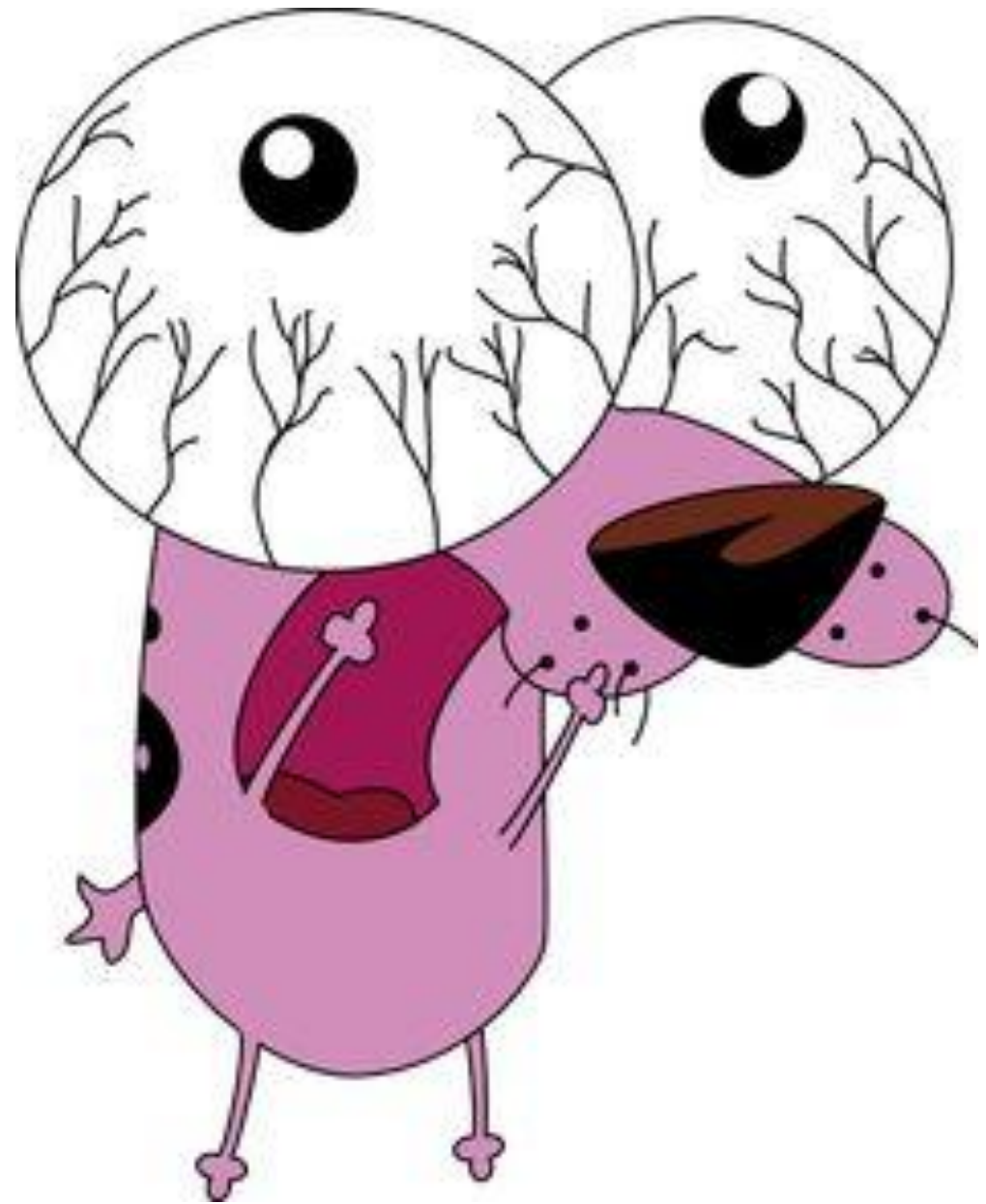


Desenvolvimento Back-end com Flask

Rodrigo Attique
Desenvolvedor de Sistemas

O que é Back-end?



Back-end

- Traduzindo para o português quer dizer, parte de traz.
- Em termos práticos é a camada de sistema mais próxima dos dados.
- Um back-end é sempre responsável por manipular os dados no banco e exibí-los através de um front.

```
settings.py  models.py  admin.py  launch
registro > models.py > ...
5  class Turma(models.Model):
6      ano = models.IntegerField()
9
10     def __str__(self):
11         return f"{self.codigo} - {self.nome}"
12
13     class UnidadeCurricular(models.Model):
14         nome = models.CharField(max_length=100)
15         descricao = models.TextField()
16
17         def __str__(self):
18             return self.nome
19
```

Como funciona?



No geral back-ends são escritos em linguagens de programação.



Como Python, Java, PHP, C#

[Esta Foto](#) de Autor Desconhecido está licenciado em [CC BY](#)



