

# Tecnologias alternativas para o desenvolvimento de Aplicações Web

Módulo 1 - Experiência Criativa: Criando  
Soluções Computacionais

ANTÔNIO DAVID VINISKI  
antonio.david@pucpr.br

PUCPR

ANDREY CABRAL MEIRA  
andrey.cabral@pucpr.br

PUCPR

# Mentimeter: 3673 4435

## Disciplinas Antecedentes

<https://www.menti.com/alr72qn6n17j>

# Agenda

- Flask Framework.
- uWSGI Server.
- Templates - Jinja2
- Aplicação Web Simples
- Interações simples com o usuário
- PBL 1 – Página de Login



# Flask

# Framework



# Flask

web development,  
one drop at a time

# O que é o Flask?



- Flask é um micro framework para desenvolvimento de aplicações web escrito em Python, designado para tornar fácil e simples os primeiros passos do desenvolvimento web.
- Desenvolvido por *Armin Ronacher*, líder de um grupo internacional de entusiastas do *Python* chamado *Pocco*.
- Baseado no **Werkzeug WSGI toolkit** e na ferramenta **Jinja2** (geração de *templates web*).
- É considerada uma dos frameworks mais populares de desenvolvimento web com Python.



# Características do Flask

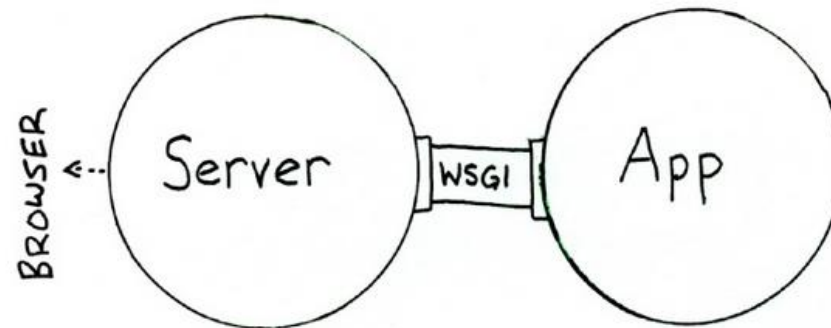


- **Flask** oferece sugestões, mas não força nenhuma dependência ou *layout* para o projeto.
- É responsabilidade do desenvolvedor escolher as ferramentas e bibliotecas que irá utilizar, como banco de dados, serviço para requisições assíncronas, bibliotecas *Javascript*, etc.
- Existe uma grande quantidade de extensões disponibilizadas pela comunidade que torna fácil a adição de novas funcionalidades a aplicação.

# O que é o Werkzeug WSGI?



- WSGI (Web Server Gateway Interface) é uma especificação de interface que permite a comunicação entre o servidor e a aplicação python.



WSGI não é um framework de desenvolvimento Web.

- Werkzeug é uma das bibliotecas de ferramentas WSGI mais avançadas para o desenvolvimento de aplicações web.
  - Implementa solicitações (*REQUESTS*), objetos de resposta (*RESPONSE OBJECTS*), dentre outras funções utilitárias.
  - É uma das bases para o funcionamento do Flask.

# O que é o Jinja2?

- **Jinja** é um mecanismo de modelagem rápido, expressivo e extensível.
- Permite a escrita de código semelhante à sintaxe do Python.
- Recebe dados da aplicação para renderizar o documento final.
- Inclui:
  - Herança e inclusão de *templates*.
  - Definição e importação de macros dentro dos *templates*.
  - Os modelos **HTML** podem usar escape automático para evitar que o XSS (*cross-site scripting*) forneça entrada não confiável do usuário.
  - Um ambiente em área restrita pode renderizar modelos não confiáveis com segurança.
  - Suporte assíncrono para a geração de modelos que lidam automaticamente com funções síncronas e assíncronas sem sintaxe extra.
  - As exceções apontam para a linha correta nos modelos para facilitar a depuração.
  - Filtros, testes, funções e até sintaxe extensíveis.





