

# Django Templates



Prof. Bruno Gomes



@profbrunogomes



# Aula de Hoje

2

- » **Django:**
  - » **Templates**
  - » **Reaproveitando Templates**
  - » **Navegação entre Páginas**
  - » **Variáveis**
  - » **Estruturas de Controle**



# 1.

## Templates

# Documentação

4

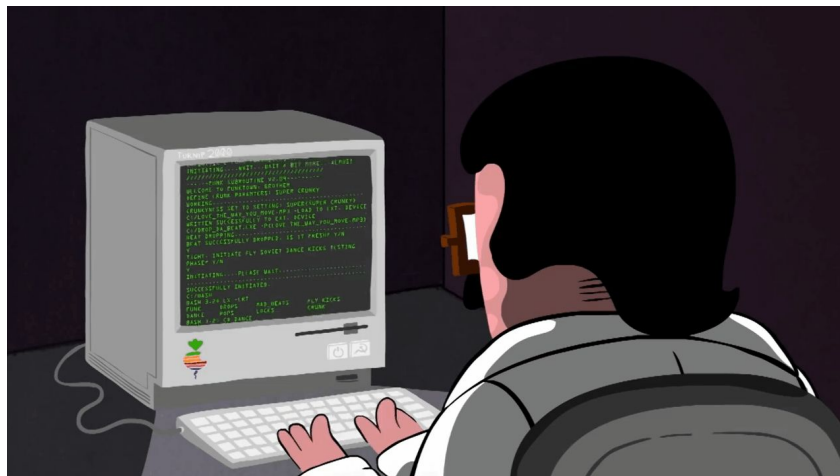
» <https://docs.djangoproject.com/en/4.1/topics/templates/>