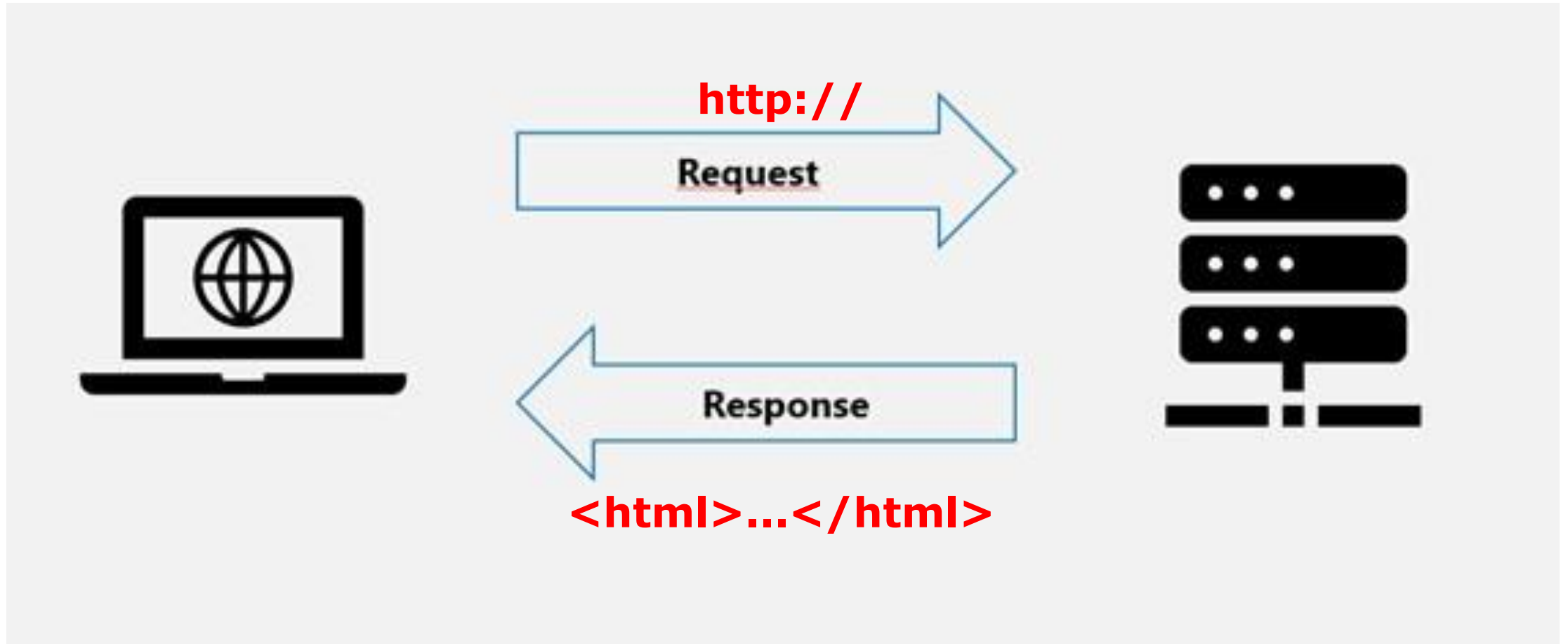
Essas linguagens são responsáveis por “gerar” o HTML que será exibido.

Front-end Vs Back-end

- Enquanto o back-end é responsável pela camada de dados.
- O front-end é o responsável pela apresentação ao usuário.
- Em resumo o enquanto o back-end usa uma linguagem de programação.
- Font-end usa linguagem de marcação HTML e JavaScript.

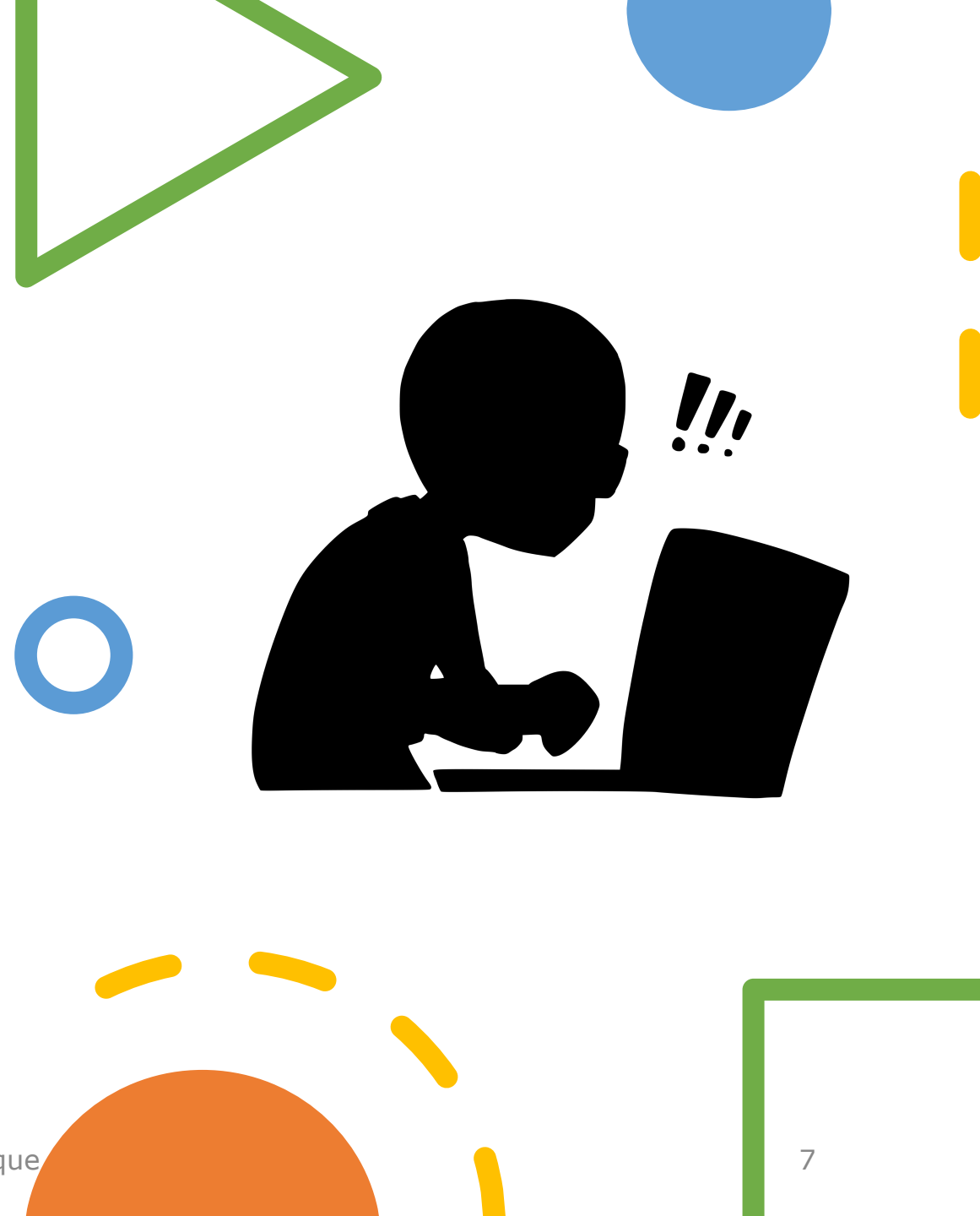


Modelo Cliente Servidor



Como funciona?