# Instalação e configuração do ambiente

- A instalação do *Flask*, *Werkzeug* e *Jinja2* é realizada em um único passo (system-wide installation):

```
> pip install Flask
```

ou

```
> python -m pip install flask
```

- ☐ Criar um diretório chamado **Módulo 1** para adicionar os exemplos implementados durante a aula.
- ☐ Criar um subdiretório chamado **exemplo00** para o exemplo clássico **Hello World!**

# Aplicação Flask



# Flask

web development,  
one drop at a time

# Aplicações Flask

- A aplicação é representada por um único objeto Flask

```
from flask import Flask
app = Flask(__name__)
## __name__ is the application name
```

- A execução do aplicativo inicia o servidor da Web (executando até você eliminá-lo).

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

# Servidor Web

- Por padrão, o Flask executa o servidor em:
  - `http://127.0.0.1:5000/`
  - Acessível pelo `localhost`, apenas.
  - Executando na porta 5000.

Pode ser customizado com os parâmetros no método `.run`:

```
if __name__ == '__main__':  
    # syntax: app.run(host=None, port=None, debug=None, **options)  
    app.run(host='0.0.0.0', port=80) # public  
    app.run(debug=True) # for development
```

# Páginas Web

- Cada página é implementada como um método.

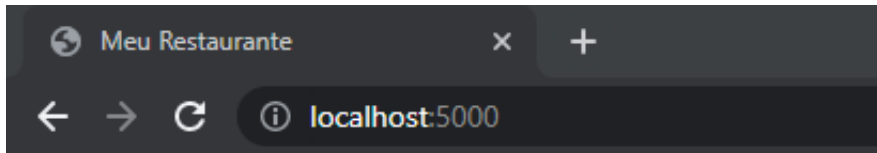
```
@app.route('/')  
def index():  
    return "Hello, web world!"
```

- Devemos especificar:
  - A URL na qual a página será visitada: '/' no *decorator* `@app.route`.
  - O nome da página: `index`.
  - O conteúdo (HTML) da página: instrução *return*.

# Exemplo 1 – Restaurante

Como solucionar?

## Página Inicial



### Meu Restaurante

Acesse o menu:

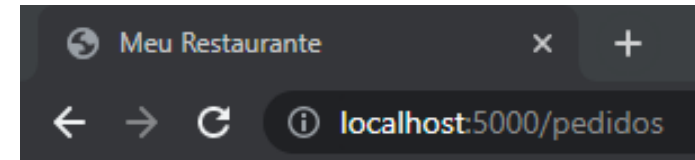
- [Listar Pedidos](#)
- [Listar Clientes](#)
- [Listar Funcionários](#)

Lembre-se de verificar corretamente os pedidos!

Caso algum produto esteja inconsistente. Solicitar a alteração!

Não realizamos a troca dos pedidos após a confeccção!

## Página Pedidos



### Pedidos

- Combo 1, comanda 2.
- Combo 2, comanda 5.
- Executivo 2, comanda 3
- Refri laranja, comanda 134
- Cerveja, comanda 12
- Batata Frita, comanda 14

Voltar para [página inicial!](#)

- Quais os componentes de uma página HTML podem estar sendo utilizados nessas duas páginas simples?

# Mentimeter: 3673 4435

- Quais são os componentes (<tags></tags>) de uma página HTML podem estar sendo utilizados nessas duas páginas simples?

`<html>...</html>`

`<head>...</head>`

`<title>...</title>`

`<body>...</body>`

`<h1>...</h1>`

```
<ul>
  <li>
  </li>
</ul>
```

`<p>...</p>`

`<div>...</div >`

`<form>...</form>`

`<h2>...</h2>`

`<a>...</a>`

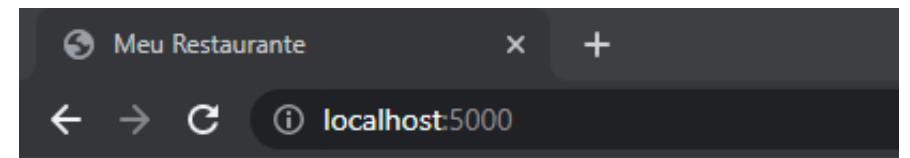
# Exemplo 1

- Implementar os métodos para gerar as páginas html.

```
@app.route('/')
def index():
    return "Conteúdo da Página Inicial!"

@app.route('/orders')
def orders():
    return "Conteúdo da página de pedidos!"
```

Lembrando que toda página é um método no Flask.



