

# Introdução ao Flask

Um guia prático para entender os fundamentos do Flask, criar sua primeira aplicação web e começar a desenvolver projetos utilizando Python.

# Como usar este material

Este material foi criado para apresentar o Flask de forma prática e progressiva. Você não precisa conhecer o framework anteriormente, mas é recomendado ter familiaridade básica com Python.

Leia os capítulos na ordem para acompanhar a construção dos conceitos ou vá diretamente ao assunto que deseja revisar. Ao longo do material, procure transformar cada conceito em uma pequena aplicação ou funcionalidade.

## Conteúdo

- 01 • O que é Flask
- 02 • Como funciona uma aplicação Flask
- 03 • Criando sua primeira aplicação
- 04 • Rotas e URLs
- 05 • Templates e páginas HTML
- 06 • Recebendo dados
- 07 • Organização do projeto
- 08 • APIs e próximos passos
- 09 • Checklist da primeira aplicação

# 01 • O que é Flask

Flask é um framework web desenvolvido em Python utilizado para criar aplicações e serviços web.

Ele possui uma estrutura enxuta e permite começar com uma aplicação simples, adicionando recursos conforme o projeto cresce.

| Conceito | Explicação                                                               |
|----------|--------------------------------------------------------------------------|
| Python   | Linguagem utilizada para desenvolver a aplicação.                        |
| Flask    | Framework responsável por facilitar a criação da aplicação web.          |
| Rota     | Define o endereço e o comportamento de uma URL.                          |
| View     | Função responsável por processar uma requisição e produzir uma resposta. |
| Template | Arquivo utilizado para construir páginas HTML dinâmicas.                 |

## PRINCÍPIO

Comece entendendo o fluxo básico de uma aplicação antes de adicionar recursos mais complexos.

## 02 • Como funciona uma aplicação Flask

Uma aplicação web recebe requisições, processa informações e devolve uma resposta.

No Flask, uma requisição pode chegar a uma determinada rota, ser processada por uma função e retornar HTML, texto, JSON ou outro tipo de resposta.

| Usuário | URL | Rota | Função | Resposta |
|---------|-----|------|--------|----------|
|---------|-----|------|--------|----------|

Por exemplo, ao acessar **/clientes**, o Flask identifica a rota, executa a função correspondente, consulta ou processa os dados necessários e retorna uma resposta.

### IMPORTANTE

O Flask não determina sozinho toda a arquitetura da aplicação. O desenvolvedor pode definir como organizar módulos, banco de dados, autenticação e outros componentes.

## 03 • Criando sua primeira aplicação

Uma das vantagens do Flask é que podemos começar com uma aplicação muito pequena.

Depois de instalar o Flask, podemos criar um arquivo responsável por iniciar nossa aplicação.

```
from flask import Flask

app = Flask(__name__)

@app.route("/")
def inicio():
    return "Olá, Flask!"

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

### Nesse exemplo:

- `Flask(__name__)` cria a aplicação.
- `@app.route("/")` define a rota principal.
- `inicio()` é executada quando essa rota é acessada.
- `return` determina a resposta enviada ao navegador.

#### OBJETIVO

Entender o ciclo básico entre navegador e aplicação antes de criar um sistema completo.

## 04 • Rotas e URLs

As rotas são uma das partes fundamentais de uma aplicação Flask. Elas determinam quais endereços a aplicação reconhece e qual função será executada.

```
@app.route("/")  
  
def inicio():  
    return "Página inicial"  
  
@app.route("/sobre")  
  
def sobre():  
    return "Sobre a empresa"
```

Nesse caso, a aplicação possui duas rotas:

- / — Página inicial.
- /sobre — Página sobre.

```
@app.route("/usuario/<nome>")  
  
def usuario(nome):  
    return f"Olá, {nome}!"
```

### PRINCÍPIO

Uma rota deve representar claramente o recurso ou ação que está sendo acessada.

## 05 • Templates e páginas HTML

Aplicações reais normalmente precisam apresentar páginas HTML, e não apenas texto. O Flask utiliza o mecanismo de templates Jinja para trabalhar com páginas dinâmicas.