- **O cliente (navegador) faz uma requisição.**
- **O back-end pega os dados do banco de dados e gera o HTML no servidor.**
- **Que é enviado através da resposta ao navegador.**
- **Que interpreta as tags HTML com os dados exibindo o site ao usuário.**



Mas e o Flask?

- **O Flask é o que chamamos de framework.**
- **Frameworks são “molduras” de código que nos permitem criar aplicativos seguindo os padrões estabelecidos por ele.**
- **Para o desenvolvimento WEB o uso de frameworks é bastante comum e podemos dividi-los em duas categorias.**
- **Full-Stacks**
- **Microframeworks**



Full-Stack

- Ou pilha completa.
- São frameworks que carregam **bibliotecas** para todas as operações.
- Banco de dados
- Templates
- ETC....
- No geral é necessário somente instalar e usar o que ele oferece.



Microframeworks

- Já os microframeworks são responsáveis apenas por gerenciar as rotas.
- Com todo o resto devendo ser instalado pelo desenvolvedor.
- No geral é necessário mais conhecimento para usá-los.
- Porém dão muito mais flexibilidade ao projeto.



Bibliotecas

- São trechos de código pronto que usamos para diversos fins.
 - **Como manipulação de data e hora.**
 - **Conexão com banco de dados**
 - **Manipulação de dados e gráficos.**
 - **Etc.**



Mas e o Flask?

- **Flask está na categoria microframework.**
- **Então teremos que instalar tudo exceto a biblioteca de templates.**
- **Flask é escrito em Python, logo, precisamos conhecer python para usá-lo.**



Eu não sei

- Python é atualmente a linguagem de programação mais simples do mundo.
- **Ela é usada por milhares de programadores e pessoas comuns.**
- **Sim, pessoas comuns que utilizam para diversas finalidades.**
- **Como automatizar tarefas, estatística e desenvolvimento.**



Configurando o Flask






- Para configurar o Flask precisamos instalar o Python e adicioná-lo ao Path.
- **Vamos pular a parte de instalar o python e partir pro que interessa.**

<https://flask.palletsprojects.com/en/3.0.x/installation/>

Iniciando o projeto



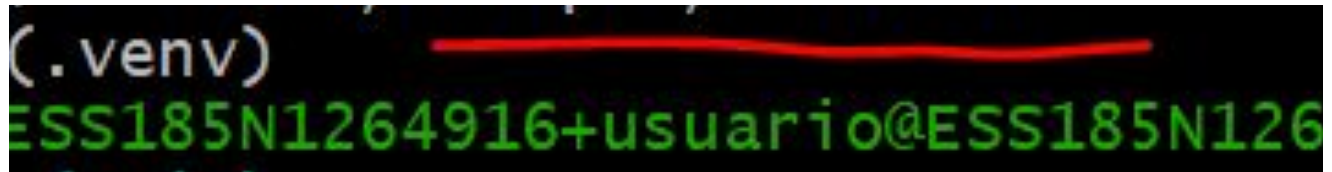
- Abra o git-bash no diretório **aulaHTML**
- **Digite os comandos.**

Nome	Data de modificação	Tipo	Tamanho
 .git	12/09/2024 13:14	Pasta de arquivos	
 exemplos	09/09/2024 13:44	Pasta de arquivos	
 static	12/09/2024 10:01	Pasta de arquivos	
 index.html	12/09/2024 10:19	Brave HTML Document	3 KB
 <u>.venv</u>	12/09/2024 14:00	Pasta de arquivos	

```
MINGW64:/g/Meu Drive/SENAI/Aulas/HTML/site
ESS185N1264916+usuario@ESS185N1264916 MINGW64 /g/Meu Drive/SENAI/Aulas/HTML/site
(main)
$ python -m venv .venv
ESS185N1264916+usuario@ESS185N1264916 MINGW64 /g/Meu Drive/SENAI/Aulas/HTML/site
(main)
$ ./venv/Scripts/activate
(.venv)
ESS185N1264916+usuario@ESS185N1264916 MINGW64 /g/Meu Drive/SENAI/Aulas/HTML/site
(main)
$ |
```


Explicando...

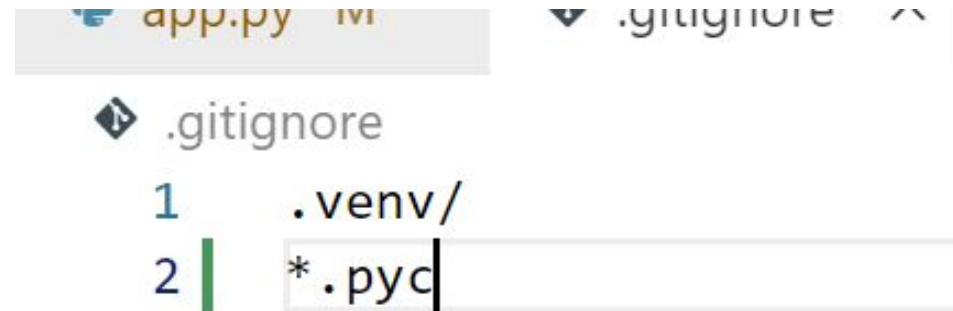
- **Primeiro precisamos criar um ambiente virtual chamado `.venv` que irá isolar nossa aplicação.**
- **Depois precisamos ativa-lo com o `.venv/scripts/activate`**
- **Por fim instalar o flask no ambiente virtual com o `pip`.**
- **É possível saber que o ambiente virtual está ativo quando visualizar o nome dele acima da linha do usuário (`.venv`).**



```
(.venv) ESS185N1264916+usuario@ESS185N126
```

Criando um .gitignore

- Quando criamos o virtualenv (.venv) adicionamos arquivos executáveis Python que não queremos versionar.
- Então vamos criar um arquivo .gitignore com o conteúdo:




The screenshot shows a code editor with a file named `.gitignore` open. The file contains two lines of text: `.venv/` on the first line and `*.pyc` on the second line. The lines are numbered 1 and 2 on the left side of the editor. The file name `.gitignore` is visible in the top right corner of the editor window.

```
1 .venv/  
2 *.pyc
```








Vamos abrir o VScode

**Podemos abrir o VS
code usando o
comando 'code .'**

**Ou da maneira
tradicional com o
botão direito do
mouse.**




FileEditSelectionViewGo...←→


🔍 site



EXPLORER

...

