



Flask

Programação Web III

Prof. Diego Max

Introdução

Python é uma das linguagens de programação mais populares e queridas por desenvolvedores, tanto iniciantes quanto veteranos.

Ela foi criada no final dos anos 80 por **Guido van Rossum**, que trabalhava no CWI (Centrum Wiskunde & Informatica) na Holanda.

Em 1991, a primeira versão pública do Python foi lançada. E o nome? Não tem nada a ver com cobras! Guido se inspirou no grupo de comédia britânico "**Monty Python's Flying Circus**".



Introdução



Introdução



Introdução

Python é uma linguagem de propósito geral, simples e fácil de aprender, o que a torna perfeita para quem está começando a programar.

Seu código é limpo, legível e parece muito com o inglês, o que facilita a escrita e manutenção.

Além disso, ela é extremamente versátil: você pode usá-la para **desenvolver websites, automatizar tarefas, analisar dados, criar inteligência artificial** e muito mais!



Aula 01 - Introdução ao desenvolvimento web com Flask

- Criando a primeira aplicação
- Renderizando views em HTML
- Enviando e recebendo dados

Aula 02 – Manipulação de dados, Templates e Estilos

- Controllers e Routes (Arquitetura MVC)
- Adicionando dados com Request
- Criando um arquivo base (template) HTML
- Manipulando arquivos estáticos (CSS, JS, IMG)

Aula 03 – Integração com APIs

- Consumindo uma API pública
- Passando parâmetros para rotas

Aula 04 – CRUD com banco de dados MySQL

- Criação do banco e leitura de dados
- Cadastro, alteração e exclusão de dados

Aula 05 – Login, segurança, sessões

- Cadastro de usuários
- Segurança, autenticação e sessões

Aula 06 – Upload de arquivos

- Upload de arquivos no Flask.

Aula 07 – Desenvolvimento de APIs

- Criando ambientes virtuais
- Desenvolvendo APIs Rest no Flask
- Deploy da API

Aula 08 – Consumo de APIs

- Desenvolvimento do front-end para consumo da API em Flask.

Métodos de avaliação



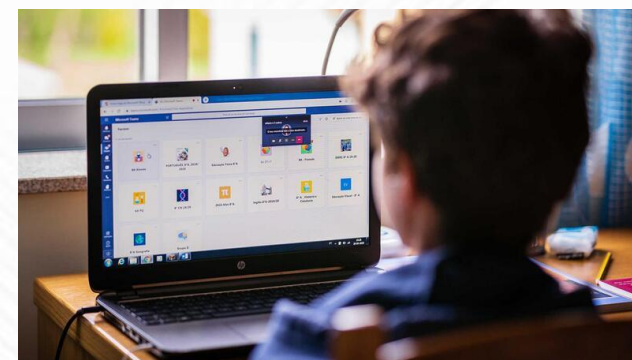
Avaliação escrita



Exercícios em sala



Participação em aula



Trabalhos

Flask

Aula 01: Introdução ao Flask

Desenvolvimento web com Python

Introdução ao Flask



Flask é um framework web compacto (micro-framework) amplamente utilizado no setor de programação com Python.

Dentre seus principais benefícios, destacam-se:

- Agilidade no desenvolvimento;
- Simplicidade na gestão de pacotes;
- Significativa robustez na capacidade de expansão;
- Eficiente desempenho performático.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def helloworld():
    return "Hello World!"

@app.route("/get_data")
def getdata():
    return "Your data"

