

FLASK

*FORJANDO APLICAÇÕES
WEB COM PYTHON*



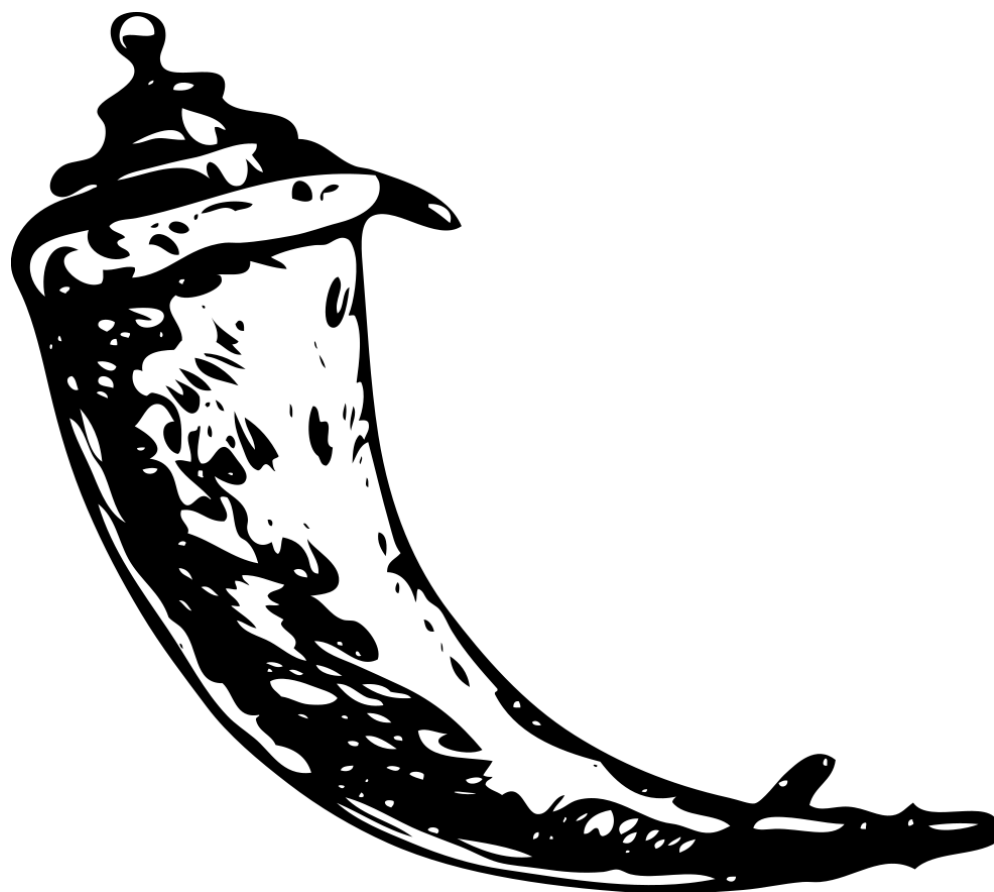
FERNANDO SOARES VIEIRA

INTRODUÇÃO

Guia Prático de Seletores FLASK

Obtenha o poder e molde a Web utilizando Python

Flask é um microframework web para Python, conhecido pela sua simplicidade e flexibilidade. Ele permite que você desenvolva aplicações web rapidamente, com uma curva de aprendizado suave. Neste ebook, vamos explorar os principais seletores e funcionalidades do Flask com exemplos de código em contextos reais.



01

INSTALAÇÃO DO FLASK

Antes de começarmos a criar aplicações com Flask, precisamos instalar o framework no nosso ambiente de desenvolvimento. Neste capítulo, vamos abordar o processo de instalação do Flask em diferentes sistemas operacionais: Windows, Linux e macOS.

PRÉ-REQUISITOS

Para instalar o Flask, você precisa ter o Python instalado em sua máquina. Você pode verificar se o Python está instalado e qual a versão utilizando o seguinte comando prompt de comando:

 Prompt de Comando

```
python --version
```

INSTALAÇÃO NO WINDOWS

1. Abrindo o Prompt de Comando:

- Pressione “Win + R”, digite ‘cmd’ e pressione ‘Enter’.