▼ OPEN EDITORS   

×  Welcome


▼ SITE

> .venv


> exemplos


> static


<> index.html


 Welcome ×


Start

 New File...

 Open File...

 Open Folder...

 Clone Git Repository...

 Connect to...

Recent

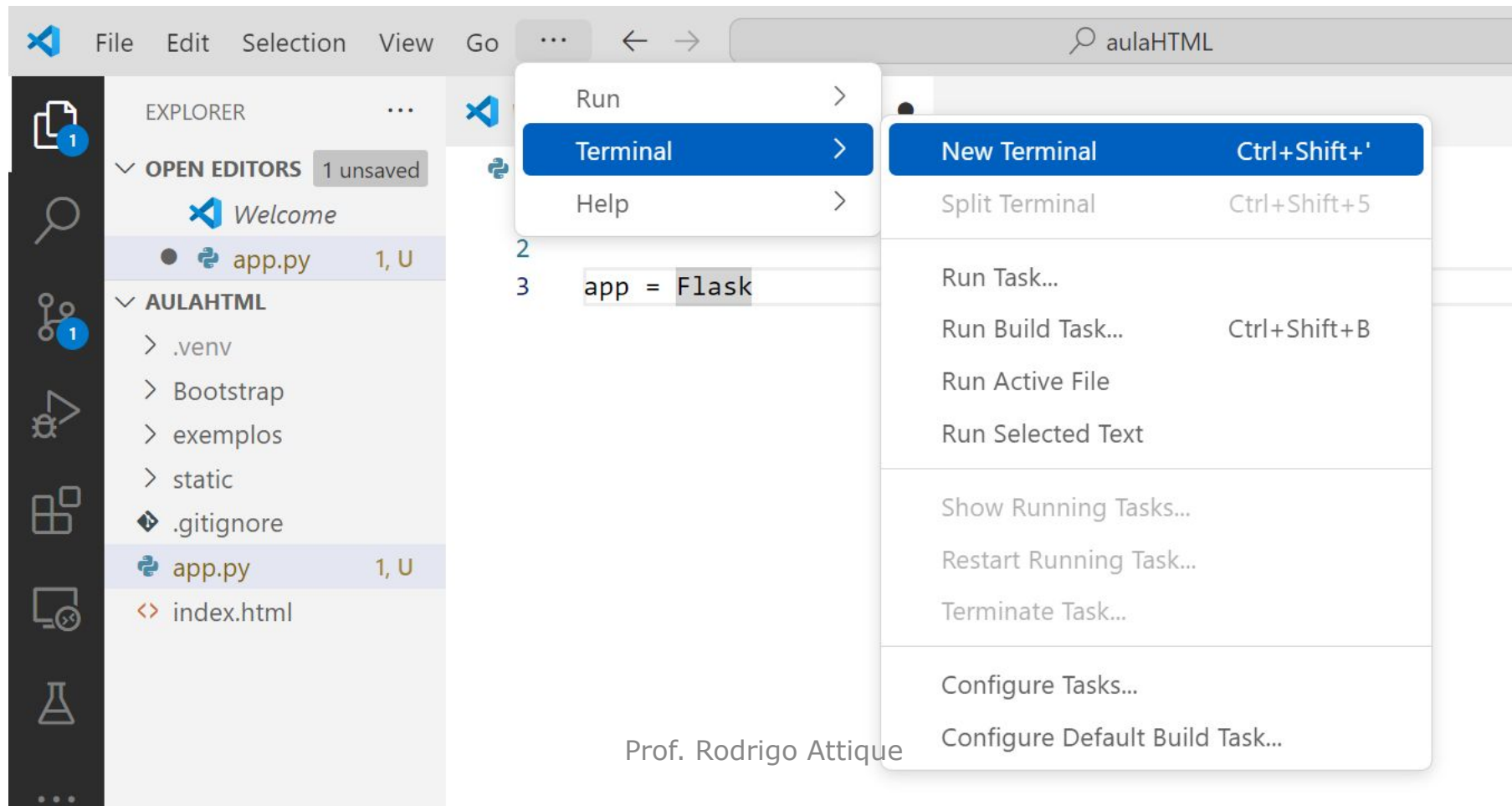
site G:\Meu Drive\SENAI\Aulas\HTML

escola C:\Users\usuario\projetos\django-teste

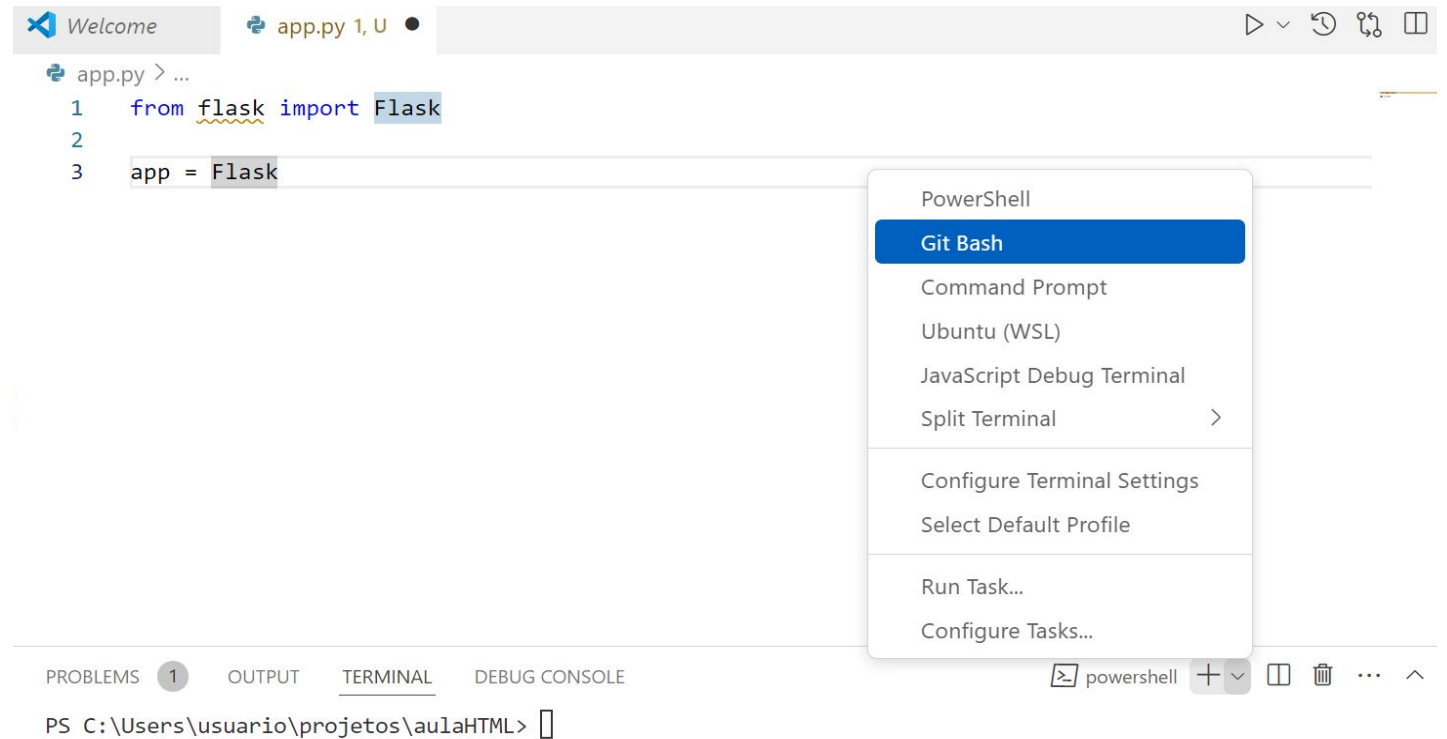
financeiras C:\Users\usuario\projetos

Instalando o flask no virtualenv

- Crie um novo terminal git-bash no próprio VScode.



Escolha o
git-bash



Instale o Flask no virtualenv

- Se o (.venv) não estiver aparecendo acima do usuário, [ative o virtualenv.](#)

PROBLEMS

1

OUTPUT

TERMINAL

DEBUG CON:

(.venv)

ESS185N1264916+usuario@ESS185N1264916 MINGW

\$ pip install Flask

Se você receber uma mensagem

- Se aparecer essa notificação execute o comando em verde e depois instale o Flask novamente com o pip.

```
[notice] A new release of pip is available: 24.0 -> 24.2
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip  
(.venv)
```

```
ESS185N1264916+usuario@ESS185N1264916 MINGW64 ~/projetos/aulaHTML (main)
```

```
○ $ █
```


Nosso primeiro app

- Na raiz do site crie um arquivo chamado app.py com o conteúdo abaixo.

 app.py > ...

```
1  from flask import Flask # Importa o flask
2
3  app = Flask(__name__) # cria uma instância
4