# Antes de começar

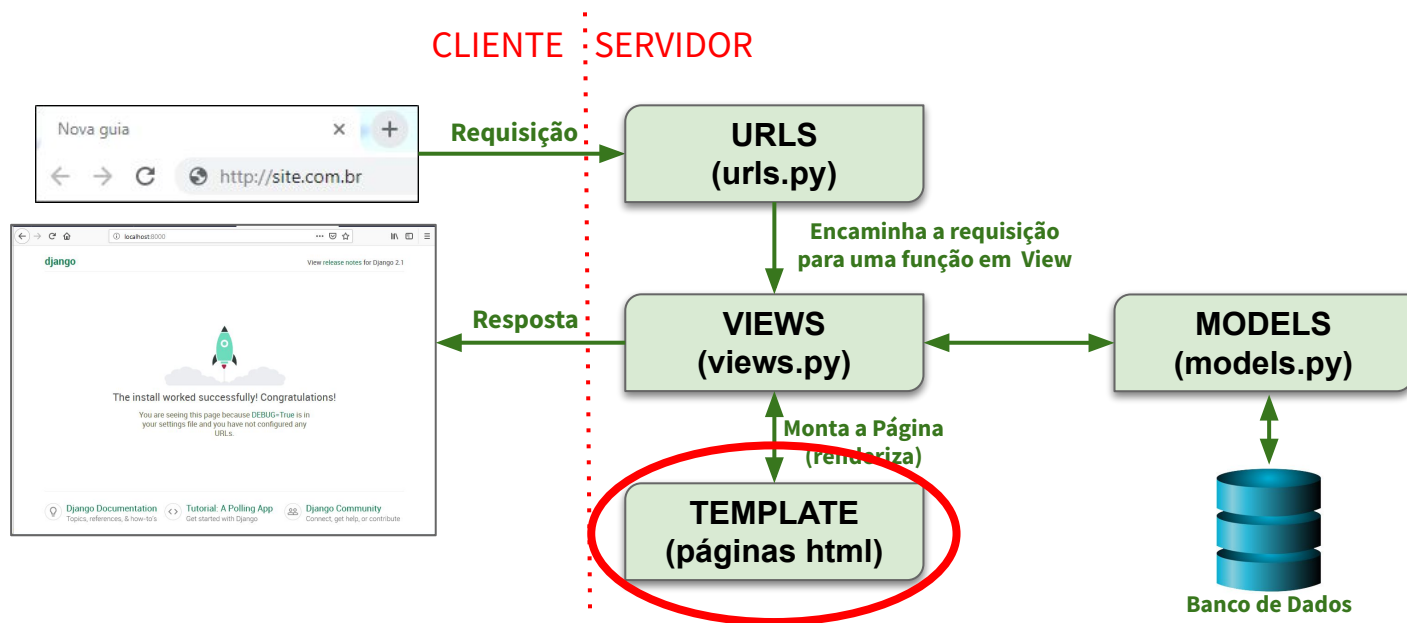
5

- » Criar um projeto chamado **projeto\_template**, e aplicativo chamado **core**.
  - » incluir o aplicativo no arquivo settings.py



# Templates

6





# Templates

7

- » Camada responsável por renderizar as páginas web (arquivos HTML, CSS e Javascript)
- » Django utiliza padrões (por *default*, vem configurado para o DjangoTemplates)
- » Arquivo **settings.py**:

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

projeto\_template/settings.py

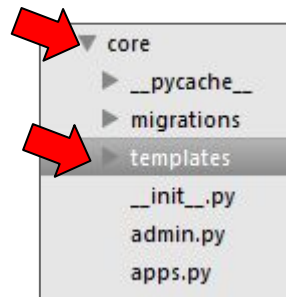
# Templates - Propriedades

8

- » **APP\_DIRS**: Se True, cada **aplicação** poderá ter uma pasta chamada templates, e o projeto irá procurar as páginas em cada uma delas.

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

projeto\_template/settings.py





# Templates - Propriedades

9

- » **DIRS** - Diretórios onde são encontrados os templates (html, por exemplo).
  - » Por padrão, não vem preenchido.
  - » As pastas templates criadas dentro das aplicações não precisam estar aqui. Apenas as que necessitam ser acessadas por toda a aplicação (por exemplo, pastas na raiz do projeto).

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

projeto\_template/settings.py

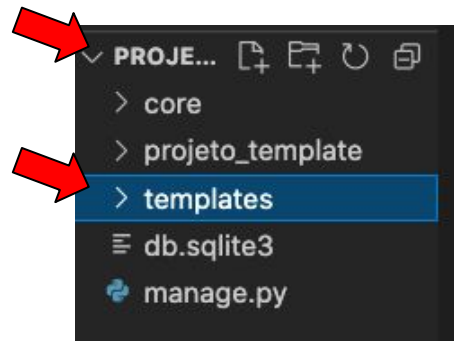
# Templates - Primeiro Exemplo

10

- » **Primeiro passo: criar** um diretório chamado “templates” na raiz do projeto e adicionar à configuração (settings.py).

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

projeto\_template/settings.py



# Templates - Primeiro Exemplo

11

- » **Segundo passo:** criar uma página HTML dentro da pasta templates, chamada **teste.html**



# Templates - Seguindo o padrão (templates)

12

- » **Terceiro passo:** Preencher a página, que servirá como template.

```
<!DOCTYPE html>
<html>
<head>
  <title>Primeira Página</title>
</head>
<body>
  <p>Página de Teste</p>
</body>
</html>
```

[core/templates/teste.html](#)

# Templates - Seguindo o padrão (templates)

13

- » **Quarto passo:** alterar a função home no arquivo **views.py**:

```
from django.shortcuts import render

def teste(request):
    return render(request, 'teste.html')
```

Verificar se já existe  
a importação do render

core/views.py



# Templates - Seguindo o padrão (templates)

14

- » **Quinto passo:** criar o path em urls.py para acessar a função anterior:

```
from django.contrib import admin
from django.urls import path
from core.views import teste

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

projeto\_template/urls.py

# Testando

15

- » Iniciar o servidor, e acessar:



