Instalação

```
pip install Flask
```

Início

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    print("Hello print")
    return "Hello return"
```

- primeiro importa-se a classe Flask.
- cria-se uma instância da classe Flask contendo o nome do pacote ou biblioteca da aplicação (comumente “__name__”).
- Essa instância representa uma aplicação WSGI. Flask utiliza isso para saber aonde procurar por recursos como templates, imagens, html, etc.
- Cria-se rotas utilizando o decorador route, para indicar ao flask qual url vai iniciar qual função.
- Defini-se uma função indicando parametros a serem recebidos, processamentos a serem realizados, e resultados a serem retornados.
- print vs return
- para iniciar a aplicação utiliza-se o comando abaixo

```
flask --app arquivo.py run
```

```
# caso o nome do arquivo seja app.py ou wsgi.py
# basta rodar flask run
flask run
```

```
# outra opção também é criar uma variável de ambiente
# FLASK_APP=aquivo.py
# depois basta rodar flask run
export FLASK_APP=arquivo.py
flask run
```

```
# Resultado esperado
* Serving Flask app 'hello.py'
* Debug mode: off

WARNING: This is a development server.
Do not use it in a production deployment.
Use a production WSGI server instead.

 * Running on http://127.0.0.1:5000

Press CTRL+C to quit
```

- Quando especificado a variável de ambiente `FLASK_APP` , flask indicará na primeira linha o nome do arquivo que está sendo executado.
- O debug mode vem desligado por padrão.
- **debug mode:** servidor automaticamente muda após cada atualização no código. E mostra um debbuger interativo em caso de erro, permitindo executar código python do navegador.
    - Apesar de ser protegido por um PIN, ainda é uma falha de segurança. Então não rode servidores com modo de debug ativado em produção.
    - Para habilitar o modo debug basta adicionar a tag `--debug` ao comando `flask` .

```
flask --app arquivo.py --debug run
ou
flask --debug run
```

- **Visibilidade**:
    - **local:** o servidor vai estar visível apenas para a máquina local.
    - **global:** para habilitar a visibilidade do servidor para a rede basta adicionar `--host=0.0.0.0` ao comando `flask run` . Isso fala para o systema escutar todos os IP públicos

Rotas

- simples
- com parâmetros
    - Tipagem
        - string: (padrão) aceita textos sem barras (”/”)
        - int: inteiros positivos
        - float: decimais positivos
        - path: similar a string porém aceita barras (“/”)

```
@app.route("/<parametro>")
def hello_name(rota/<parametro>):
    return f"Hello, parametro!"
```

- Listar todas as rotas

```
flask --app arquivo.py routes
```

- **HTML Escaping:** um ataque comum é a utilização de script em páginas htmls, para se proteger desse tipo de ataque recomenda-se que as rotas que utilizam parâmetros abertos realizem “escape” de marcações html. Para isso utilizamos a função `escape` do pacote markupsage. Segue exemplo:

```
from markupsafe import escape

@app.route("/<name>")
def hello_name(name):
    return f"Hello, {escape(name)}!"
```

- final de rota
    - **com barra(@app.route("/caminho/")):** Funciona como se fosse uma pasta, onde há outras rotas a serem acessadas. Se definir com a barra no final e tentar acessar na URL sem “/caminho”, o navegador te redireciona para “/caminho/”

- **sem barra(@app.route("/caminho")):** caso o caminho seja definido sem barra, ele funciona como se fosse um arquivo. Se tentar acessar a url com a barra “/caminho/”, o navegador retornará página não encontrada (erro 404).

Funções

url_for(’nome_funcao’, parametros, …): recebe o nome de uma função e seus argumentos, caso seja adicionado algum parâmetro desconhecido ele é adicionado ao final da url.

render_template(’arquivo_dentro_de_templates’): renderizará o arquivo html aplicando jinja2.

Pastas

- **static:** armazenar os arquivo que não serão renderizados utilizando Jinja, como HTML estáticos, CSS, e Javascripts.
- **templates:** armazenar os arquivos que serão renderizados pelo Jinja2.

Jinja2

{{ Variáveis }}

{%if%}…{%elif%}…{%else%}…{%endif%}

{%for%}…{%endfor%}

Requests

- requests é um objeto que pode ser importado do flask para utilização dos itens da requisição.
- resquests.args é uma estrutura de dados utilizada para armazenar os argumentos enviados pelo método get (passados juntos a url após o “?”).
- É possível transformar a estrutura requests.args em dicionário utilizando o comando `request.args.to_dict()`