if __name__ == "__main__":
    app.run()
```

Exemplo de código com Flask.

Introdução ao Flask



O Flask foi lançado em 01 de abril de 2010 e foi desenvolvido pelo austríaco Armin Ronacher.

Quando lançou o Flask, Armin tinha apenas 21 anos idade.

Hoje, com 36 anos, Armin é palestrante frequente em conferências de código aberto.



Criador do Flask, Armin Ronacher.

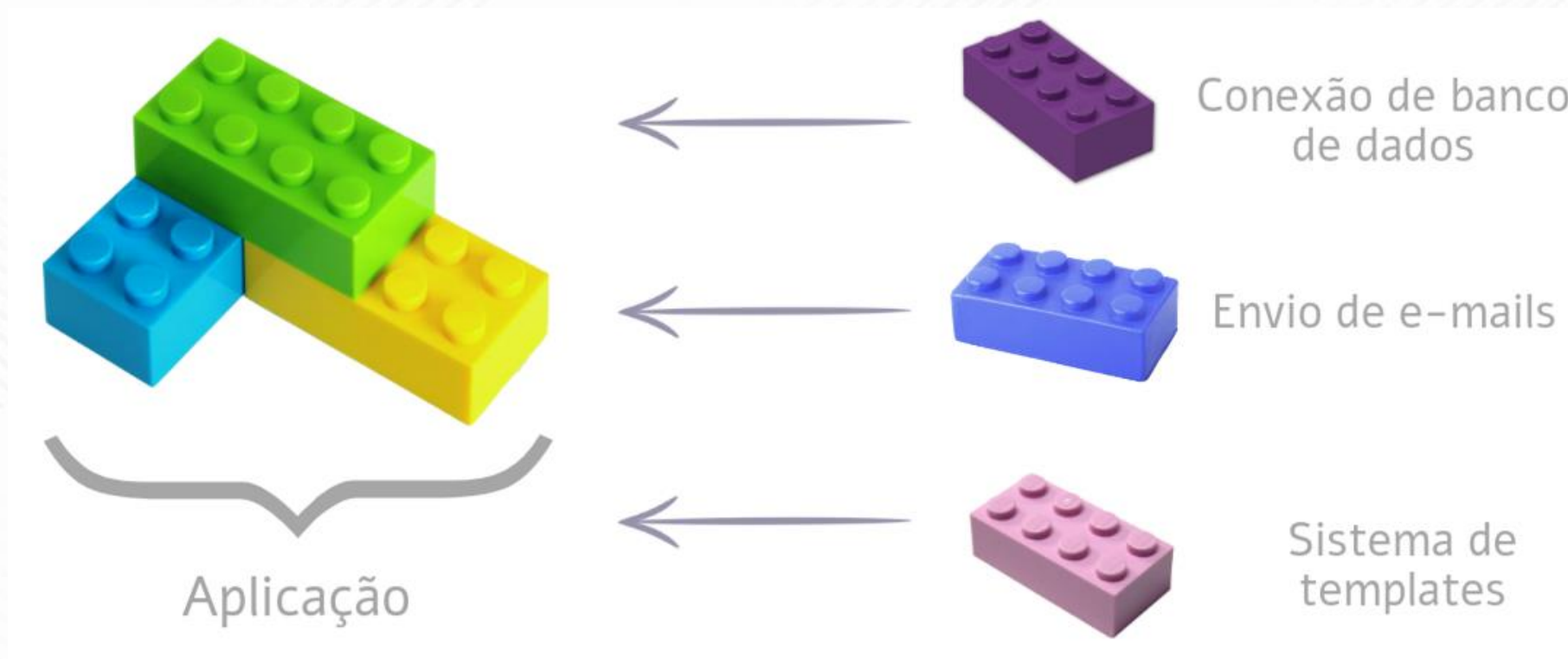


Logo e slogan do Flask.



Flask em inglês significa "Frasco".

Introdução ao Flask



Comparação de um Micro-Framework com peças de Lego.



django

vs



Flask
web development,
one drop at a time

Comparação entre Django e Flask.

Introdução ao Flask



:



:

NETFLIX

Uber



Flask

Tópico 01.1: Criando a primeira aplicação

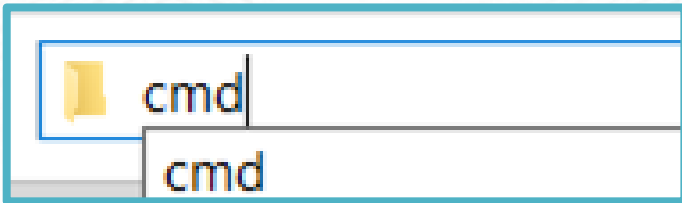
Introdução ao Flask

Criando a primeira aplicação



Com o Python já instalado no computador, o primeiro passo para criarmos nossa aplicação no Flask, é criar a pasta do projeto.

1. Crie uma pasta em um diretório de sua escolha que receberá os arquivos do projeto.
2. Criada a pasta devemos abri-la diretamente em nosso editor de código, usaremos o [VS Code](#). Uma maneira rápida de fazer isso é, com a pasta aberta, vá até a barra de endereço, digite “**cmd**” e aperte **Enter**, assim a pasta será aberta no [prompt de comando do Windows](#).



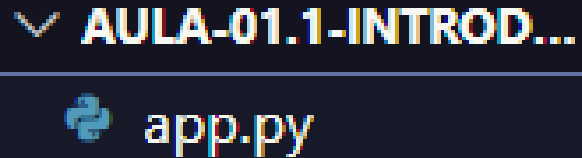
3. Feito isso, basta digitar no terminal “**code .**” e a pasta será aberta automaticamente no VS Code.