```
http://localhost:8000/teste
```



Página de Teste

# Explicando

16

```
from django.shortcuts import render

def teste(request):
    return render(request, 'teste.html')
```

core/views.py

- » Ao invés de retornar uma função `HttpResponse`, agora iremos usar uma função chamada **render**, responsável por renderizar uma página:
  - » A renderização é responsável por executar os comandos python, inserindo informações na página que estavam em variáveis, ou executando estruturas de controle (condicional, repetição). Ao final, é gerada uma página somente com código HTML e enviada ao cliente como resposta.

# Dúvidas?



# Vamos Praticar

18

- » Criar uma nova página, chamada index.html, nela deverá ter um título de nível 2 com o texto:
  - » **Página Inicial.**
- » Ela deverá ser acessada através do endereço:
  - » <http://localhost:8000>





# 2.

## Reaproveitando Templates

# Documentação

20

» <https://docs.djangoproject.com/en/4.1/ref/templates/language/>



# Reaproveitando os Templates

21

- » É possível aproveitar um template, e utilizá-lo como base para as páginas do projeto, por exemplo:



# Reaproveitando os Templates

22

- » O reaproveitamento é feito importando/estendendo as partes do layout que se repetem nas páginas;



# Reaproveitando os Templates

23

- » O que será modificado em cada página será apenas o conteúdo.





## Qual a vantagem em reaproveitar template?

24

- » Se o layout mudar, não é necessário alterar o código em todas as páginas, somente na base (ou na parte importada).
- » Se for preciso acrescentar uma nova opção no menu, basta acrescentar somente no template menu, que irá afetar todas as outras páginas que importam este template.

## Como é feito em Django?

25

- » O Django reaproveita **herdando** o conteúdo de páginas!!
- » A herança é feita com uma tag chamada **extends**:
  - » O sistema de templates contém várias tags e filtros projetados para manipular a lógica da apresentação da aplicação.

# Herança de Templates

26

- » **Primeiro passo:** na pasta templates, criar um template que servirá de base para todas as páginas.
  - » Sugestão: criar um arquivo **base.html**



# Herança de Templates

27

- » **Segundo passo:** copiar todo o conteúdo de index.html para base.html, e acrescentar as tags:

```
<!doctype html>
<html>
<head>
<title>Título da Página</title>
</head>
<body>
    <p>Cabeçalho de Base</p>
    <p>Menu de Base</p>
    <p>Conteúdo de Base</p>
    <p>Rodapé de Base</p>
</body>
</html>
```

core/templates/base.html

## Para quem criou o Projeto agora

28

- » Criar a página **index.html**, e configurar:

views.py

```
def home(request):  
    return render(request, 'index.html')
```

urls.py

```
from core.views import home  
  
path('', home),
```



## Antes: Tags no Django

29

- » Para realizar a herança, é necessário utilizar uma tag, e sua sintaxe é:



- » Uma tag permite adicionar funcionalidades no template, como loops (for), lógica (if), criar texto de saída, herança, entre outros....

# Tags no Django

30

- » Algumas tags necessitam ser fechadas:



```
{% tag %}  
...conteúdo...  
{% endtag %}
```

## Voltando: Herança de Templates

31

- » **Terceiro passo:** apagar conteúdo de index.html, e acrescentar a herança abaixo:

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

core/templates/index.html

# Testando

32

- » Acessem a página:

Navegador

`http://localhost:8000`



Cabeçalho de Base

Menu de Base

Conteúdo de Base

Rodapé de Base

# Substituindo conteúdo no template base

33

- » Utiliza bloco de comando, e sua **sintaxe** é:

Template Pai

