

Flask

Desenvolvendo aplicações Web com o Framework Flask

Flask

Introdução

Flask é um microframework para Python baseado em Werkzeug, Jinja 2 e boas intenções. E, antes que pergunte, sua licença é baseada na licença open source BSD.

Flask é divertido.

*Tradução livre, do site oficial.

O site oficial é feito em flask:

<http://flask.pocoo.org>



[https://pt.wikipedia.org/wiki/Licença BSD](https://pt.wikipedia.org/wiki/Licença_BSD)

Flask

Microframework

Flask é definido como microframework porque possui o núcleo básico para que você possa estruturar sua aplicação *web* como desejar, fazendo com que sua aplicação possa crescer sem problemas.

Podemos começar com uma aplicação pequena, com um *app* feito em um único arquivo e ir crescendo aos poucos até ter os módulos bem estruturados, permitindo a escalabilidade.

Por padrão, o *Flask* não inclui uma camada de abstração de banco de dados, validação de formulário ou qualquer outra implementação para funcionalidades que são fornecidas por bibliotecas existentes. Em vez de implementar essas funcionalidades, o *Flask* suporta extensões para adicionar funcionalidades à sua aplicação, como se realmente estivessem implementadas no *Flask*.

Flask

Jinja 2

Jinja 2 é uma biblioteca para *Python* que foi projetada para ser flexível, rápida e segura. É um *template engine* (motor de template) escrito em *Python*. Os templates criados são escritos utilizando marcações e o *Jinja2* renderiza estes *templates*.

```
<title>{% block title %}{% endblock %}</title>
<ul>
{% for user in users %}
  <li><a href="{{ user.url }}">{{ user.username }}</a></li>
{% endfor %}
</ul>
```

O código interpretado pelo *Jinja2* dever ser iniciado com os caracteres “{%” e finalizado com os caracteres “%}”. Uma variável é definida com chaves duplas {{variável}}.

WerkZeug e WSGI

WerkZeug (Alemão, significa ferramentas) é uma biblioteca (conjunto de ferramentas) para desenvolvimento de aplicações *WSGI*. Possuindo a implementação básica deste padrão para *interceptor requests* e para lidar com *response*, controle de cache, *cookies*, *status HTTP*, roteamento de *urls* e também conta com uma poderosa ferramenta de *debug*.

WSGI ou *Web Server Gateway Interface* (Interface de Porta de Entrada do Servidor *Web*) é uma especificação para uma interface simples e universal entre servidores *web* e aplicações *web* ou *frameworks* para a linguagem de programação *Python*. Esta especificação foi adotada como padrão para desenvolvimento de aplicações *web* em *Python*.

Flask

Boas intenções

Ser baseado em boas intenções significa dizer que o Flask é Pythônico, seguindo as premissas do ZEN do Python.

Entre no interpretador do

Python e digite:

`import this`

Você verá o Zen do Python

que foi escrito por Tim Peters

```
Command Prompt - python
C:\Users\evaldo>python
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.   Although that way may not be o
bvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

Boas intenções

The Zen of Python Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced.	In the face of ambiguity, refuse the temptation to guess. There should be one-- and preferably only one --obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than *right* now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea -- let's do more of those!	O Zen do Python Bonito é melhor que feio. Explícito é melhor que implícito. Simples é melhor que complexo. Complexo é melhor que complicado. Plano é melhor que aninhado. Esparsos é melhor que denso. Legibilidade conta. Casos especiais não são especiais o bastante para se quebrar as regras. Embora a simplicidade supere o purismo. Erros nunca deveriam passar silenciosamente. A menos que explicitamente silenciados.	Ao encarar a ambiguidade, recuse a tentação de adivinhar. Deveria haver uma – e preferencialmente apenas uma – maneira óbvia de se fazer isto. Embora aquela maneira possa não ser óbvia à primeira vista se você não for holandês. Agora é melhor que nunca. Embora nunca, seja muitas vezes melhor que pra já. Se a implementação é difícil de explicar, é uma má idéia. Se a implementação é fácil de explicar, pode ser uma boa idéia. Namespaces são uma idéia estupenda – vamos fazer mais deles!
---	---	---	--

Flask

Instalando o Flask