## Meu Restaurante

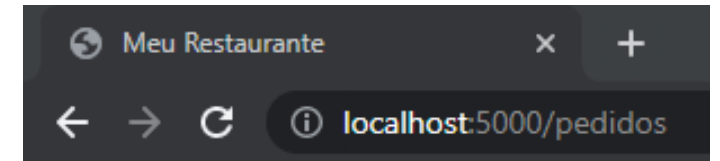
Acesse o menu:

- [Listar Pedidos](#)
- [Listar Clientes](#)
- [Listar Funcionários](#)

Lembre-se de verificar corretamente os pedidos!

Caso algum produto esteja inconsistente. Solicitar a alteração!

Não realizamos a troca dos pedidos após a confecção!



## Pedidos

- Combo 1, comanda 2.
- Combo 2, comanda 5.
- Executivo 2, comanda 3
- Refri laranja, comanda 134
- Cerveja, comanda 12
- Batata Frita, comanda 14

Voltar para [página inicial!](#)



# Exemplo 1 – Página Inicial

```
@app.route('/')
def index():
    return """<html>
        <head>
            <title>Meu Restaurante</title>
        </head>
        <body>
            <h2>Meu Restaurante</h2>
            <h3>Acesse o menu:</h3>
            <ul>
                <li><a href="/orders">Listar Pedidos</a></li>
                <li><a href="#">Listar Clientes</a></li>
                <li><a href="#">Listar Funcionários</a></li>
            </ul>
            <p>Lembre-se de verificar corretamente os pedidos!</p>
            <p>Caso algum produto esteja inconsistente. Solicitar a alteração!</p>
            <p>Não realizamos a troca dos pedidos após a confecção!</p>
        </body>
    </html>
    """
```

# Exemplo 1 – Página de Pedidos

```
@app.route('/orders')
def orders():
    return """
        <html>
            <head>
                <title>Meu Restaurante</title>
            </head>
            <body>
                <h1>Pedidos</h1>
                <ul>
                    <li>Combo 1, comanda 2.</li>
                    <li>Combo 2, comanda 5.</li>
                    <li>Executivo 2, comanda 3</li>
                    <li>Refri laranja, comanda 134</li>
                    <li>Cerveja, comanda 12</li>
                    <li>Batata Frita, comanda 14</li>
                </ul>
                <p>Voltar para <a href="/">página inicial</a>!</p>
            </body>
        </html>
    """
```

# Exercício 1

- Criar os métodos (os quais representam as páginas) para retornar a lista dos clientes e dos funcionários da aplicação do restaurante.

# Template HTML com

- Incorporar HTML em *strings* no Python é:
  - Feio;
  - Propenso a erros;
  - Complexo (ou seja, deve seguir as regras de escape do HTML e as regras de citação do Python);
  - Eu disse feio?
- Templating: separação da estrutura (fixa) do texto HTML (*template*) das partes variáveis (variáveis interpoladas).
- O *Flask* oferece suporte ao mecanismo de modelagem ***Jinja2***.

# Template básico com Jinja2

- *Templates* devem estar na subpasta `./templates`;
- *Templates* são arquivos HTML, com a extensão `.html`;
- Os *templates* são processados quando solicitados pela página Flask.

```
return render_template('orders.html')
```

# Exemplo 2 – Solução com *template* HTML

- Copiar o conteúdo do diretório `exemplo01` para o diretório `exemplo02`.
- Inserir os códigos de retorno dos métodos `index` e `orders` nos arquivos HTML.
- Ajustar os métodos que representam as páginas para que retornem a renderização dos *templates* HTML.
- Testar a aplicação.

# Exemplo 2 – Solução com *template* HTML

```
from flask import Flask, render_template

app = Flask(__name__)

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

@app.route('/orders')
def orders():
    return render_template("orders.html")

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

# Exemplo 2 – home.html