```
{% block nome_bloco %}  
conteudo_padrao(opcional)  
{% endblock %}
```

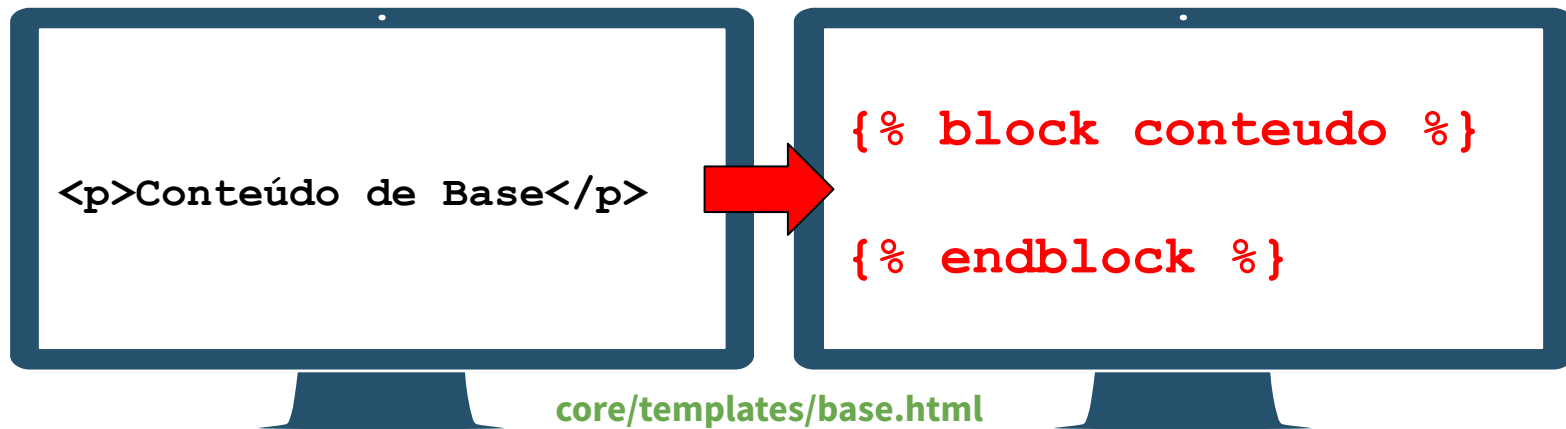
Template Filho

```
{% extends "base.html" %}  
  
{% block nome_bloco %}  
conteudo_que_substituirá  
{% endblock %}
```

# Substituindo conteúdo no template base

34

- » **Primeiro passo:** na **base**, substituir a tag de Conteúdo pelo bloco conteúdo:





## Substituindo conteúdo no template base

35

- » **Segundo passo:** em **index.html**, acrescentar a tag do bloco conteúdo:

```
{% extends "base.html" %}  
{% block conteúdo %}  
  
{% endblock %}
```

core/templates/index.html

## Substituindo conteúdo no template base

36

- » **Terceiro passo:** acrescentar um texto dentro do bloco em index.

```
{% extends "base.html" %}  
  
{% block conteudo %}  
<p>Conteúdo de Index</p>  
{% endblock %}
```

core/templates/index.html

# Testando

37

- » Acessem a página:

Navegador

`http://localhost:8000`



Cabeçalho de Base

Menu de Base

Conteúdo de Index

Rodapé de Base

# Exemplificando

38

index.html

```
{% extends "base.html" %}
{% block conteudo %}
<p>Conteúdo de Index</p>
{% endblock %}
```

http://localhost:8000

Cabeçalho de Base  
Menu de Base  
Conteúdo de Index  
Rodapé de Base

base.html

```
<body>
<p>Cabeçalho de Base</p>
<p>Menu de Base</p>
{% block conteudo %}
{% endblock %}
<p>Rodapé de Base</p>
</body>
```

## Situação: Texto padrão em Base

39

- » Existe um conteúdo padrão em base, que deve aparecer também em index:

core/templates/base.html

```
{% block conteudo %}  
<p>Conteúdo de Base</p>  
{% endblock %}
```

core/templates/index.html

```
{% block conteudo %}  
{{ block.super }}  
<p>Conteúdo de Index</p>  
{% endblock %}
```

# Testando

40

» Acessem a página:



`http://localhost:8000`

Navegador



Cabeçalho de Base

Menu de Base

Conteúdo de Base

Conteúdo de Index

Rodapé de Base



## Observação

41

- » `{{ }}` é a sintaxe para imprimir valores de variáveis
- » Neste exemplo, `block.super` recebe o valor definido no bloco chamado `conteudo`.

