Django Web Framework

Template Filters and Template Inheritance

Template Filters-

Remember we used template variables to display data on a web page. Now if we want to filter that data or manipulate the text value then we use Template filters.

They are applied in the template using the pipe (|) character. Template filters can perform a variety of tasks, such as formatting dates, manipulating strings, converting data types, and much more.

Applied using pipes (|) after a variable name in your template. You can chain multiple filters together for complex formatting.

Let's see an example-

Practical 1: Django Template Filters

Step 1: Create Django project and application, configure application and template in settings.py. Create template folder.

```
C:\Users\LENOVO\Documents\djangocourse>django-admin startproject templatefiltersProject

C:\Users\LENOVO\Documents\djangocourse>cd templatefiltersProject

C:\Users\LENOVO\Documents\djangocourse\templatefiltersProject>python manage.py startapp templateapp
```

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'templateapp'
]
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [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',
            ],
        },
    },
]
```

```
templates\templateapp
    display.html
```

Step 2: Create view, write template code, urls and visit the website by starting the development server-

```
# Create your views here.
def display_view(request):
    name = 'harsh'
    context={
        'name':name,
    }
    return render(request,'templateapp/display.html',context)
```

templates > templateapp > display.html > html > body > h1

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Template filters</title>
</head>
<body>
    <h1>
        Name: {{name}}
    </h1>
</body>
</html>
```

Here we have only used template variable to display the name.

```python
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('templateapp.urls')),
]
```

templateapp > urls.py > ...

```python
from django.urls import path
from . import views

urlpatterns=[
    path('',views.display_view),
]
```

127.0.0.1:8000

Name: harsh

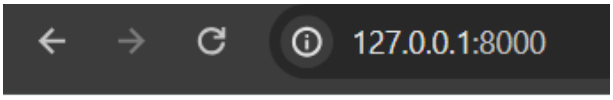
Step 3: Using our first template filter-

1.upper: Converts a string to uppercase.

Update template code-

```
<h1>
    Name: {{name | upper}}
</h1>
```

Now your name will appear in uppercase on the web browser-



Name: HARSH

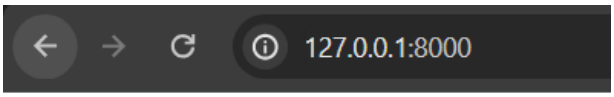
Same way we can use **lower** to display the data in lower-case.

2.Default: Provides a default value if the variable is empty or does not exist.

Update the template code-

```
<h1>
    Name: {{name | upper}} <br/>
    Country: {{country | default:'India'}}
</h1>
```

The default value will be displayed, because we didn't define any variable name 'country' in our views.



Name: HARSH
Country: India

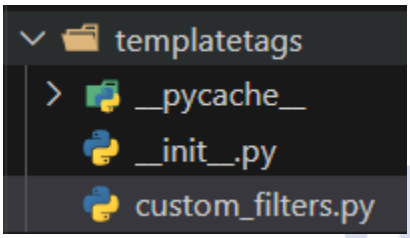
We can also create custom filters-

To create your own filter based on certain requirements we use custom filters.

How to create custom filters-

Step 1: Create a templatetags folder inside our application folder (templateapp).

Inside templatetags create **__init__.py** file, so that python treat this directory as a package. And create a file named **custom_filters.py** (you can give any name to this file).



custom_filters.py-

templateapp > templatetags > custom_filters.py > ...

```python
from django import template
register=template.Library()


def concatname(value):
    last_name = 'Trivedi'
    return f"{value} {last_name}"

register.filter('concatname',concatname)


#We can also register filter by using decorators
#@register.filter(name='concatname')
```

Explanation-

1.The file starts by importing template from the django module. This import is necessary to register the custom filter with Django's templating system.

2. A variable named register is created using template.Library(). This variable will be used to register the custom filter.

3. The file defines a function named concatname that takes a single argument value. This argument is expected to be a string.