```
Command Prompt
C:\Users\evaldo>pip install Flask
Collecting Flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
    100% |████████████████████████████████████████| 92kB 499kB/s
Requirement already satisfied: Jinja2>=2.4 in c:\evaldo\ferramentasdesenvolvimento\python\python36\lib\site-packages (from Flask)
Collecting click>=2.0 (from Flask)
  Downloading click-6.7-py2.py3-none-any.whl (71kB)
    100% |████████████████████████████████████████| 71kB 2.2MB/s
Collecting itsdangerous>=0.21 (from Flask)
  Downloading itsdangerous-0.24.tar.gz (46kB)
    100% |████████████████████████████████████████| 51kB 1.4MB/s
Collecting Werkzeug>=0.7 (from Flask)
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (327kB)
    100% |████████████████████████████████████████| 327kB 905kB/s
Requirement already satisfied: MarkupSafe>=0.23 in c:\evaldo\ferramentasdesenvolvimento\python\python36\lib\site-packages (from Flask)
Installing collected packages: click, itsdangerous, Werkzeug
Running setup.py install for itsdangerous ... done
```

```
evaldowolkers@evaldo ~ $ sudo pip3 install Flask
[sudo] password for evaldowolkers:
The directory '/home/evaldowolkers/.cache/pip/http' or its parent directory is
not owned by the current user and the cache has been disabled. Please check t
he permissions and owner of that directory. If executing pip with sudo, you ma
y want sudo's -H flag.
The directory '/home/evaldowolkers/.cache/pip' or its parent directory is not
owned by the current user and caching wheels has been disabled. check the perm
issions and owner of that directory. If executing pip with sudo, you may want
sudo's -H flag.
Collecting Flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
    100% |████████████████████████████████████████| 92kB 1.2MB/s
Collecting itsdangerous>=0.21 (from Flask)
  Downloading itsdangerous-0.24.tar.gz (46kB)
    100% |████████████████████████████████████████| 51kB 5.1MB/s
Collecting Werkzeug>=0.7 (from Flask)
  Downloading Werkzeug-0.14.1-py2.py3-none-any.whl (322kB)
```


Flask

Exemplo com WerkZeug usando localhost

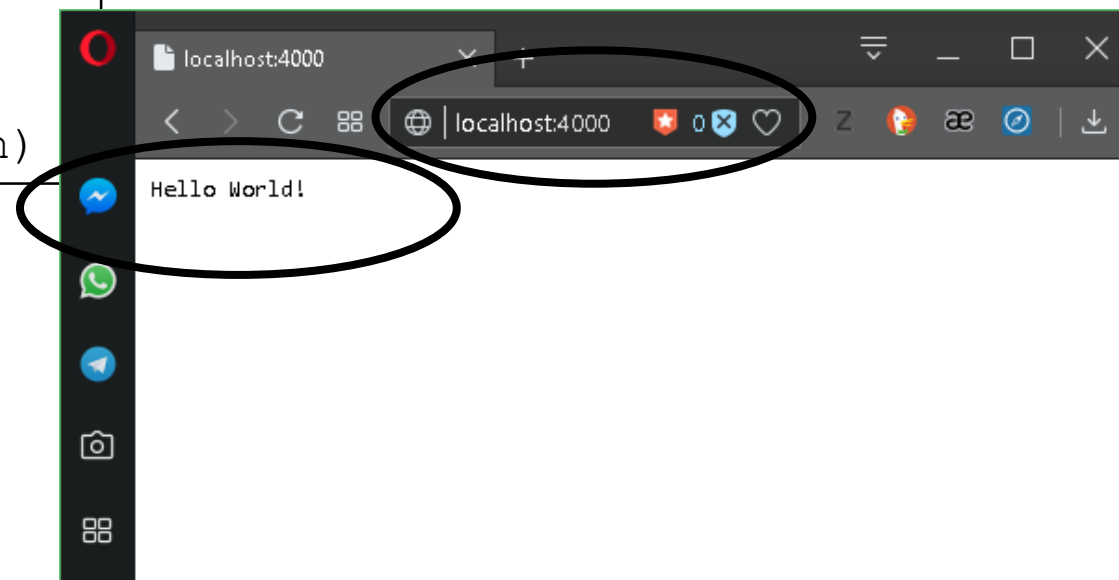
```
from werkzeug.wrappers import Request, Response

@Request.application
def application(request):
    return Response('Hello World!')

