

# HTML E TEMPLATES

# O QUE VEREMOS HOJE

**01** HTML

**02** ELEMENTOS HTML

**03** ESTRUTURA DO DOCUMENTO HTML

**04** TEMPLATES NO DJANGO

**05** VARIÁVEIS NO TEMPLATE

**06** ESTRUTURAS LÓGICAS NO TEMPLATE

# HTML

HTML (Hypertext Markup Language) é a linguagem de marcação padrão para criar e estruturar páginas web. Ela define o conteúdo DE um site, usando uma série de elementos e tags.

O navegador web interpreta o código HTML e o renderiza visualmente, exibindo o conteúdo e a formatação da página para o usuário final.

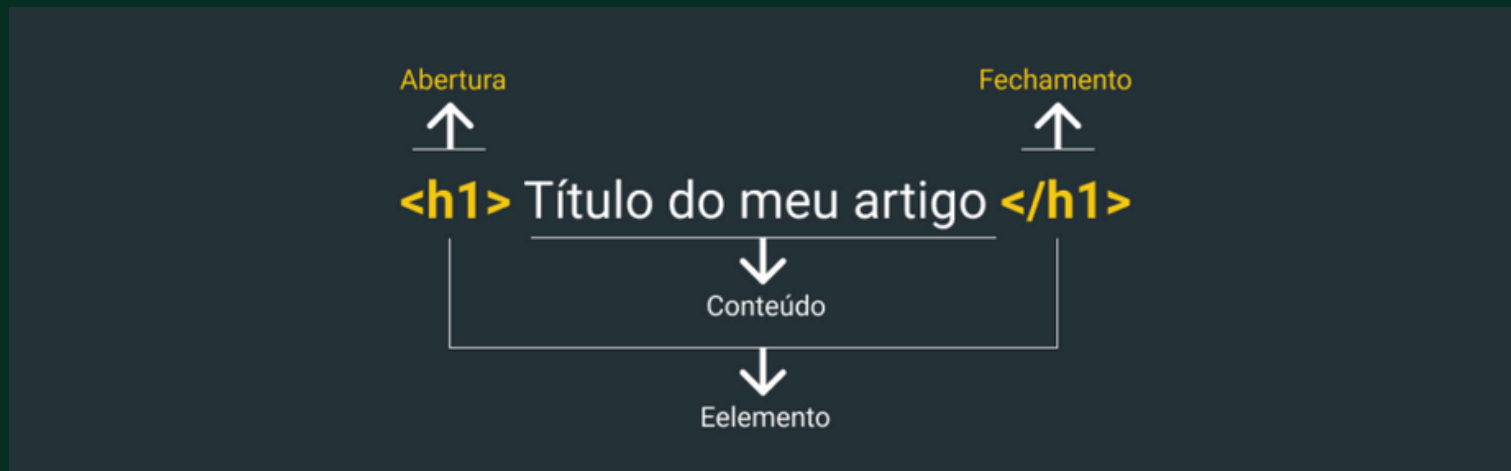
# HTML



# ELEMENTOS HTML

Os elementos são definidos por meio de tags e incluem todo o conteúdo na página, como textos, imagens, botões, etc.

As tags delimitam os elementos através da **tag de abertura e de fechamento**. O conteúdo do elemento é colocado entre as tags.



# ESTRUTURA DE UM DOCUMENTO HTML

Toda página HTML tem as tags `<!DOCTYPE>`, `<html>`, `<head>` e `<body>`, nessa ordem.

- O **`<!DOCTYPE>`** declara o tipo de documento e deve ser a primeira linha de qualquer documento HTML para garantir que o navegador interprete o código corretamente.
- A tag **`<html>`** é o elemento raiz que envolve todo o conteúdo da página HTML.
- O **`<head>`** contém informações sobre o documento, como o título da página, metadados, links para folhas de estilo e scripts JavaScript.
- O **`<body>`** contém o conteúdo visível da página, como textos, imagens, links e outros elementos que são exibidos no navegador.