4. Inside the function, the code defines a string variable last_name with the value "Trivedi". It then uses an f-string to format a new string by combining the value argument and the last_name variable.

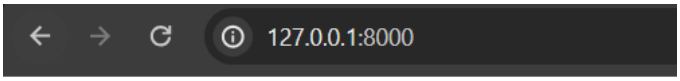
5. The register.filter decorator is used to register the concatname function as a custom filter. The decorator takes an optional argument name to specify a custom name for the filter in templates. In this case, the default name concatname is likely being used.

Step 2: Update the template and start the development server.

We use {%load custom_filters%} to load the custom filter on our template

{%load file_name%}

```
templates > templateapp > display.html > html
<!DOCTYPE html>
{%load custom_filters%}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Template filters</title>
</head>
<body>
    <h1>
        Name: {{name | upper}} <br/>
        Country: {{country | default:'India'}} <br/>
        Full name: {{name | concatname}}
    </h1>
</body>
</html>
```



Name: HARSH
Country: India
Full name: harsh Trivedi

Template Inheritance

Template Inheritance is a functionality to display same content over multiple template files without needing to write them again and again. Template Inheritance promotes reusability and reduces repetitive code.

Here, we basically write the main code in a base template file and load the base template in child templates.

Template inheritance is a powerful feature in Django that allows you to create reusable layouts and manage the structure of your web pages efficiently. It works by establishing a hierarchy of templates, where child templates inherit the basic structure from a parent template.

How to achieve Template Inheritance-

1.Base Template- In the base template, we write the code that will be reused by other templates. Now, suppose we have the same header and footer in multiple webpages. So, we write our code for header and footer in the base template. And the middle section will be written in the child template. But, here in base template we have to create block that will mark that the child template content will be displayed here.

To do this we use template tags as below-

{%block name_of_the_block%} {%endblock%}

2.Child template- These child templates inherit the structure from the base template using the {% extends %} tag. Each child template can then override specific blocks defined in the base template by placing their own content within those blocks.

To load the base template structure in the child template we use **{%extends base_template_name%}** template tag.

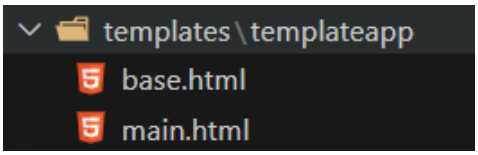
Let's see a simple example to simply understand the template inheritance-

Step 1: Create a new project and application, configure application in settings.py, create templates directory and configure it in settings.py.

```
C:\Users\LENOVO\Documents\djangocourse>django-admin startproject templateinheritanceProject

C:\Users\LENOVO\Documents\djangocourse>cd templateinheritanceProject

C:\Users\LENOVO\Documents\djangocourse\templateinheritanceProject>python manage.py startapp templateapp
```




```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'templateapp',
]
```

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR,'templates'],
```

Step 2: Create base.html,about.html and main.html-

base.html-

```
templates > templateapp > base.html > html
<!DOCTYPE html>
{%load static%}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Gym Website</title>
    <link rel="stylesheet" href="{%static 'templateapp/css/base.css'%}">
</head>
<body>
    <div class="header">
        <h1>POWER GYM</h1>
        <ul>
            <li><a href="{%url 'home'%}">Home</a></li>
            <li><a href="{%url 'about'%}">About</a></li>
        </ul>
    </div>

    {%block mainblock%} {%endblock%}

    <div class="footer">
        <h2>Power GYM</h2>
        <p>Contact: abc@gmail.com</p>
    </div>
</body>
</html>
```

In the above template code, in base.html you can see the block part-

```
{%block mainblock%} {%endblock%}
```

Now in child templates we have to write our code within this block.

main.html- Extends the base template