if __name__ == '__main__':
    from werkzeug.serving import run_simple
    run_simple('localhost', 4000, application)
```

Ainda não usamos o Flask.

O servidor foi iniciado usando localhost, que seria o mesmo que usar 127.0.0.1



Flask

Exemplo com Werkzeug usando o IP do Host

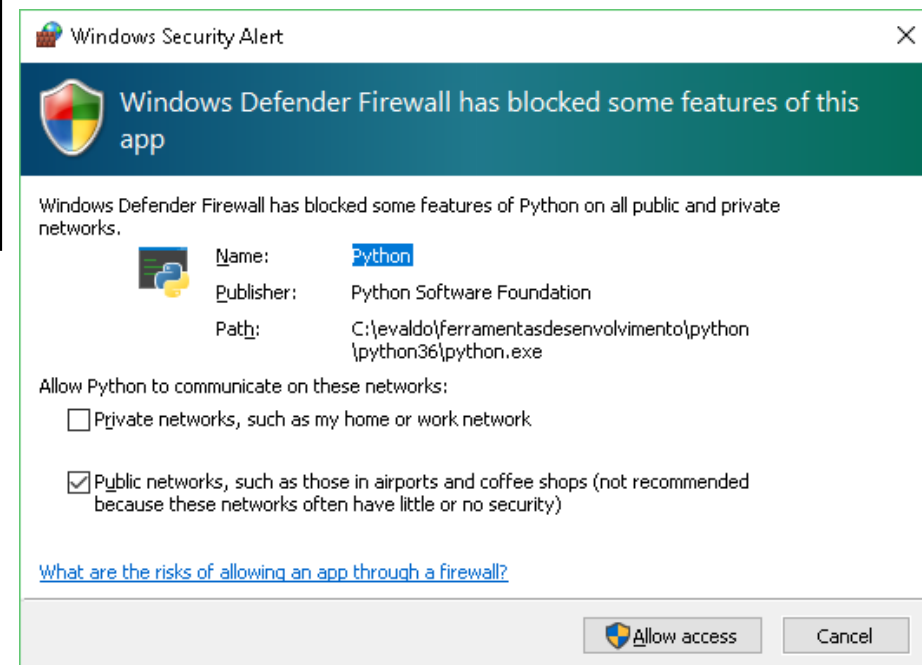
```
from werkzeug.wrappers import Request, Response

@Request.application
def application(request):
    return Response('Hello World!')

if __name__ == '__main__':
    from werkzeug.serving import run_simple
    run_simple('192.168.0.130', 4000, application)
```

Observação:

Se você for subir o servidor *web* informando o endereço *IP* do seu computador, se estiver usando o *Windows*, pode ser que um *Firewall* solicite permissão de acesso à porta informada (no exemplo porta 4000).



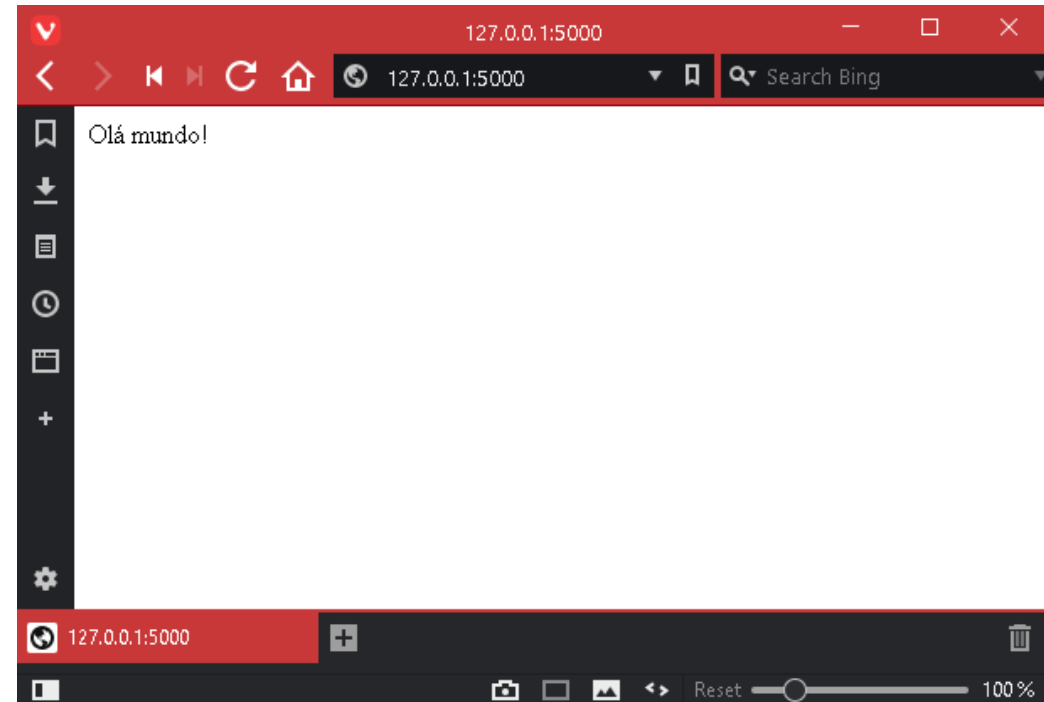
Flask

Primeira aplicação Flask – Aplicação mínima

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

@app.route("/")
def ola():
    return u'Olá mundo!'

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