```
<!-- home.html -->
<html>
  <head>
    <title>Meu Restaurante</title>
  </head>
  <body>
    <h2>Meu Restaurante</h2>
    <h3>Acesse o menu:</h3>
    <ul>
      <li><a href="/orders">Listar Pedidos</a></li>
      <li><a href="#">Listar Clientes</a></li>
      <li><a href="#">Listar Funcionários</a></li>
    </ul>
    <p>Lembre-se de verificar corretamente os pedidos!</p>
    <p>Caso algum produto esteja inconsistente. Solicitar a alteração!</p>
    <p>Não realizamos a troca dos pedidos após a confeccão!</p>
  </body>
</html>
```



# Exemplo 2 – orders.html

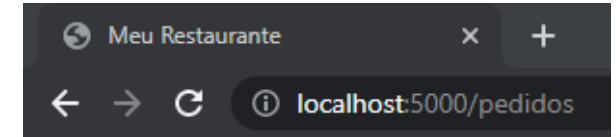
```
<!-- orders.html -->
<html>
  <head>
    <title>Meu Restaurante</title>
  </head>
  <body>
    <h1>Pedidos</h1>
    <ul>
      <li>Combo 1, comanda 2.</li>
      <li>Combo 2, comanda 5.</li>
      <li>Executivo 2, comanda 3</li>
      <li>Refri laranja, comanda 134</li>
      <li>Cerveja, comanda 12</li>
      <li>Batata Frita, comanda 14</li>
    </ul>
    <p>Voltar para <a href="/">página inicial</a>!</p>
  </body>
</html>
```

# Jinja2 - Utilizando a passagem de parâmetros para a página HTML

# Parâmetros no HTML com Jinja2

- Considere a página de pedidos(**orders.html**.)
  - E se os pedidos estivessem armazenados em uma lista de *strings*?

```
pedidos = ["Combo 1, comanda 2","Executivo 2, comanda 3",  
           "Refri laranja, comanda 134","Cerveja, comanda 12",  
           "Batata Frita, comanda 14"]
```



## Pedidos

- Combo 1, comanda 2.
- Combo 2, comanda 5.
- Executivo 2, comanda 3
- Refri laranja, comanda 134
- Cerveja, comanda 12
- Batata Frita, comanda 14

Voltar para [página inicial!](#)

- O Jinja2 permite a passagem de parâmetros para a página HTML e a leitura dos valores por meio de estruturas condicionais e de repetição

```
return render_template('orders.html', orders = pedidos)
```

# Declarações vs Expressões

- Templates podem interpolar os valores passados como parâmetro:
  - `{{ parameter }}`
  - `{{ expression }}`
- Os templates podem incluir instruções de programação:
  - `{% statement %}`
- Os templates podem acessar alguns objetos implícitos:
  - `request, session;`
- Uma `{% declaracao %}` controla o fluxo de execução em um *template*.
- Uma `{{ expressão }}` avalia o valor de uma variável (ou expressão) e imprime o resultado no arquivo HTML.
  - <https://jinja.palletsprojects.com/en/3.0.x/templates/>



# Principais declarações do Jinja2

- Estrutura de repetição:

```
{% for var in list %}  
    ...  
{% endfor %}
```



- Estruturas condicionais:

```
{% if condition %} ...  
{% elif cond %} ...  
{% else %} ...  
{% endif %}
```



# Exemplo 3

- Copiar o conteúdo do diretório `exemplo02` para o diretório `exemplo03`.
- Modificar o exemplo 2, ajustando o método `orders` para que retorne para a página HTML a lista de temas.
- Utilizar as expressões e declarações para imprimir a lista no HTML da página Aula 1 (`orders.html`).

# Exemplo 3 – Declarações e Expressões

```
from flask import Flask, render_template


app = Flask(__name__)

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

@app.route('/orders')
def orders():
    pedidos = ["Combo 1, comanda 2","Executivo 2, comanda 3",
               "Refri laranja, comanda 134","Cerveja, comanda 12",
               "Batata Frita, comanda 14"]

    return render_template("orders.html", orders = pedidos)

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



Precisamos definir o nome do parâmetro  
que será requisitado pela página HTML

# Exemplo 3 – pedidos.html

```
<!-- pedidos.html -->
<html>
  <head>
    <title>Meu Restaurante</title>
  </head>
  <body>
    <h1>Pedidos</h1>
    <ul>
      {% for order in orders %}
      <li>{{ order }}</li>
      {% endfor %}
    </ul>
    <p>Voltar para <a href="/">página inicial</a>!</p>
  </body>
</html>
```



# Herança de *template* com Jinja2

- A parte mais poderosa do Jinja2 é a herança de *template*.
- A herança permite que você crie um modelo básico de “esqueleto” que contém todos os elementos comuns do seu site e define blocos que os modelos “filhos” podem substituir.
- Exemple
  - Menu;
  - Rodapé;
  - Estrutura e organização dos componentes;
  - Criação de módulos específicos para as páginas que compartilham características em comum.