```
python_flask\aula-01.1-introducao-flask>code .
```

Criando a primeira aplicação



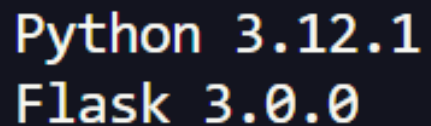
4. Com a pasta já aberta no editor de códigos, criaremos o primeiro arquivo do projeto que receberá o nome de “**app.py**”

A screenshot of the VS Code file explorer showing a folder named 'AULA-01.1-INTROD...' with a sub-file named 'app.py' next to it, indicated by a Python icon.

5. O próximo passo é instalar o Flask. Abra o terminal do VS Code com as teclas **CTRL + aspas** (“), e no terminal digite “**pip install flask**” e aperte **Enter**.

A screenshot of a terminal window with a dark background. The text '> pip install flask' is entered, followed by a white cursor block.

6. Quando a instalação do Flask terminar, verifique sua versão com o comando “**flask --version**”

A screenshot of a terminal window showing the command 'flask --version' entered.A screenshot of a terminal window showing the output of the 'flask --version' command: 'Python 3.12.1' and 'Flask 3.0.0'.

Criando a primeira aplicação



7. Agora começaremos a incluir os códigos no arquivo “**app.py**”. Os código devem ser incluídos em sequência, conforme sua ordem de apresentação:

```
from flask import Flask
```

O `flask` minúsculo é a biblioteca Python que você instalou. O `Flask` em maiúsculo é uma classe dessa biblioteca que deve ser importada.

```
app = Flask(__name__)
```

Aqui criamos uma instância da classe “`Flask`” em um novo objeto chamado “`app`”. Agora “`app`” possui todos os métodos e atributos da classe “`Flask`”.

O que está entre parênteses é a variável “`__name__`”, que possui o nome da nossa aplicação. Por padrão, quando estamos executando um arquivo Python pelo interpretador, o “`__name__`” desse arquivo recebe o valor “`__main__`”. Se esse arquivo for importado como um `módulo`, o “`__name__`” recebe como valor o seu *filename* original, ou seja, o nome que foi dado ao arquivo.

Criando a primeira aplicação



```
@app.route('/')  
  
def home():  
    return '<h1>Esta é a homepage</h1>'
```

Um **decorator** é definido pelo símbolo de `@`. Exemplo: `@app.route('/')`. Um **decorator** atribui uma nova funcionalidade para a função que está abaixo. Logo estamos definindo que a função `home()` que retorna o texto “*Esta é a homepage*”, seja acessada através da rota “/” que é a rota principal da aplicação.

```
if __name__ == '__main__':  
    app.run(host='localhost', port=5000,  
            debug=True)
```

Nesta linha criamos uma **condição**: se o valor de “`__name__`” for igual a “`__main__`”, ou seja, como vimos anteriormente, o valor “`__main__`” só é atribuído a variável “`__name__`” quando executamos o arquivo diretamente pelo interpretador. Então, se esse for o caso, a aplicação será executada em **localhost**, na **porta 5000**, com o modo **debug ativado**. O modo **debug** serve para que nossa aplicação seja constantemente **atualizada** durante o desenvolvimento.

Criando a primeira aplicação



Para rodarmos a aplicação no servidor, devemos digitar no terminal do VS Code o comando “**python app.py**”.