2. Criando um Ambiente Virtual (opcional, mas recomendado):

- Navegue até o diretório onde você deseja criar seu projeto:

```
Prompt de Comando  
cd caminho\para\seu\projeto
```

- Crie o ambiente virtual:

```
Prompt de Comando  
python -m venv venv
```

- Ative o ambiente virtual:

```
Prompt de Comando  
venv\Scripts\activate
```

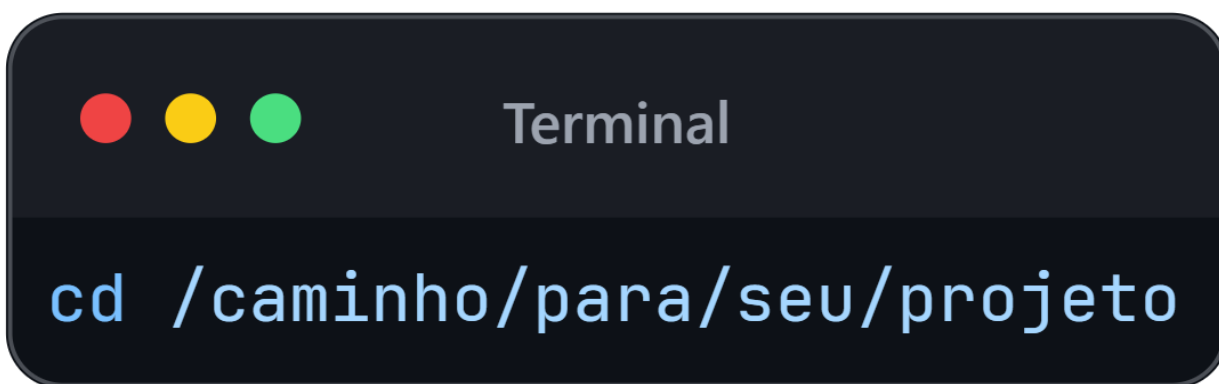

INSTALAÇÃO NO LINUX/MACOS

1. Abrindo o Prompt de Comando:

- Linux: Pressione Ctrl + Alt + T para abrir o terminal.
- MacOS: Pressione Command + Space, digite Terminal e pressione Enter.

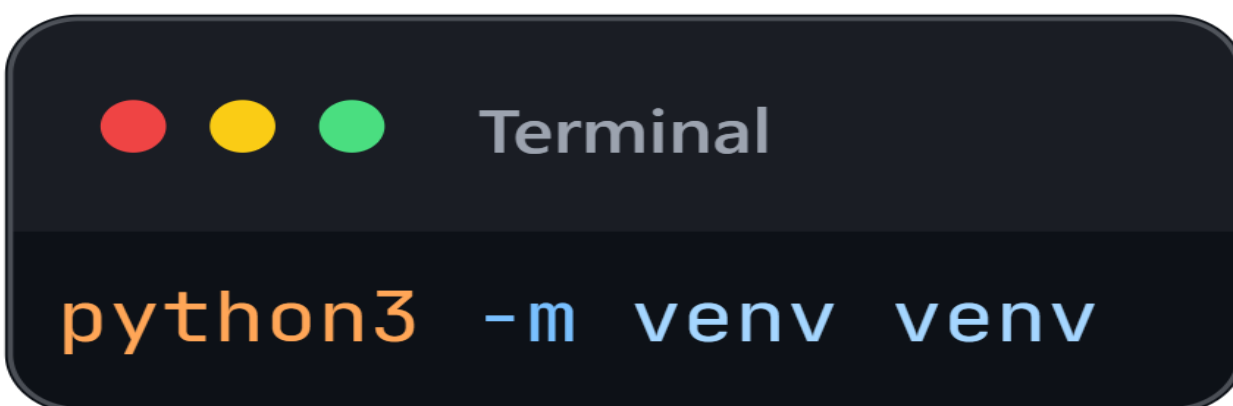
2. Criando um Ambiente Virtual (opcional, mas recomendado):