5  @app.route("/", methods=('GET',)) # Assina uma rota
6  def index(): # função responsável pela página
7      return "<h1>Página inicial</h1>" # HTML retornado
```

Executando o projeto

- No terminal abaixo vamos digitar o comando: **flask run**
- E depois acessar o endereço indicado no navegador.

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
TypeError: Rule.__init__() got an unexpected keyword argument 'method'
(.venv)
```

```
ESS185N1264916+usuario@ESS185N1264916 MINGW64 ~/projetos/aulaHTML (main)
```

```
○ $ flask run 1
```

```
* Debug mode: off
```

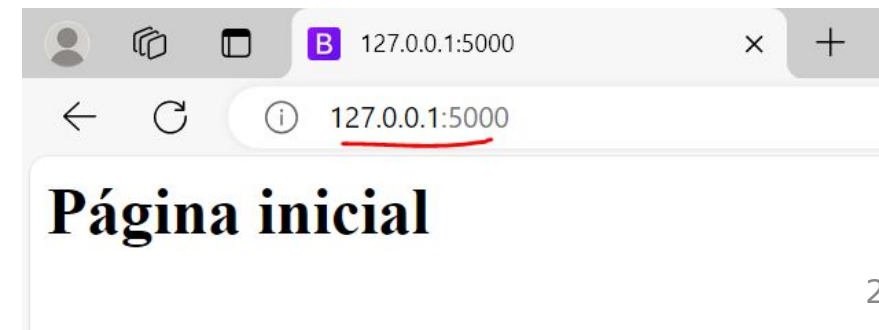
```
WARNING: This is a development server. Do not use it in a production deployment WSGI server instead.
```

```
* Running on http://127.0.0.1:5000
```

```
Press CTRL+C to quit
```

Rotas

- Antes de **começarmos...**
- Uma rota é nada mais nada menos do que o endereço da página.
- Todo site é composto por seu servidor.
- E as páginas são acessadas por um endereço depois do endereço do site.
- Logo se tivermos um servidor com endereço **10.9.12.200** poderemos acessar no navegador com:
- **http://10.9.12.200**



Servidor

- Um servidor é um computador responsável por hospedar o programa do site.
- Como todo computador em rede precisa de um endereço IP.
- Para saber o IP do servidor basta digitar **ipconfig** no terminal;