```
> python app.py
```

Com a aplicação rodando, basta acessarmos o endereço **localhost:5000** no navegador:



Você acaba de criar sua primeira aplicação com Flask. No próximo tópico veremos como renderizar uma página HTML com o Flask.

Criando a primeira aplicação



Aula 01.1 - Código completo:

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return '<h1>Esta é a Homepage!</h1>'
if __name__ == '__main__':
    app.run(host='localhost', port=5000,
debug=True)
```


Flask

Tópico 01.2: Renderizando views em HTML

Introdução ao Flask

Renderizando views em HTML



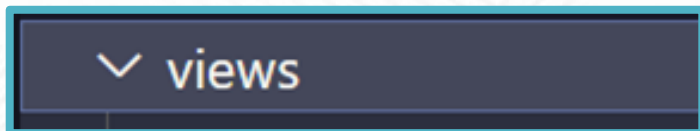
Começaremos agora a criar as páginas HTML que farão parte de nossa aplicação. O primeiro passo é importar o `render_template` do Flask como no código a seguir:

```
from flask import Flask, render_template
```

Após isso, iremos definir o nome da pasta que irá armazenar as nossas views, ou seja, nossas páginas em HTML.

```
app = Flask(__name__, template_folder='views')
```

Caso não seja definido nenhum nome, por padrão o Flask irá buscar as páginas em uma pasta de nome “`templates`”. No nosso caso, definimos que essa pasta terá o nome de “`views`”, essa pasta deve ser criada na raiz da pasta do seu projeto.



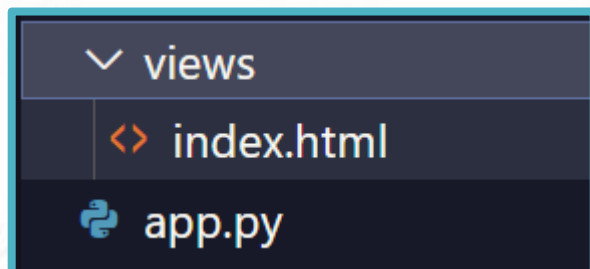
Renderizando views em HTML



Agora na função `home()`, aonde estávamos apenas retornando um texto iremos renderizar a página “`index.html`” quando a rota principal for acessada, conforme o código a seguir:

```
@app.route('/')  
  
def home():  
    return render_template('index.html')
```

Como a página “`index.html`” ainda não existe iremos cria-la dentro da pasta “`views`” criada anteriormente.