- Navegue até o diretório onde você deseja criar seu projeto:



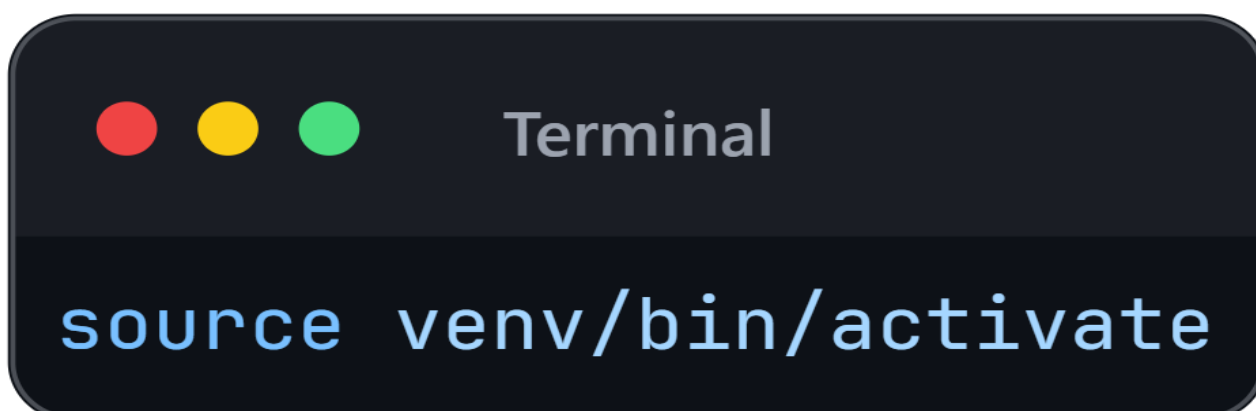
```
cd /caminho/para/seu/projeto
```

- Crie o ambiente virtual:



```
python3 -m venv venv
```

- Ative o ambiente virtual:



```
source venv/bin/activate
```

INSTALANDO O FLASK

Com o ambiente virtual ativado, instale o Flask usando o pip (Em todos os sistemas operacionais):



Prompt de Comando / Terminal

```
pip install Flask
```

Para confirmar que o Flask foi instalado corretamente, você pode executar o seguinte comando:



Prompt de Comando / Terminal

```
python -m flask --version
```


02

CRIANDO A PRIMEIRA APLICAÇÃO

Agora que o Flask está instalado, vamos criar um pequeno projeto para garantir que tudo está funcionando corretamente.

CRIANDO UM PROJETO FLASK

1. Crie um arquivo chamado *'app.py'*:

```
app.py

from flask import Flask # Importa a classe Flask

app = Flask(__name__) # Cria uma instância da aplicação Flask

@app.route('/') # Define a rota para a URL raiz
def home():
    return "Hello, Flask!" # Função que retorna a mensagem "Hello, Flask!"

if __name__ == '__main__':
    app.run(debug=True) # Executa a aplicação em modo debug
```

2. Execute a aplicação no CMD/Terminal:

```
CMD / Terminal

python app.py
```

3. Abra seu navegador e acesse:

```
Navegador

http://127.0.0.1:5000/
```



03

ROTEAMENTO E CRIAÇÃO DE ROTAS

O roteamento é uma parte essencial de qualquer aplicação web. Em Flask, usamos o decorador `@app.route` para definir rotas que mapeiam URLs para funções específicas.

@APP.ROUTE

O seletor ‘*@app.route*’ é utilizado para mapear URLs para funções Python. Vamos ver um exemplo:

```
app.py

@app.route('/about') # Define a rota para a URL /about
def about():
    return "Sobre nós" # Função que retorna a mensagem "Sobre nós"
```

Neste exemplo, a URL ‘*/about*’ será associada à função ‘*about*’, que retorna o texto “Sobre nós”.