# ESTRUTURA DE UM DOCUMENTO HTML

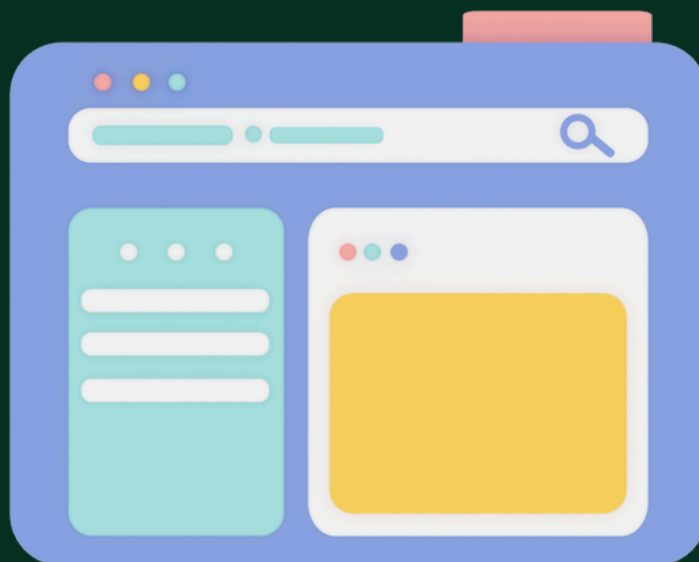


```
1  <!DOCTYPE html>
2  <html>
3      <head>
4
5      </head>
6      <body>
7
8      </body>
9  </html>
```

# TEMPLATES NO DJANGO

Os templates no Django são uma forma de separar o conteúdo visual da lógica da aplicação. Eles permitem criar páginas HTML dinâmicas, onde o layout e os dados exibidos são gerados de forma independente.

Com templates, é possível utilizar variáveis e comandos simples para exibir dados processados pelo Django diretamente no HTML, tornando as páginas mais interativas e personalizadas.



# TEMPLATES NO DJANGO

Para que o Django reconheça os templates é necessário adicionar o aplicativo ao **INSTALLED\_APPS** no **settings.py**.

Assim, o projeto do Django pode reconhecer cada aplicativo e os recursos de cada um, como templates, models, arquivos estáticos, etc.

```
1 # Application definition
2
3 INSTALLED_APPS = [
4     'django.contrib.admin',
5     'django.contrib.auth',
6     'django.contrib.contenttypes',
7     'django.contrib.sessions',
8     'django.contrib.messages',
9     'django.contrib.staticfiles',
10    'app' # nome do aplicativo Django (pasta principal do app, onde estão views.py, models.py, etc.)
11 ]
12
```



# CONFIGURAÇÕES DE TEMPLATES NO SETTINGS

O settings.py tem uma seção chamada TEMPLATES que controla onde o Django procura pelos templates e como eles são processados.

- **BACKEND:** é o motor de templates interpreta a sintaxe do Django permitindo usar python no HTML e permite gerar HTML dinâmico.
- **DIRS:** É uma lista de diretórios onde o Django buscará templates adicionais, fora das pastas dos apps.
- **APP\_DIRS:** Define se o Django deve procurar automaticamente por templates nas pastas templates/ dentro de cada app registrado no INSTALLED\_APPS.
- **OPTIONS:** Os context\_processors permitem adicionar informações contextuais automaticamente aos templates (como dados do usuário autenticado).

# RETORNANDO HTML NA VIEW

Para que uma view retorne um HTML, podemos usar a função `render()` do Django facilita a renderização de um template HTML.

Ela permite carregar um template e enviar como resposta ao navegador sem necessidade de lógica complexa.

Não precisamos importar o HTML na view, pois a função `render()` já sabe onde encontrar o template dentro da estrutura de diretórios do Django, desde que o arquivo HTML esteja na pasta `templates/`