```
{% block conteudo %}  
{{ block.super }}  
<p>Conteúdo de Index</p>  
{% endblock %}
```

[core/templates/index.html](#)

# Dúvidas?



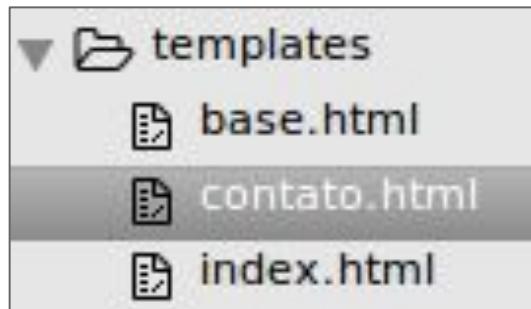
# 3.

## Navegação entre Páginas

## Navegação entre Páginas

44

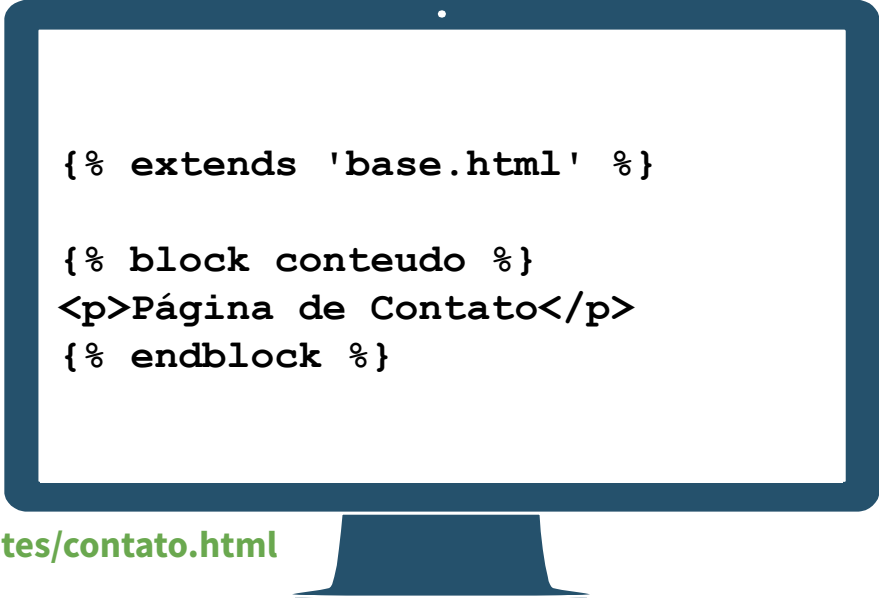
- » **Primeiro passo:** criar um novo template, correspondente ao menu contato: **contato.html**



# Navegação entre Páginas

45

- » **Segundo passo:**  
Acrescentar o  
conteúdo de  
contato em  
**contato.html**



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

{% block conteudo %}
<p>Página de Contato</p>
{% endblock %}
```

[core/templates/contato.html](#)

## Navegação entre Páginas

46

- » **Terceiro passo:** No arquivo **views.py**, acrescentar a função correspondente à nova página:

```
def contato(request):  
    return render(request, 'contato.html')
```

core/views.py



## Navegação entre Páginas

47

- » Quarto passo: no arquivo **urls.py**, alterar import e acrescentar o endereço de redirecionamento para a função contato:

```
from core.views import contato

urlpatterns = [
    path('contato/', contato),
]
```

projeto\_template/urls.py

# Testando

48

» Acessem a página:

Navegador

`http://localhost:8000/contato`



Cabeçalho de Base

Menu de Base

Página de Contato

Rodapé de Base

## Criando o Link

49

- » Em **index.html**, adicionar um link para a página de contato:

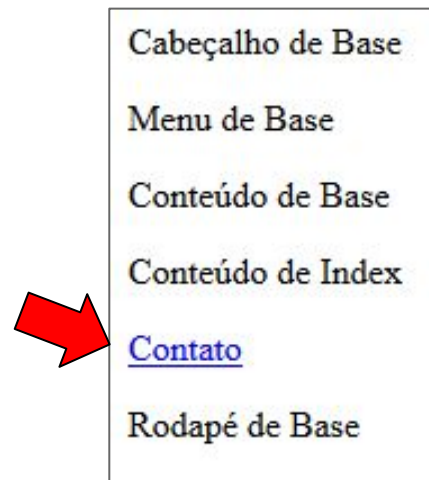
```
{% block conteudo %}  
{{ block.super }}  
<p>Conteúdo de Index</p>  
  
<p><a href="/contato">Contato</a></p>  
  
{% endblock %}
```

core/templates/index.html

# Testando

50

- » Acessar localhost, e clicar no link:



- » Se a página estiver em uma aplicação diferente ou mesmo em determinados diretórios da aplicação, não ficam visíveis e o link não funcionará.

## Solução: nomear as URLs

52

- » Basta nomear os links, e chamar pelo nome, que o acesso se dar independente da URL configurada (**urls.py**):

```
path('', home, name="home"),  
path('contato/', contato, name="contato"),
```

projeto\_template/urls.py



## Navegação entre Páginas

53

- » Em **index.html**, alterar o link de Contato para o nome da URL:

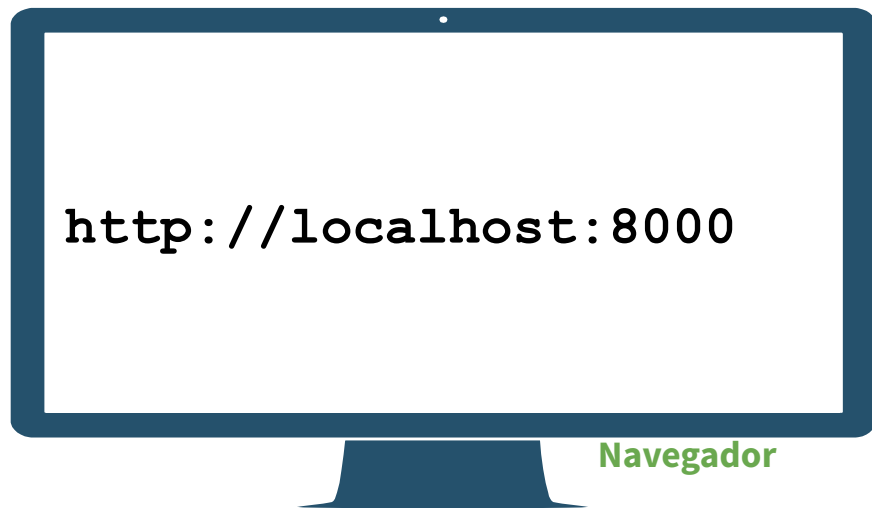
```
<p><a href="{% url 'contato' %}">Contato</a></p>
```

core/templates/index.html

# Testando

54

- » Acessar localhost, e clicar no link:



Navegador



Cabeçalho de Base

Menu de Base

Conteúdo de Base

Conteúdo de Index

[Contato](#)

Rodapé de Base

# Dúvidas?



## Vamos Praticar

56

- » Na página de contato, criar um link para index, utilizando o nome da URL.



- » Alterar as páginas já criadas para que o título delas apareçam assim:
  - » HOME | Site da disciplina
  - » CONTATO | Site da Disciplina
- » O que está marcado de vermelho é apenas para simbolizar o que se repete nelas (ou seja, será importado).