Renderizando views em HTML



No arquivo “index.html” criaremos o seguinte código HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home</title>
</head>
<body>
  <h2>Esta é a Homepage.</h2>
  <hr>
</body>
</html>
```

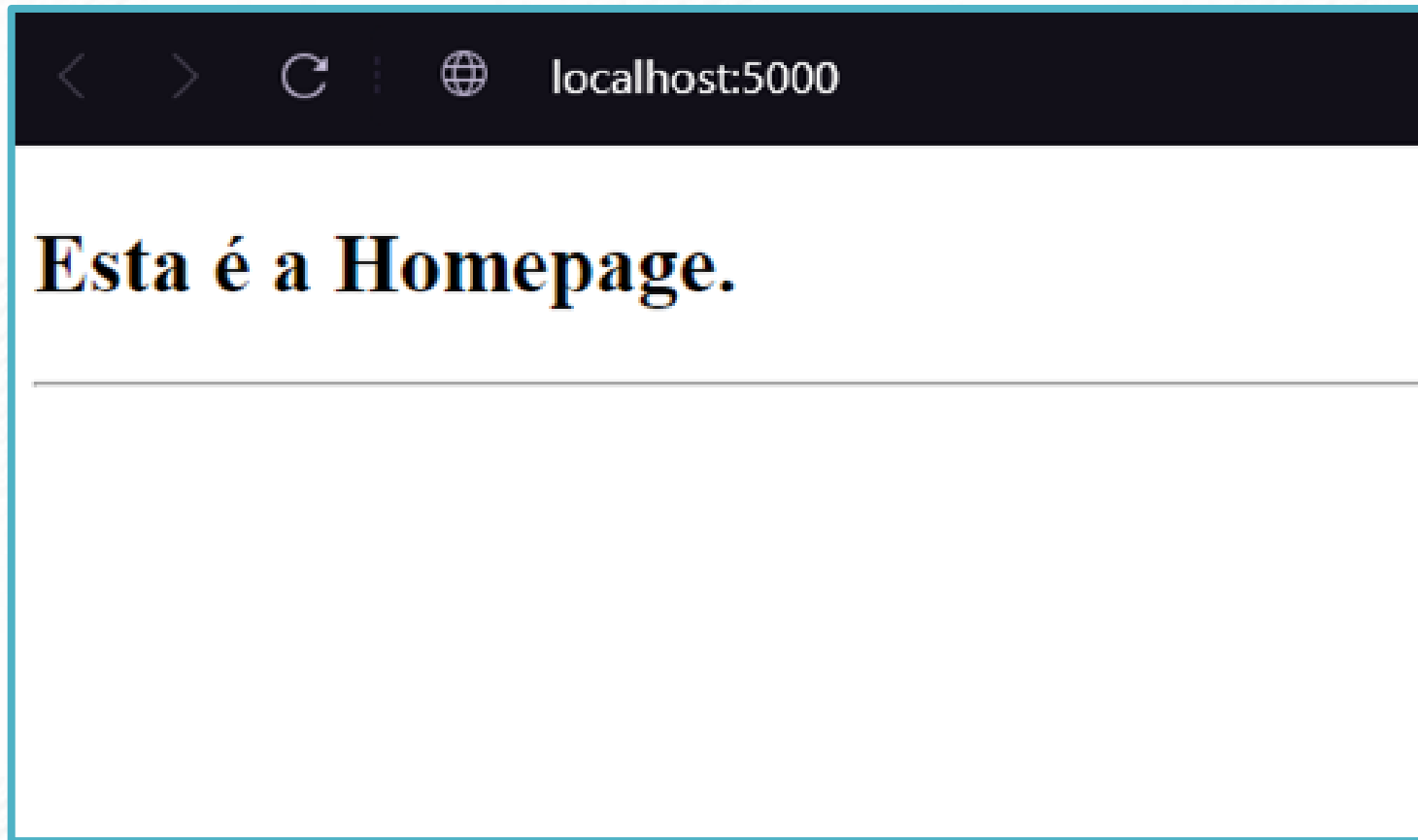
Feito isso, acessaremos novamente nossa aplicação pelo navegador em “localhost:5000”.
Se necessário, rode novamente o comando “**python app.py**” para iniciar aplicação no servidor.

```
> python app.py
```

Renderizando views em HTML



Temos a seguinte saída:



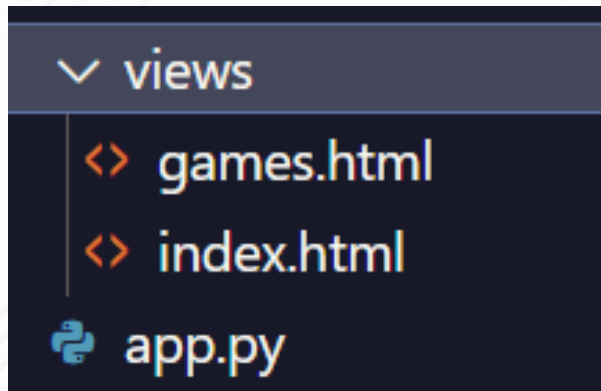
Renderizando views em HTML



Feito isso, criaremos agora uma nova rota chamada “/games”, aonde iremos renderizar a página “games.html”.