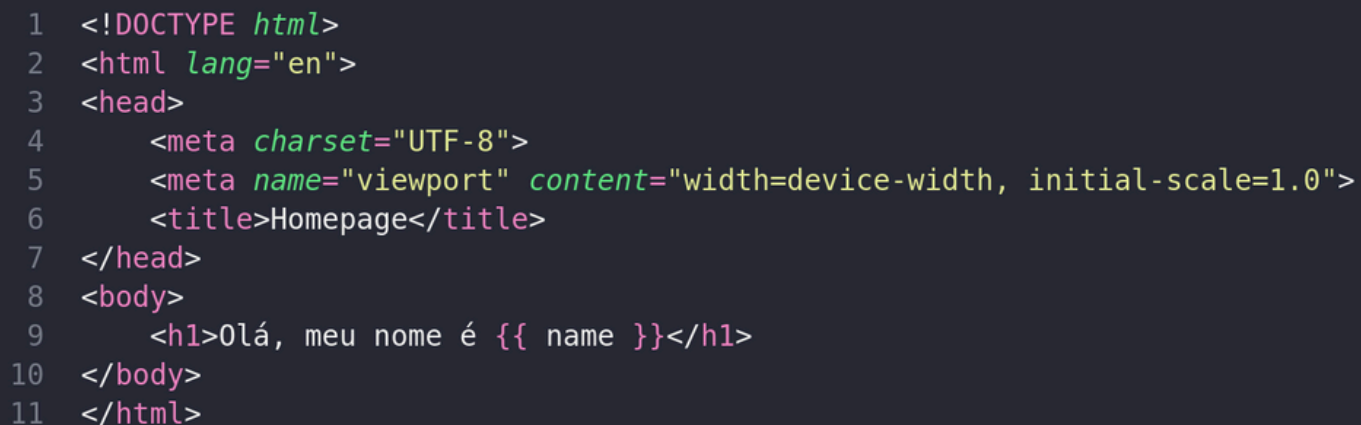


```
1  from django.shortcuts import render
2
3
4  def home(request):
5      return render(request, 'home.html')
```

# VARIÁVEIS NO HTML

No Django é possível gerar HTML dinâmicos que permitem exibir dados personalizados, que podem estar armazenados em um banco de dados ou gerados de forma dinâmica.

Para exibir esses dados, podemos usar variáveis dentro do código HTML usando a sintaxe `{{ nome_da_variavel }}`.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Homepage</title>
7 </head>
8 <body>
9     <h1>Olá, meu nome é {{ name }}</h1>
10 </body>
11 </html>
```

# VARIÁVEIS NO HTML

Mas para que essa variável seja acessível dentro do HTML, temos que passar essa informação através da view.

Essa informação é passada dentro de um dicionário. Ele serve para transferir informações dinâmicas da view (backend) para o template (frontend).



```
1  from django.shortcuts import render
2
3
4  def home(request):
5      context = {
6          'name': 'Aline'
7      }
8      return render(request, 'home.html', context)
9
```

# ATIVIDADE PRÁTICA

Crie um view para mostrar as informações de um produto específico no template.

A view deve passar as seguintes informações do produto:

- nome do produto
- descrição
- preço
- estoque


Essas informações devem aparecer no template.

# CONDICIONAIS NOS TEMPLATES

O Django possibilita usar o Python dentro do HTML para permitir a exibição dos dados de forma ainda mais dinâmica.

Uma das estruturas Python que podemos utilizar é o If para verificar condições e assim, mostrar ou ocultar partes do conteúdo de acordo com valores dinâmicos.

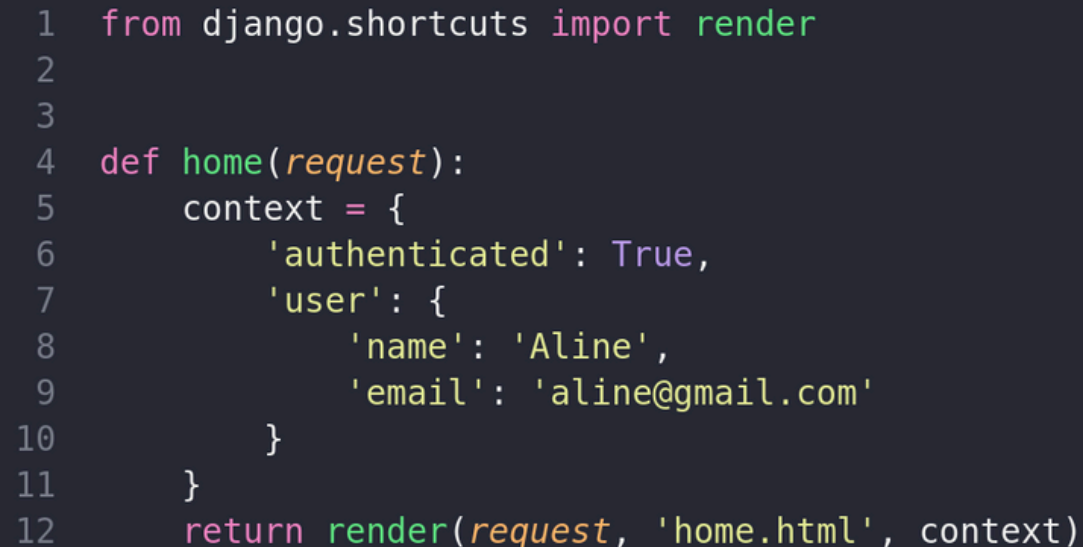
No Django, o conteúdo HTML é misturado com lógica de template (como if e for). Então, para o Django diferenciar HTML puro de código de template, usamos `{% %}` para envolver comandos.