```
C:\Evaldo\FerramentasDesenvolvimento\Python\Python36\python.exe C:/Temp/flask_aula/exemplo2.py
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

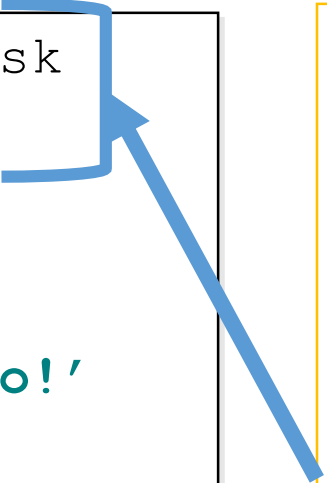
Flask

Primeira aplicação Flask – Aplicação mínima

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

@app.route("/")
def ola():
    return u'Olá mundo!'

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



Importamos a classe base do Flask e criamos uma instância chamada app.

O app é a aplicação WSGI que deverá ser passada para o servidor de aplicação que a estiver servindo.

O primeiro parâmetro recebido por Flask() deve ser o nome do pacote onde o app está definido (import_name). A variável “__name__” pega este valor dinamicamente, porém, não é uma boa prática usar assim. O objetivo do primeiro parâmetro (veremos que existem mais parâmetros) é dar ao Flask uma ideia do que pertence à sua aplicação. Este nome é usado para encontrar recursos no sistema de arquivos, pode ser usado por extensões para melhorar a informação de depuração e muito mais.

Se seu projeto está em “exemplo/app.py”, você pode informar:
app = Flask('exemplo').

“__name__” pode ser usado quando seu projeto possui um único arquivo e não pertence a um pacote.

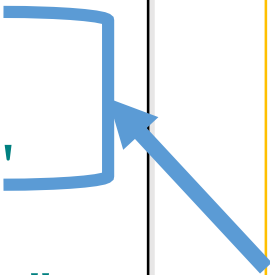
Flask

Primeira aplicação Flask – Aplicação mínima

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

@app.route("/")
def ola():
    return u'Olá mundo!'

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



Views e roteamento de URL

Views são funções ou classes que respondem por uma URL definida. A função da view é capturar os parâmetros enviados pelo cliente via URL, efetuar o processamento e responder com algum conteúdo, podendo ser texto simples (plano), texto no formato JSON, stream de dados, um template HTML renderizado, etc.

O Werkzeug abstrai boa parte deste trabalho tornando isto uma tarefa bastante simples. Ao usar `@app.route` está sendo alimentada uma lista de mapeamento implementada pelo `werkzeug.routing.Map` que contém elementos `Rule` que são as regras que ligam uma URL a uma função Python no projeto.

O decorador `route()` é usado para ligar uma função à uma URL.


Flask

Primeira aplicação Flask – Aplicação mínima

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

@app.route("/")
def ola():
    return u'Olá mundo!'

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



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

def ola():
    return u'Olá mundo!'

app.add_url_rule("/", endpoint="ola", view_func=ola)

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

Podemos definir views usando o decorator `@app.route`. Neste caso o Flask usa o nome da função automaticamente como endpoint, no exemplo acima, o endpoint para a url `/` é `ola`. E podemos utilizar também roteamento explícito principalmente se quisermos automatizar ou centralizar o mapeamento de URLs em um único local do projeto. Neste caso precisamos informar o nome do endpoint e da view:

```
app.add_url_rule("/", endpoint="ola", view_func=ola)
```

Comente a linha do `@app.route` e adicione esta linha após a definição da função `ola()` e antes do `if`.