```
Sufixo DNS específico de conexão. . . . . : 
Va Adaptador de Rede sem Fio Wi-Fi:
Sufixo DNS específico de conexão. . . . . : sp.local
Endereço IPv6 de link local . . . . . : fe80::31a:2262:b8c9:5d4b%16
Endereço IPv4. . . . . : 10.134.75.76
Máscara de Sub-rede . . . . . : 255.255.255.0
Gateway Padrão. . . . . : 10.134.75.1
```

Mas por que IP

- **Todo dispositivo ligado a rede precisa de um endereço IP.**
- **O protocolo TCP/IP é o responsável por endereçar os micros.**
- **Quando uma máquina oferece um serviço em rede ela é chamada servidor.**
- **O serviço é acessado através do endereço desse servidor seguido da porta de serviço.**
- **No caso de serviços http usaremos a porta 80 e 443.**


```
ESS185N1264916+usuario@ESS185N1264916 MINGW64 ~
```

```
$ ping -4 x.com
```

```
Disparando x.com [172.66.0.227] com 32 bytes de dados:
```

```
Resposta de 172.66.0.227: bytes=32 tempo=45ms TTL=55
```

```
Resposta de 172.66.0.227: bytes=32 tempo=33ms TTL=55
```

```
Resposta de 172.66.0.227: bytes=32 tempo=24ms TTL=55
```

```
Resposta de 172.66.0.227: bytes=32 tempo=32ms TTL=55
```

```
Estatísticas do Ping para 172.66.0.227:
```

```
Pacotes: Enviados = 4, Recebidos = 4, Perdidos = 0 (0% de  
perda),
```

```
Aproximar um número redondo de vezes em milissegundos:
```

```
Mínimo = 24ms, Máximo = 45ms, Média = 33ms
```

Como assim?

Digite o comando ping -4 qualquer_site

```
ESS185N1264916+usuario@ESS185N1264916 MINGW64 ~
```

```
$ ping -4 google.com
```

```
Disparando google.com [142.250.219.46] com 32 bytes de dados:
```

```
Resposta de 142.250.219.46: bytes=32 tempo=26ms TTL=55
```

```
Resposta de 142.250.219.46: bytes=32 tempo=84ms TTL=55
```

Domínio vs IP

- **Conforme vimos na imagem acima a saída do comando ping -4 x.com retorna o endereço IP para o domínio x.com.**
- **Na internet há milhares de endereços IP que funcionam como o nosso RG.**
- **Entretanto quando nos apresentamos para outras pessoas usamos nosso nome associado a esse RG.**
- **Um domínio nada mais é do que um nome para evitarmos digitar números no navegador.**

Mas e o Flask?

- **Voltando ao framework Flask, quando estivermos desenvolvendo nosso site vamos usar endereços IPs.**
- Logo todas as nossas rotas serão acessadas de forma local com um 127.0.0.1:5000.
- Repare que o Flask não trabalha na porta 80, logo precisamos indicar a porta no endereço.
- Então quer dizer que meu site depois de pronto vai usar esse endereço?
- **Depois é preciso hospedá-lo em um servidor e apontar um domínio para o IP do servidor.**

Adicionando novas rotas

- Repare que repetimos a mesma estrutura mudando apenas algumas informações.

```
5  @app.route("/", methods=('GET',)) # Assina uma rota
6  ✓ def index(): # função responsável pela página
7      |     return "<h1>Página inicial</h1>" # HTML retornado
8
9  @app.route("/outra_pagina", methods=('GET',))
10 ✓ def outra():
11     |     return "<h1>Outra página</h1>"
```

Explicando



- **Linha 5 e 9: Definimos duas rotas a primeira é a rota principal (/) e a segunda uma rota secundária (/outra..)**
- **Linha 6 e 10: Cada rota precisa de uma função responsável.**
- **Linha 7 e 11: Toda função precisa de um retorno, no caso o HTML processado.**
- **Reinicie o flask com <CTRL>+<C> e execute o flask run**

Configurando o VSCode para Flask

- A forma usada é a maneira tradicional, podemos criar uma configuração no VSCode para executar o Flask para nós.
- Siga os passos a seguir.



File Edit Selection View

RUN AND DEBUG

▼ RUN

Run and Debug

To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

Show automatic Python

Select debugger

Python Debugger

Install an extension for Python...

```
5 @app.route("/", methods=('GE
```

Select a debug configuration

Debug Configuration

Python File Debug the currently active Python file

Python File with Arguments Debug the currently active Python file with arguments

Module Debug a Python module by invoking it with '-m'

Remote Attach Attach to a remote debug server

Attach using Process ID Attach to a local process

Django Launch and debug a Django web application

FastAPI Launch and debug a FastAPI web application

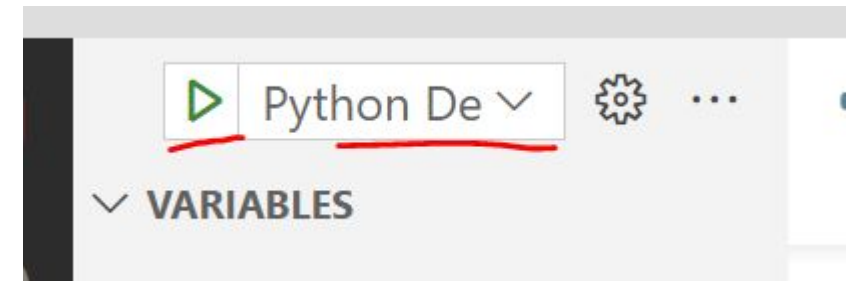
Flask Launch and debug a Flask web application

Pyramid Launch and debug a Pyramid web application