# Criação de um template base

- O primeiro passo para utilizar a herança de *template* do Jinja2 é criar o *template* base.
- É este *template* que vai definir toda a estrutura base que os outros *templates* irão herdar.
- Para que seja possível criar uma estrutura que permita a inserção do conteúdo das páginas filhas na estrutura do arquivo base, o Jinja2 possibilita a criação de blocos de conteúdos com a (tag `block`).

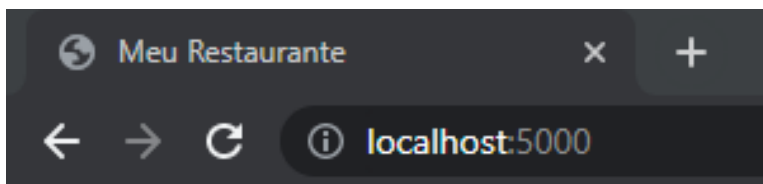
```
{% block blockname %}{% endblock %}
```

- Para que a página filha herde o template base, precisa adicionar o seguinte comando no início da página:

```
{% extends "base.html" %}
```

# Exemplo 4 – Herança de Template

- Copiar o conteúdo do diretório exemplo03 para o diretório exemplo04.
- Criar um arquivo chamado `base.html` na pasta `templates`.
- Manter as seguintes informações no arquivo base:



## Meu Restaurante

Acesse o menu:

- [Listar Pedidos](#)
- [Listar Clientes](#)
- [Listar Funcionários](#)

- Criar um bloco para adicionar os conteúdos específicos das páginas.
- Entender o arquivo base nas páginas.
- Inserir o conteúdo de cada uma delas no bloco de conteúdo.

# Exemplo 4 – Herança de Template I

## ○ Conteúdo do arquivo base.html

```
<!-- base.html -->
<html>
  <head>
    <title>Meu Restaurante</title>
  </head>
  <body>
    <h2>Meu Restaurante</h2>
    <h3>Acesse o menu:</h3>
    <ul>
      <li><a href="pedidos">Listar Pedidos</a></li>
      <li><a href="#">Listar Clientes</a></li>
      <li><a href="#">Listar Funcionários</a></li>
    </ul>
    <div id="content">
      {% block content %} {% endblock %}
    </div>
  </body>
</html>
```

# Exemplo 4 – Herança de Template II

- Conteúdo do arquivo home.html

```
<!-- home.html -->
{% extends "base.html" %}
{% block content %}
    <p>Lembre-se de verificar corretamente os pedidos!</p>
    <p>Caso algum produto esteja inconsistente. Solicitar a alteração!</p>
    <p>Não realizamos a troca dos pedidos após a confeccão!</p>
{% endblock %}
```

# Exemplo 4 – Herança de Template III

## ○ Conteúdo do arquivo orders.html

```
<!-- pedidos.html -->
{% extends "base.html" %}
{% block content %}
    <h1>Pedidos</h1>
    <ul>
        {% for order in orders %}
        <li>{{ order }}</li>
        {% endfor %}
    </ul>
    <p>Voltar para <a href="/">página inicial</a>!</p>
{% endblock %}
```

# PBL – Portifólio de Aprendizagem

# Portifólio de Aprendizagem - Individual

- Criar uma aplicação web para armazenar o seu portfólio de aprendizagem.
- O portfólio deve conter informações de no mínimo de três disciplinas.
  - As informações de cada disciplina podem ser agrupadas por módulos, aulas, RA, etc.
  - Podem ser armazenados os temas principais, anotações, agenda de cada aula.
  - Também podem ser armazenados arquivos de texto, imagens, códigos fonte, etc.
- O portfólio é individual e será aprimorado com base nos temas de estudo apresentados no decorrer da disciplina.
  - Cada estudante irá armazenar as suas próprias informações.



# Referências

- Documentação do Flask:
  - <https://flask.palletsprojects.com/en/2.2.x/>
- Documentação do Jinja2.
  - <https://jinja.palletsprojects.com/en/3.1.x/>
- Documentação WSGI.
  - <http://wsgi.tutorial.codepoint.net/intro>