# 4.

## Parâmetros



# Parâmetros

59

- » É possível enviar parâmetros ao template;
- » O envio é feito na view, e a sintaxe para receber e exibir o valor de um parâmetro no template é através das chaves `{{ }}`;

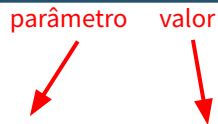
## Enviando um nome à Index

60

- » Primeiro Passo: Definir o parâmetro e o valor na view que será enviado ao template index.html:

```
def home(request):  
    return render(request, 'index.html', {'nome': 'Bruno'})
```

parâmetro    valor



core/views.py

## Recebendo o nome no template

61

- » Segundo Passo: Exibir o valor do parâmetro nome em **index.html** utilizando `{{ }}`:

```
<p><a href="{% url 'contato' %}">Contato</a></p>
```

```
<p>Olá {{nome}}</p>
```

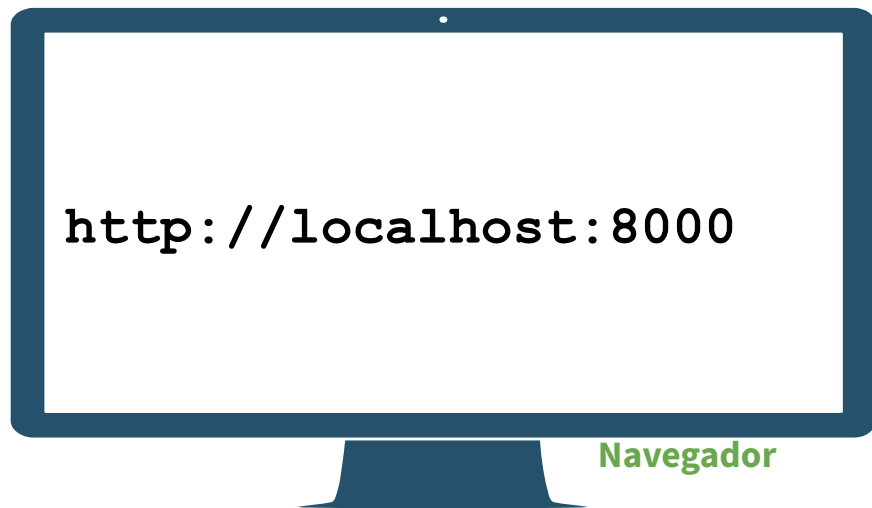
```
{% endblock %}
```

core/templates/index.html

# Testando

62

- » Acessar localhost, e clicar no link:



Cabeçalho de Base
Menu de Base
Conteúdo de Base
Conteúdo de Index
<u>Contato</u>
<b>Olá Bruno</b>
Rodapé de Base

# Enviando o valor de uma variável

63

- » Basta inserir no lugar do valor a variável

```
def home(request):  
    nome_usuario = 'Bruno'  
    return render(request, 'index.html', {'nome': nome_usuario})
```

core/views.py

# Passando vários parâmetros

64

- » Agrupar valores dos parâmetros em um único contexto:

```
def home(request):  
    contexto = {  
        'nome': 'Bruno',  
        'idade': 25,  
        'lista': [3, 2, 10, 1]  
    }  
    return render(request, 'index.html', contexto)
```

core/views.py



# Recebendo o nome no template

65

- » Exibir os valores das variáveis:

```
<p><a href="{% url 'contato' %}">Contato</a></p>
```

```
<p>Olá {{nome}}</p>
```

```
<p>Sua idade é {{idade}}</p>
```

```
{% endblock %}
```

core/templates/index.html

# Testando

66

- » Acessar localhost, e clicar no link:



Cabeçalho de Base

Menu de Base

Conteúdo de Base

Conteúdo de Index

Contato

Olá Bruno

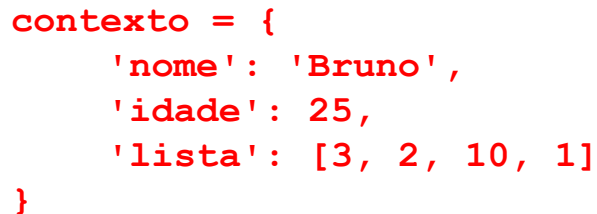
Sua idade é 25

Rodapé de Base

# Explicando

67

- » O contexto é um dicionário mapeando nomes de variáveis para objetos Python.



```
contexto = {  
    'nome': 'Bruno',  
    'idade': 25,  
    'lista': [3, 2, 10, 1]  
}
```

core/views.py

- » Documentação:
  - » <https://docs.djangoproject.com/en/2.2/intro/tutorial03/#write-views-that-actually-do-something>