```
3 // Hover to view descriptions of existing attributes.
4 // For more information, visit: https://go.microsoft.com/fwlink
5 "version": "0.2.0",
6 "configurations": [
7     {
8         "name": "Python Debugger: Flask",
9         "type": "debugpy",
10        "request": "launch",
11        "module": "flask",
12        "env": {
13            "FLASK_APP": "app.py",
14            "FLASK_DEBUG": "1"
15        },
16        "args": [
17            "run",
18            "--debugger",
19            "--reload"
20        ],
21        "jinja": true,
22        "autoStartBrowser": false
```

Executando com VSCode

- Feito isso já podemos ver o botão Play na guia Debug.
- É possível apertar Play ou teclar <F5>.
- Quando começar a executar você verá mensagens no terminal como da maneira anterior e um painel de controle no canto superior.
- A partir de então não será mais preciso usar flask run ou reiniciar o serviço a cada alteração



Desafio!

- Exclua a rota `outra_pagina` e crie mais 3 rotas novas.
 - `/galeria`
 - `/contato`
 - `/sobre`
-
- **Não esqueça de testar as rotas**

Enviando dados na response

- Todo framework permite processar dados e enviá-los para o usuário.
- Como flask usa Python vamos manipular variáveis Python em nossas funções.
- Depois retorná-los.

Modifique o index e teste a rota

```
5 @app.route("/", methods=('GET',)) # Assina uma rota
6 def index(): # função responsável pela página
7     nome = 'Rodrigo' # use o seu nome
8     # HTML retornado
9     return f"""<h1>Página inicial</h1>
10         <p>Olá {nome}, que nome bonito!
11         """
```



Desafio!

- Crie uma rota que devolva para o usuário em uma lista HTML os nomes de 3 personagens de desenho (inclusive anime) que marcaram sua infância.

- **Dica:**

- `/personagen`
- `{p1}...`



Passando argumentos de URL

- Também é possível enviar dados para o framework através de argumentos de URL.
- Esses argumentos são como variáveis que chegam as funções.
- Repare que sempre assinamos nossas rotas com o método GET.
- Isso quer dizer que vamos passar esses valores diretamente na URL.

Modifique o index

```
app.py > ...
1  from flask import (Flask, request) # Importa o flask
2
3  app = Flask(__name__) # cria uma instância
4
5  @app.route("/", methods=('GET',)) # Assina uma rota
6  def index(): # função responsável pela página
7      nome = request.args.get('nome') # use o seu nome
8      # HTML retornado
9      return f"""<h1>Página inicial</h1>
10         <p>Olá {nome}, que nome bonito!
11         """
```

Faça os testes



Página inicial

Olá None, que nome bonito!



Página inicial

Olá Rodrigo, que nome bonito!

Podemos receber mais valores....

- Modifique personagens e faça os testes

```
25 @app.route("/personagens", methods=('GET',)) # Assina uma rota
26 def personagens(): # função responsável pela página
27     p1 = request.args.get('p1') # use o seu nome
28     p2 = request.args.get('p2')
29     p3 = request.args.get('p3')
30     # HTML retornado
31     return f"""<h1>Página inicial</h1>
32         <ul><li>{p1}</li><li>{p2}</li><li>{p3}</li></ul>
33     """
```

127.0.0.1:5000/personagens?p1=batman&p2=miranha&p3=goku

Desafios

1. Crie uma rota chamada area que receba a largura e o comprimento de um terreno e retorne para o usuário a área informada $\rightarrow L=? * C=? \rightarrow Area = ?$
2. Crie uma rota que receba um número e retorne para o usuário se ele é par ou ímpar.
3. Crie uma rota que receba o nome e o sobrenome e retorne para o usuário a sentença \rightarrow sobrenome, nome.

NOTA: use `int(valor)`, `float(valor)`, `str(valor)` para converter dados

Recebendo parâmetros pré-definidos

- A forma ensinada acima é muito prática porém faz nossas URLs ficarem muito poluídas.
- Vamos começar a trabalhar com um conceito chamado URLs elegantes.
- Nesse caso ao invés de argumentos vamos usar parâmetros separados por '/' barra.
- **Logo: /teste?arg1=teste&arg2=teste**
- **Fica: /teste/teste/teste**

Modifique o index e faça o teste

