to build robust web applications quickly. One of t...

4 min read

Format Numbers in Django Templates

Formatting numbers in Django templates is essential for creating a polished and user-friendly interface. Whether you use Django's built-in filters for common...

2 min read

Concatenate Strings in Django Templates

Concatenating strings in Django templates can be achieved through several methods, including using built-in filters, formatting, and custom template tags....

3 min read

What is the equivalent of None in django Templates

In Python, `None` is a special value that means "no value." When using Django templates, you might need to check if a variable is `None` to show something el...

2 min read

Handle and Iterate Nested Dictionaries in Django Templates

Django templates offer various tags and filters to work with data. they are designed to readable and expressive. To iterate through a dictionary which...

4 min read

Article Tags : Python Django-templates Python Django

Practice Tags : python