```
@app.route('/games')  
  
def games():  
    return render_template('games.html')
```

Assim, agora temos a seguinte estrutura:



Renderizando views em HTML



No arquivo “`games.html`” criaremos o seguinte código HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Games</title>
</head>
<body>
  <h2>Esta é a página de Games.</h2>
  <hr>
</body>
</html>
```

Feito isso, acessaremos novamente nossa aplicação pelo navegador, porém agora a rota a ser acessada deve ser “`localhost:5000/games`”.

Renderizando views em HTML



Temos a seguinte saída:



Nossa aplicação já está renderizando páginas em HTML. No próximo tópico veremos como enviar e receber dados nessas páginas.

Flask

Tópico 01.3: Enviando e recebendo dados

Introdução ao Flask

Enviando e recebendo dados



Aprenderemos agora como enviar e receber valores de variáveis nas páginas. Para isso, criaremos algumas variáveis dentro da função “`games()`” e atribuiremos valores a elas.

Após isso, dentro dos parênteses da função `render_template()` passaremos os valores dessas variáveis, depois do nome da página que está sendo renderizada, conforme código a seguir:

```
@app.route('/games')
def games():
    titulo = 'CS-GO'
    ano = 2012
    categoria = 'FPS Online'
    return render_template('games.html',
                           titulo=titulo,
                           ano=ano,
                           categoria=categoria)
```

Enviando e recebendo dados



Agora, para que essas variáveis sejam exibidas em suas páginas de destino utilizaremos o template engine **Jinja**.

O Jinja é um template engine escrito em Python que facilita a criação de páginas HTML em aplicações Python.

Basicamente, ele serve para permitir que as informações trocadas entre uma aplicação escrita em Python e suas páginas HTML seja feita de forma mais simples e intuitiva, garantindo que o desenvolvedor consiga criar templates de forma mais fácil para suas aplicações.



Para mesclar código Python dentro de páginas HTML utilizamos as chaves do Jinja `{{ }}` ou `{% %}`, como veremos a seguir.

Enviando e recebendo dados



Na página “`games.html`” criaremos uma lista `` com os valores das variáveis que foram passados anteriormente, para isso utilizaremos as chaves do Jinja, conforme código a seguir:

```
<body>
  <h2>Esta é a página de Games.</h2>
  <hr>
  {# Comentário no jinja2 #}
  <ul>
    <li>Título: {{titulo}}</li>
    <li>Ano: {{ano}}</li>
    <li>Categoria: {{categoria}}</li>
  </ul>
  <p>0 jogo {{titulo}} tem {{2024-ano}} anos.</p>
</body>
```

Obs: Para comentar no Jinja utilize `{# #}`

Enviando e recebendo dados



Agora, na página “`games.html`” teremos a seguinte saída:

Esta é a página de Games.

- Título: CS-GO
- Ano: 2012
- Categoria: FPS Online

O jogo CS-GO tem 12 anos. $\longrightarrow 2024 - 2012 = 12$

Você aprendeu a enviar e receber dados nas páginas com o Jinja. No próximo tópico veremos como adicionar valores em uma lista Python.

Flask

Tópico 01.4: Enviando valores em uma lista

Introdução ao Flask

Enviando valores em uma lista



Visto como enviar valores por variáveis simples, agora iremos passar valores para a página através da estrutura de dados `lista` do Python.

Para isso criaremos uma lista chamada “`jogadores`” que recebeu alguns nomes dos jogadores atuais daquele jogo. Feito isso, também passamos a lista “`jogadores`” para página “`games.html`” junto com as outras variáveis que vimos anteriormente.

```
@app.route('/games')
def games():
    titulo = 'CS-GO'
    ano = 2012
    categoria = 'FPS Online'
    jogadores = ['Pedro', 'João', 'Marcos', 'Maria', 'Diego']
    return render_template('games.html', titulo=titulo, ano=ano, categoria=categoria,
                           jogadores=jogadores)
```

Enviando valores em uma lista



Na página “[games.html](#)”, basta exibirmos a lista dentro das chaves `{{}}` do Jinja, assim como fizemos com as outras variáveis.