VARIÁVEIS DE ROTA

Podemos capturar partes da URL como variáveis:

```
app.py

@app.route('/user/<username>') # Define a rota com uma variável de URL <username>
def show_user_profile(username):
    return f'Perfil do usuário: {username}' # Função que retorna o perfil do usuário
```

Aqui, *'username'* será passado para a função como um argumento.

ROTEAMENTO COM MÉTODOS HTTP

Você pode especificar os métodos HTTP que uma rota aceita usando o argumento *'methods'*:

```
app.py

from flask import request # Importa o objeto request

@app.route('/login', methods=['GET', 'POST']) # Define a rota que aceita métodos GET e POST
def login():
    if request.method == 'POST':
        # Processa o formulário de login
        return 'Processando Login'
    return 'Página de Login'
```

04

TRABALHANDO COM TEMPLATES

Templates permitem que você separe a lógica da aplicação da apresentação. Flask usa o motor de templates Jinja2 para renderizar HTML.

RENDERIZANDO HTML COM RENDER_TEMPLATE

Flask facilita a renderização de templates HTML:

```
app.py

from flask import render_template # Importa a função render_template

@app.route('/hello/<name>') # Define a rota com uma variável de URL <name>
def hello(name):
    return render_template('hello.html', name=name) # Renderiza o template hello.html com a variável name
```

E no arquivo *'hello.html'*:

```
hello.html

<!doctype html>
<html>
  <head><title>Hello</title></head>
  <body>
    <h1>Hello, {{ name }}!</h1> ←!— Exibe a variável name passada pelo template —→
  </body>
</html>
```

A URL ficará dessa forma:

http://127.0.0.1:5000/hello/seu_nome, onde “seu_nome” será o nome que você colocará e será exibido na tela do navegador

INCLUINDO LÓGICA NOS TEMPLATES

Você pode adicionar lógica simples dentro dos templates:

```
hello.html

{% if name %}  <!-- Condicional que verifica se a variável name está definida -->
    <h1>Hello, {{ name }}!</h1>
{% else %}
    <h1>Hello, Stranger!</h1>
{% endif %}
```

INCLUINDO TEMPLATES

Templates podem ser incluídos uns nos outros para evitar repetição de código:

```
hello.html

<!-- layout.html -->
<!doctype html>
<html>
  <head>
    <title>{% block title %}My Website{% endblock %}</title> <!-- Bloco de título que pode ser sobrescrito -->
  </head>
  <body>
    {% block content %}{% endblock %} <!-- Bloco de conteúdo que pode ser sobrescrito -->
  </body>
</html>
```

```
hello.html

<!-- page.html -->
{% extends "layout.html" %} <!-- Estende o template layout.html -->

{% block title %}Page Title{% endblock %} <!-- Sobrescreve o bloco de título -->

{% block content %}
  <h1>Welcome to my page</h1> <!-- Sobrescreve o bloco de conteúdo -->
{% endblock %}
```

05

MANIPULAÇÃO DE FORMULÁRIOS

Manipular formulários é uma tarefa comum em aplicações web. Flask facilita o processo de recebimento e processamento de dados de formulários.

RECEBENDO DADOS DE FORMULÁRIOS

Para manipular formulários, utilizamos o método ‘*POST*’:

```
app.py

from flask import request # Importa o objeto request

@app.route('/login', methods=['GET', 'POST']) # Define a rota que aceita métodos GET e POST
def login():
    if request.method == 'POST':
        username = request.form['username'] # Obtém o valor do campo username do formulário
        return f'Bem-vindo, {username}!' # Retorna uma mensagem de boas-vindas
    return render_template('login.html') # Renderiza o template login.html
```

E no arquivo ‘*login.html*’:

```
login.html

<!doctype html>
<html>
  <body>
    <form method="post"> <!-- Formulário que envia dados usando o método POST -->
      <label for="username">Username:</label>
      <input type="text" name="username" id="username"> <!-- Campo de texto para o username -->
      <input type="submit" value="Login"> <!-- Botão de envio do formulário -->
    </form>
  </body>
</html>
```