```
templates > templateapp > main.html > link
{%extends 'templateapp/base.html'%}
{%block mainblock%}
{%load static%}
<link rel="stylesheet" href="{%static 'templateapp/css/main.css'%}">
<div class="main">
    <h3>Power Gym is a fitness center empowering you to achieve your health and wellness goals.</h3>
    <p>Power Gym Vision and Goals: <br/>
        Promote Holistic Health and Wellness: Power Gym aims to foster an environment where physical fitness and mental well-being
        Accessibility and Inclusivity: The gym strives to be a welcoming space for individuals of all fitness levels, backgrounds,
        Innovative Fitness Solutions: Power Gym is committed to staying at the forefront of fitness technology and trends, offering
    </p>
</div>
{%endblock%}
```

about.html- Extends the base template

```
templates > templateapp > about.html > div.about > p
{%extends 'templateapp/base.html'%}
{%block mainblock%}
{%load static%}
<link rel="stylesheet" href="{%static 'templateapp/css/about.css'%}">
<div class="about">
    <p>Power Gym is a comprehensive fitness center designed to cater to a variety of workout needs and fitness goals. It offers state-of-the
</div>
{%endblock%}
```

NOTE: I included CSS for styling, if you don't want CSS you can simply avoid the static load and CSS link part.

Step 3: Define views and URLs-