```
1 {% if condição1 %}  
2     <!-- Conteúdo a ser exibido se condição1 for verdadeira -->  
3 {% elif condição2 %}  
4     <!-- Conteúdo a ser exibido se condição2 for verdadeira e condição1 for falsa -->  
5 {% else %}  
6     <!-- Conteúdo a ser exibido se nenhuma das condições anteriores for verdadeira -->  
7 {% endif %}
```

# CONDICIONAIS NOS TEMPLATES

Mas antes que possamos usar o `if`, também precisamos passar as informações de validação pela view, da mesma forma que é feito com as variáveis.

A screenshot of a code editor window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. The code is a Python function named `home` that takes a `request` object as an argument. It imports `render` from `django.shortcuts`. Inside the function, a `context` dictionary is created with the following values: `'authenticated': True`, and `'user': {'name': 'Aline', 'email': 'aline@gmail.com'}`. The function returns the result of `render(request, 'home.html', context)`.

```
1  from django.shortcuts import render
2
3
4  def home(request):
5      context = {
6          'authenticated': True,
7          'user': {
8              'name': 'Aline',
9              'email': 'aline@gmail.com'
10         }
11     }
12     return render(request, 'home.html', context)
```

# CONDICIONAIS NOS TEMPLATES

E no HTML podemos usar o If com esses dados que foram passados na view, usando a sintaxe de {% %} entorno do if, elif, else e endif.


```
1 <body>
2   <h1>Bem-vindo ao Nosso Site!</h1>
3
4   {% if authenticated %}
5     <p>Olá, {{ user.name }}! Estamos felizes em ver você de novo.</p>
6     <a href="">Sair</a>
7   {% else %}
8     <p>Olá, visitante! Faça <a href="">login</a> para acessar mais recursos.</p>
9   {% endif %}
10 </body>
```



# LOOPS NOS TEMPLATES

Além de condicionais, podemos usar loops para exibir dados dinâmicos.

A estrutura de loop possível de usar no Django é o for e possui uma sintaxe parecida com o if.



```
1 {% for item in lista %}
2     <!-- Conteúdo a ser repetido para cada item na lista -->
3 {% endfor %}
```


# LOOPS NOS TEMPLATES

Assim como nas variáveis e nas condicionais, precisamos passar a lista através da view.

```
1  from django.shortcuts import render
2
3
4  def home(request):
5      context = {
6          'users': [
7              {'username': 'Alice', 'age': 23},
8              {'username': 'Bob', 'age': 27},
9              {'username': 'Charlie', 'age': 31},
10         ]
11     }
12     return render(request, 'home.html', context)
13
```

# LOOPS NOS TEMPLATES

E no HTML, usamos a estrutura do for pra percorrer a lista e mostrar o conteúdo.



```
1  <body>
2      <h1>Usuários Cadastrados</h1>
3      <ul>
4          {% for user in users %}
5              <li>
6                  {{ user.name }} - {{ user.age }}
7              </li>
8          {% endfor %}
9      </ul>
10 </body>
```

## ATIVIDADE PRÁTICA

Crie uma lista de produtos, sendo cada um produto um dicionário da mesma forma que o exercício anterior (nome, descrição, preço e estoque).

A lista de produtos deve ser exibida no template usando um for, mas só devem aparecer os produtos que tiverem com o estoque acima de 0.