VALIDAÇÃO DE FORMULÁRIOS

Você pode validar dados de formulários no lado do servidor:

```
app.py

@app.route('/login', methods=['GET', 'POST']) # Define a rota que aceita métodos GET e POST
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != 'admin': # Verifica se o username não é "admin"
            error = 'Usuário inválido' # Define uma mensagem de erro
        else:
            return 'Bem-vindo, admin!' # Retorna uma mensagem de boas-vindas
    return render_template('login.html', error=error) # Renderiza o template login.html com a variável error
```

E atualizar o template para exibir erros:

```
login.html

<!doctype html>
<html>
  <body>
    <form method="post"> <!-- Formulário que envia dados usando o método POST -->
      <label for="username">Username:</label>
      <input type="text" name="username" id="username"> <!-- Campo de texto para o username -->
      <input type="submit" value="Login"> <!-- Botão de envio do formulário -->
    </form>
    {% if error %} <!-- Condicional que verifica se há uma mensagem de erro -->
      <p style="color: red;">{{ error }}</p> <!-- Exibe a mensagem de erro em vermelho -->
    {% endif %}
  </body>
</html>
```

06

TRABALHANDO COM JSON

Retornar dados em formato JSON é essencial para criar APIs modernas. Flask facilita a geração de respostas JSON.

RETORNANDO JSON

Para retornar dados em formato JSON, usamos a função *'jsonify'*:

```
app.py

from flask import jsonify # Importa a função jsonify

@app.route('/api/data') # Define a rota para a URL /api/data
def get_data():
    data = {'name': 'Flask', 'type': 'Framework'} # Dados que serão retornados como JSON
    return jsonify(data) # Retorna os dados em formato JSON
```

RECEBENDO JSON

Podemos também receber dados em formato JSON:

```
app.py

@app.route('/api/echo', methods=['POST']) # Define a rota que aceita o método POST
def echo():
    data = request.get_json() # Obtém os dados JSON enviados pelo cliente
    return jsonify(data) # Retorna os dados em formato JSON
```


MANIPULANDO ERROS DE JSON

É importante lidar com erros ao receber JSON:

```
app.py

@app.route('/api/echo', methods=['POST']) # Define a rota que aceita o método POST
def echo():
    data = request.get_json()
    if not data:
        return jsonify({'error': 'Bad Request', 'message': 'No JSON data found'}), 400 # Retorna um erro 400 se os dados JSON não forem encontrados
    return jsonify(data) # Retorna os dados em formato JSON
```

07

PRÁTICAS E DICAS

Adotar melhores práticas ajuda a desenvolver aplicações Flask mais robustas e eficientes. Vamos discutir algumas dessas práticas.

ESTRUTURA DE PROJETO

Organize seu projeto em uma estrutura modular:

```
Pasta do Projeto

/seuprojeto
  /static # Arquivos estáticos (CSS, JS, imagens)
  /templates # Templates HTML
  __init__.py # Inicialização do app
  models.py # Modelos do banco de dados
  routes.py # Rotas da aplicação
  forms.py # Formulários
  config.py # Configurações da aplicação
```

CONFIGURAÇÕES DE AMBIENTE

Use diferentes configurações para desenvolvimento e produção:

```
import os

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'um_segredo_difícil_de_adivinhar'
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or 'sqlite:///site.db'

class DevelopmentConfig(Config):
    DEBUG = True

class ProductionConfig(Config):
    DEBUG = False
```


LOGS

Implemente logs para monitorar a aplicação:

```
app.py

import logging
from logging.handlers import RotatingFileHandler

if not app.debug:
    handler = RotatingFileHandler('app.log', maxBytes=10000, backupCount=1)
    handler.setLevel(logging.INFO)
    app.logger.addHandler(handler)
```



AGRADECIMENTOS

OBRIGADO POR LER!

Esse Ebook foi gerado por IA e diagramado por um humano.

O passo a passo, assim como as imagens utilizadas se encontram no meu repositório do GitHub.

<https://github.com/fsvieira122/ebook-gerado-por-ia>

Todo conteúdo foi gerado para fins didáticos de construção, foi revisado e testado.

Autor:



Fernando Soares Vieira



[Github](#)



[LinkedIn](#)