```
templateapp > views.py > about_view
from django.shortcuts import render

# Create your views here.
def display_view(request):
    return render(request,'templateapp/main.html')


def about_view(request):
    return render(request,'templateapp/about.html')
```

templateapp > urls.py > ...

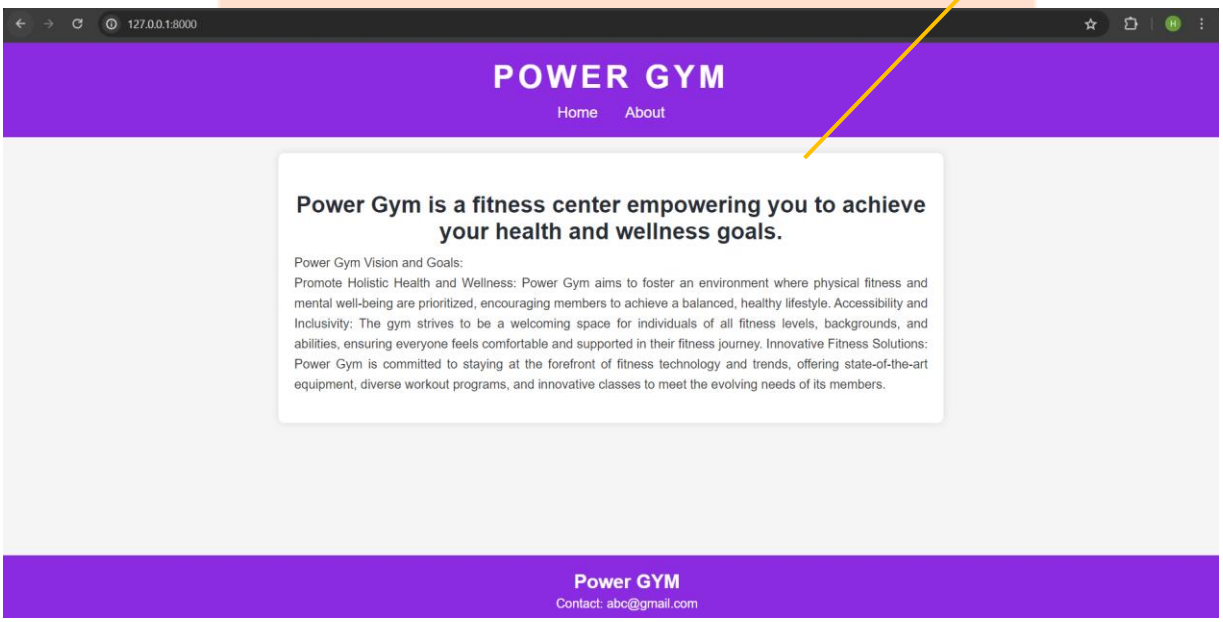
```python
from django.urls import path
from . import views
urlpatterns=[
    path('',views.display_view,name='home'),
    path('about/',views.about_view,name='about'),
]
```

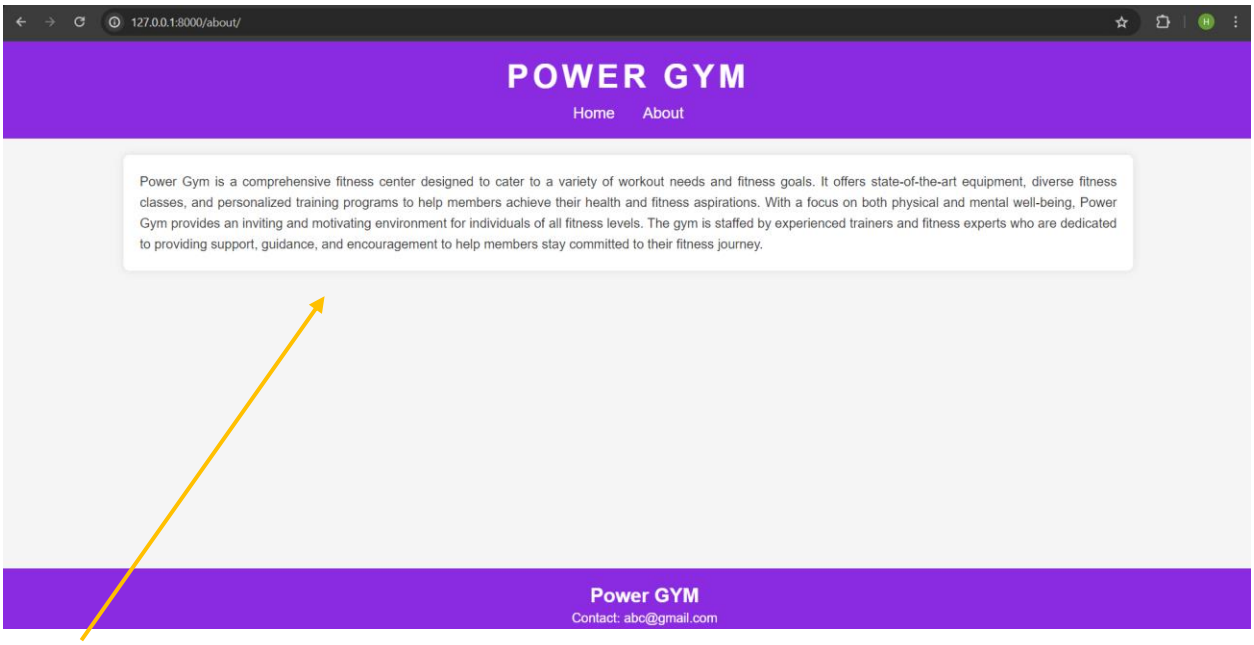
```python
from django.contrib import admin
from django.urls import path,include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('templateapp.urls')),
]
```

This part is of main.html. And the header and footer are coming from base.html

Step 4: Start the server and visit the URLs-





About.html

Step 6: (Optional) Include CSS for styling-

You can define your own styling or use ChatGPT to help you in styling the webpage by copying your template code and asking GPT to give you CSS file for it.

In the next chapter we are going to start learning Models and Databases.

If you are following the course series and learning the concepts easily, please subscribe to our YouTube channel. This will motivate us to create more educational, course videos and study material.

Join our growing community of tech enthusiasts! Subscribe to our YouTube channel, where we simplify programming languages in the easiest way possible. Gain a deeper understanding with our clear explanations and receive exclusive notes to enhance your learning journey. Don't miss out on valuable insights – hit that subscribe button now and embark on a programming adventure with us!

Subscribe to our channel:
https://www.youtube.com/@HTEASYLEARNING