O código da página “[games.html](#)” ficará assim:

```
<ul>
  <li>Título: {{titulo}}</li>
  <li>Ano: {{ano}}</li>
  <li>Categoria: {{categoria}}</li>
</ul>
<p>O jogo {{titulo}} tem {{2024-ano}} anos.</p>
<p>Está sendo jogador por:</p>
<p>{{jogadores}}</p>
</body>
```


Enviando valores em uma lista



Agora acessando a rota “/games”, o resultado será o seguinte:

Esta é a página de Games.

- Título: CS-GO
- Ano: 2012
- Categoria: FPS Online

O jogo CS-GO tem 12 anos.

Está sendo jogado por:

['Pedro', 'João', 'Marcos', 'Maria', 'Diego'] → Lista “jogadores”

Enviando valores em uma lista



Agora iremos exibir os nomes dos jogadores em uma **lista ordenada** do HTML, dentro da estrutura de repetição **for**.

No Jinja para incluirmos um código em Python começamos com **{%** e fechamos com **%}**
No caso do **for**, para encerramos o **loop**, fechamos com **{% endfor %}**. O código ficará assim:

```
<p>Está sendo jogador por:</p>
<ol>
    {% for j in jogadores: %}
        <li>{{j}}</li>
    {% endfor %}
</ol>
```

Enviando valores em uma lista



Agora acessando a rota “/games”, teremos a seguinte saída:

- Título: CS-GO
- Ano: 2012
- Categoria: FPS Online

O jogo CS-GO tem 12 anos.

Está sendo jogador por:

1. Pedro
2. João
3. Marcos
4. Maria
5. Diego

Itens da lista “jogadores”
sendo exibidos através do **for**.

Você aprendeu a enviar valores de uma **lista** Python e exibi-los na página. No próximo tópico veremos como enviar e receber valores de um **dicionário**.

Flask

Tópico 01.5: Enviando valores em um dicionário

Introdução ao Flask

Enviando valores em um dicionário



Visto como enviar valores por uma **lista**, veremos agora como enviar e exibir valores através da estrutura de dados **dicionário** do Python. Os dicionários diferentes das listas, são compostos por uma **chave** (**key**) e um **valor** (**value**). Um dicionário é iniciado com uma chave { e é fechado com outra chave }.

Criaremos agora um dicionário com o nome “**game**”, com suas respectivas **chaves** e **valores**. Após isso, passamos esse dicionário para a página “**games.html**”, junto com nossa lista “**jogadores**” que criamos anteriormente.

```
@app.route('/games')
def games():
    game = {'Título' : 'CS-GO',
            'Ano' : 2012,
            'Categoria' : 'FPS Online'}
    jogadores = ['Pedro', 'João', 'Marcos', 'Maria', 'Diego']
    return render_template('games.html',
                           game=game,
                           jogadores=jogadores)
```

Enviando valores em um dicionário



Já no HTML, iremos utilizar da estrutura de repetição `for` para exibir os dados do dicionário. Onde “`k`” serão as `chaves` (`keys`) e “`v`” os `valores` (`values`).

Repare também que no parágrafo que calcula a idade do jogo, tivemos que alterar para “`game['Título']`” e “`game['Ano']`”, já que esses dados agora estão dentro do nosso dicionário “`game`”.

```
<body>
  <h2>Esta é a página de Games.</h2>
  <hr>
  {% for k, v in game.items() %}
  {{k}}: {{v}}<br>
  {% endfor %}
  <p>O jogo {{game['Título']}} tem {{2024-game['Ano']}} anos.</p>
  <p>Está sendo jogador por:</p>
```

Enviando valores em um dicionário



Agora acessando a rota “/games”, teremos a seguinte saída:

Esta é a página de Games.

Título: CS-GO

Ano: 2012

Categoria: FPS Online

O jogo CS-GO tem 12 anos.

Está sendo jogador por:

1. Pedro
2. João
3. Marcos
4. Maria
5. Diego

Itens do dicionário “game”
sendo exibidos através do **for**.

Você aprendeu a enviar e receber valores de um dicionário Python, encerrando essa aula de **Introdução ao Flask**. Na próxima aula veremos: **Arquitetura MVC, Requisições HTTP, Templates e Static Files**.



Flask

Programação Web III

Prof. Diego Max