```
projeto/  
├── app.py  
├── templates/  
└── index.html
```

```
from flask import render_template  
  
@app.route("/")  
def inicio():  
    return render_template("index.html")
```

Dessa forma, a aplicação deixa de retornar apenas uma mensagem e passa a renderizar um documento HTML.

### FLUXO

Aplicação → Dados → Template → HTML → Navegador

## 06 • Recebendo dados

Aplicações web frequentemente precisam receber informações dos usuários. Formulários são um exemplo comum.

Um formulário pode enviar dados para a aplicação por meio de uma requisição HTTP.

| GET                                                            | POST                                                        |
|----------------------------------------------------------------|-------------------------------------------------------------|
| Normalmente utilizado para solicitar ou consultar informações. | Normalmente utilizado para enviar dados para processamento. |

```
from flask import request

@app.route("/cadastro", methods=["POST"])
def cadastro():
    nome = request.form["nome"]
    return f"Cadastro recebido: {nome}"
```

Antes de armazenar ou utilizar dados recebidos, é importante validar as informações.

- Validar campos obrigatórios.
- Verificar formatos.
- Tratar valores inesperados.
- Retornar mensagens claras.

## 07 • Organização do projeto

Projetos pequenos podem começar com poucos arquivos. Conforme a aplicação cresce, separar responsabilidades ajuda a manter o código compreensível.

```
projeto/  
  
■■■ app.py  
  
■■■ templates/  
  
■ ■■■ index.html  
  
■■■ static/  
  
■ ■■■ css/  
  
■ ■■■ js/  
  
■■■ requirements.txt
```

**app.py** — Inicialização e configuração inicial da aplicação.

**templates/** — Páginas HTML utilizadas pela aplicação.

**static/** — Arquivos estáticos, como CSS, JavaScript e imagens.

**requirements.txt** — Dependências Python utilizadas pelo projeto.

### PRINCÍPIO

Organize o projeto para que outra pessoa consiga entender onde cada responsabilidade está localizada.

## 08 • APIs e próximos passos

O Flask também pode ser utilizado para desenvolver APIs.

Em uma API, em vez de retornar uma página HTML, uma rota pode fornecer dados para outras aplicações. Um formato comum para troca de informações é o JSON.

```
from flask import jsonify

@app.route("/api/status")

def status():

    return jsonify({

        "status": "online"

    })
```

### Alguns caminhos naturais de estudo:

- Métodos HTTP e JSON.
- APIs REST.
- Banco de dados.
- Autenticação e validação.
- Blueprints e variáveis de ambiente.
- Testes e deploy.

#### PRÓXIMO PASSO

Escolha um pequeno projeto para praticar. O objetivo não é aprender todo o Flask de uma vez, mas transformar conceitos em funcionalidades reais.

## 09 • Checklist da primeira aplicação

Use esta lista para revisar os fundamentos antes de avançar para projetos maiores.

- ✓ Python instalado e funcionando.
- ✓ Flask instalado no ambiente do projeto.
- ✓ Aplicação Flask criada.
- ✓ Rota principal configurada.
- ✓ Servidor local funcionando.
- ✓ Pelo menos uma página HTML criada.
- ✓ Templates organizados.
- ✓ Arquivos estáticos configurados.
- ✓ Dados recebidos tratados corretamente.
- ✓ Projeto dividido em arquivos de forma organizada.
- ✓ Dependências registradas.
- ✓ Aplicação testada localmente.

### Projeto para praticar

- Lista de tarefas
- Cadastro de contatos
- Sistema simples de produtos
- API de estudos
- Projeto com banco de dados

#### PRÓXIMO PASSO

O objetivo não é aprender todo o Flask de uma vez. É entender os fundamentos, praticar e evoluir a aplicação conforme novos conceitos forem aprendidos.

Flask é apenas o começo. O mais importante é aprender a transformar conceitos em aplicações reais.