# Variável no Contexto

68

- » É possível também enviar valores de variáveis no contexto:

```
def home(request):  
    nome_usuario = 'Bruno'  
    contexto = {  
        'nome': nome_usuario,  
        'idade': 25,  
        'lista': [3, 2, 10, 1]  
    }  
    return render(request, 'index.html', contexto)
```

core/views.py

# Testando

69

- » Acessar localhost, e clicar no link:



Cabeçalho de Base

Menu de Base

Conteúdo de Base

Conteúdo de Index

Contato

Olá Bruno

Sua idade é 25

Rodapé de Base

# Dúvidas?





# 5.

## Estruturas de Controle

## Parâmetros

72

- » É possível utilizar estruturas de condição e repetição diretamente no template;
  - » IF...ELSE
  - » FOR
- » A sintaxe para utilizá-las é {% %};

# Estrutura de Condição

73

## » Sintaxe

```
{% if condição %}
```

Código HTML - Caso condição verdadeira

```
{% else %}
```

Código HTML - Caso condição falsa

```
{% endif %}
```

## Exemplo: Estrutura de Condição

74

- » Utilizando o exemplo anterior, verificar pela idade se a pessoa é maior ou não (index.html):

```
<p>Sua idade é {{idade}}</p>
```

```
{% if idade < 18 %}
```

```
<p>Tem menos de 18 anos</p>
```

```
{% else %}
```

```
<p>Tem mais de 18 anos</p>
```

```
{% endif %}
```

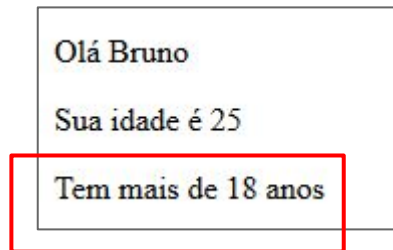
```
{% endblock %}
```

[core/templates/index.html](#)

# Testando

75

- » Acessar localhost, e clicar no link:



# Estrutura de Repetição

76

## » Sintaxe

```
{% for var in lista %}
```

Código HTML que será repetido

```
{% endfor %}
```



## Exemplo: Estrutura de Repetição

77

- » Adicionar um laço for para imprimir todos os números da lista (index.html):

```
<p>Tem mais de 18 anos</p>
{% endif %}

{% for numero in lista %}
<p>Valor: {{numero}}</p>
{% endfor %}

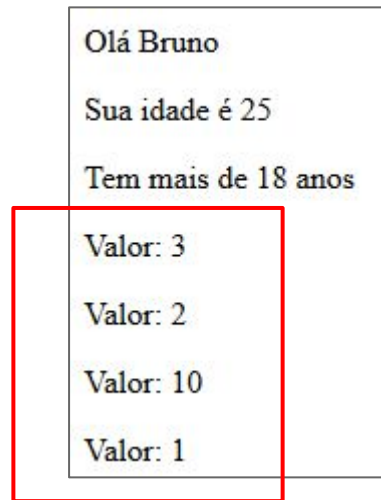
{% endblock %}
```

core/templates/index.html

# Testando

78

- » Acessar localhost, e clicar no link:



# Dúvidas?



# Atividade Avaliativa

80

- » Criar um projeto django que tenha:
  - » Uma página que sirva como base para as demais (sugestão: [base.html](#));
  - » Duas páginas HTML que herdam a base (inserir parágrafos e títulos nelas);
  - » Criar dois arquivos css, com no mínimo 2 propriedades em cada:
    - » Um deverá ficar dentro da pasta **statics** (padrão do django, que fica no aplicativo);
    - » E outro arquivo css ficará em uma **pasta definida por você** na raiz do projeto.
  - » Importar os arquivos css em base.html (as propriedades deverão afetar as 2 páginas que herdam base).
  - » Uma das páginas deverá exibir uma imagem que você desejar (esta imagem deverá estar salva na pasta **statics**).