Flask

Segunda aplicação Flask – Definindo várias rotas

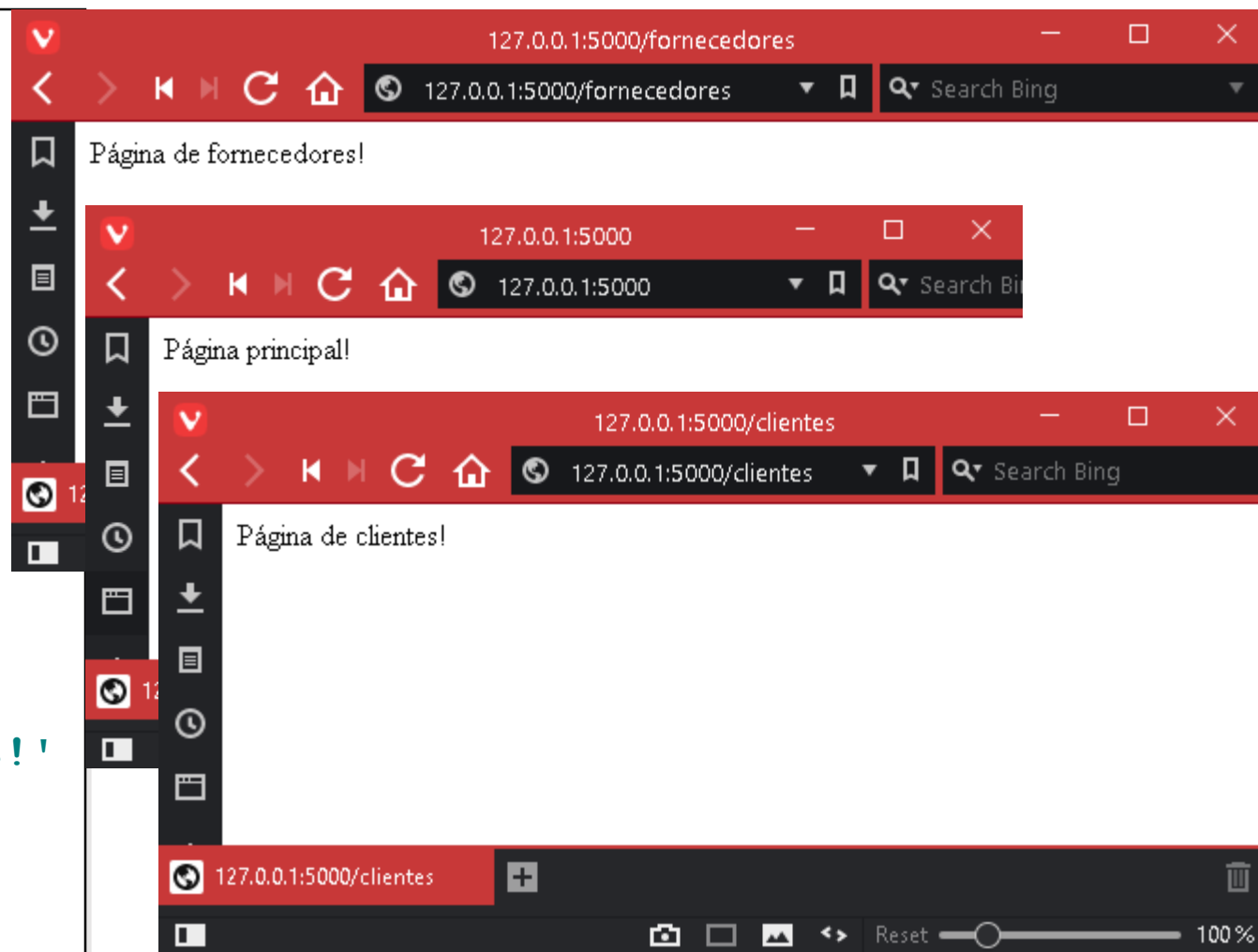
```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def index():
    return u'Página principal!'

@app.route("/clientes")
def cadastro_clientes():
    return u'Página de clientes!'

@app.route("/fornecedores")
def cadastro_fornecedores():
    return u'Página de fornecedores!'

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



Flask

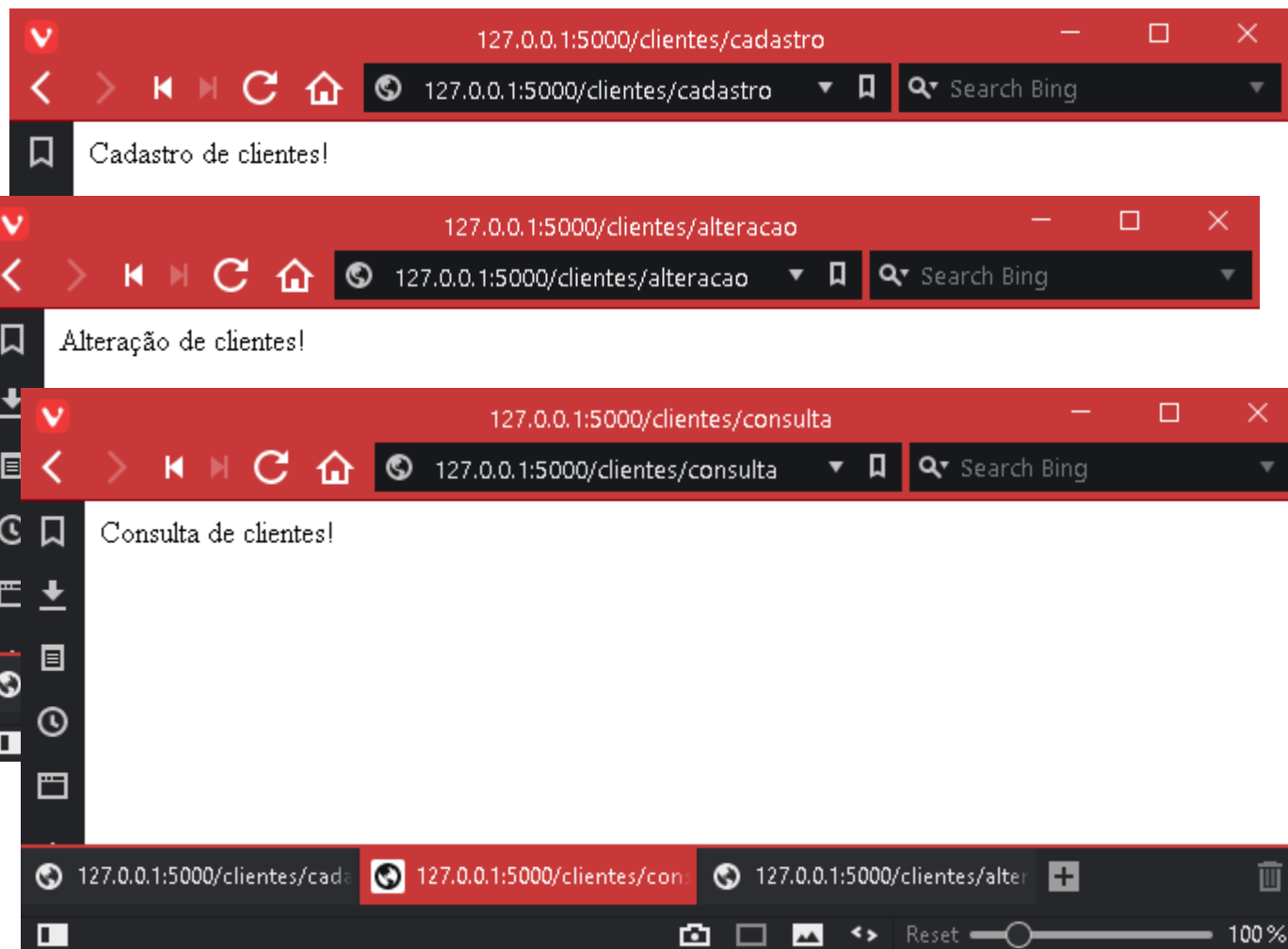
Terceira aplicação Flask – Definindo regras variáveis (regras de URL)

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

@app.route("/")
def index():
    return u'Página principal!'

@app.route("/clientes/<funcao>")
def clientes(funcao):
    if(funcao.lower()=="cadastro"):
        return u'Cadastro de clientes!'
    elif(funcao.lower()=="consulta"):
        return u'Consulta de clientes!'
    elif(funcao.lower()=="alteracao"):
        return u'Alteração de clientes!'

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



FIM