```
5 @app.route("/<string:nome>", methods=('GET',)) # Assina uma rota
6 def index(nome): # função responsável pela página
7     # HTML retornado
8     return f"""<h1>Página inicial</h1>
9         <p>Olá {nome}, que nome bonito!
10        """
```

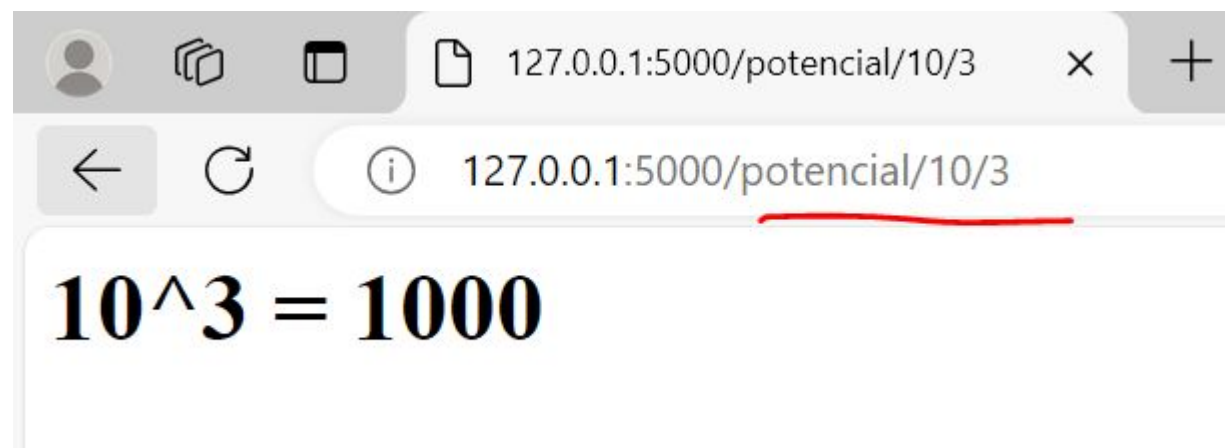


Mas qual devo usar?

- **Ambas as forma são corretas.**
- **Porém do ponto de vista praticidade e segurança:**
 - **A segunda forma fixa os argumentos que a função deve receber.**
 - **O desenvolvedor tem uma visão mais clara para manutenção do código.**
 - **Do lado do cliente são menos parâmetros para trabalhar na URL.**

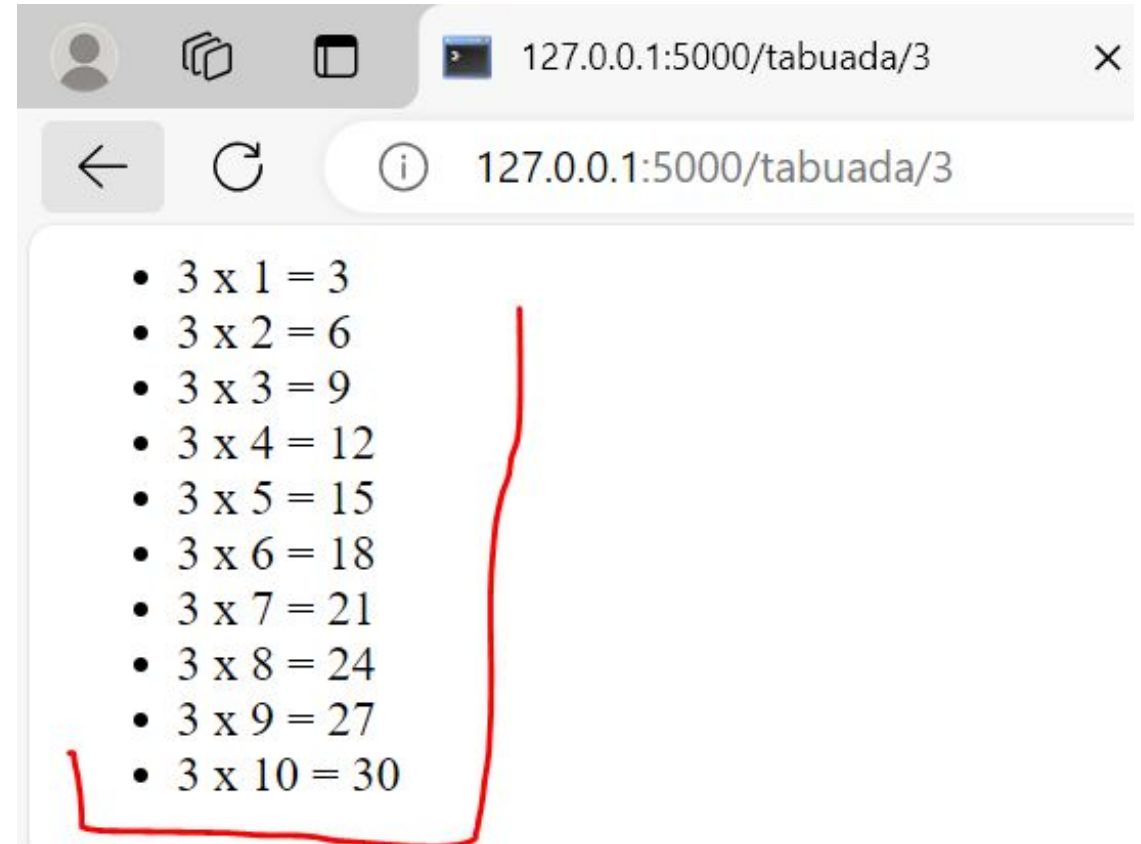
DESAFIO!

- Crie uma nova rota chamada potencial que recebe dois números e retorna ao usuário o numero da esquerda elevado ao número da direita.
- [Use a documentação do flask como referencia.](#)



DESAFIO 2

- Crie uma rota chamada tabuada que receba um número e exiba a tabuada desse número.
 - Use `` e `` para organizar.
 - Use um laço for para gerar a tabuada

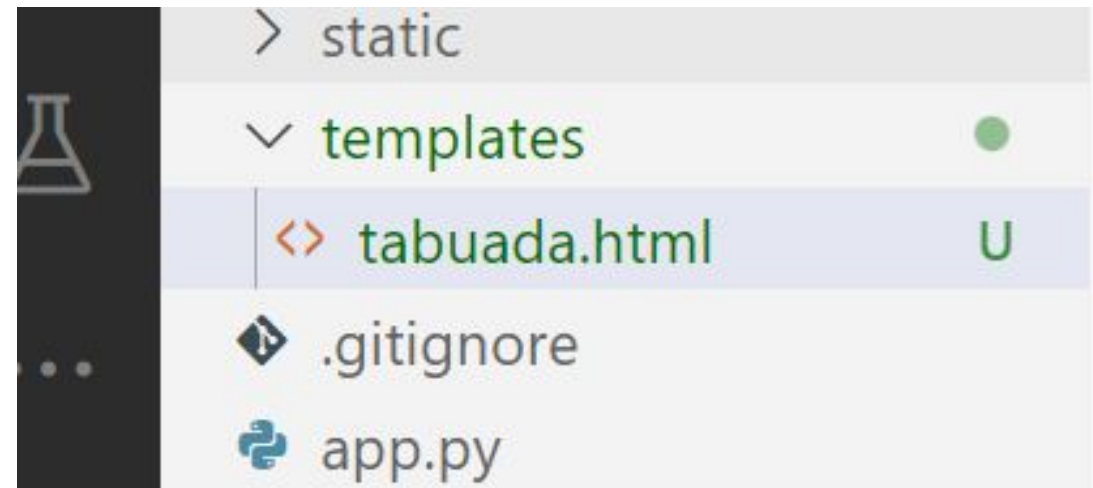


Templates

- Da maneira em que estamos trabalhando é inviável para construir sites.
- Para resolver isso o flask usa uma biblioteca de templates ou Template Engine chamada Jinja2
- O Jinja nos permite escrever código HTML separado do Python.
- Basta retornar o caminho do template ao invés de uma string html

Preparando o projeto

- Para o flask reconhecer os templates Jinja (HTML) precisamos coloca-los dentro de um diretório chamado templates.
- Crie um diretório chamado templates na raiz do projeto.
- Crie um arquivo chamado tabuada.html dentro desse diretório



Modifique o app.py

```
app.py > tabuada
1  from flask import (Flask, render_template, request) # Importa o flask
2
38  @app.route("/tabuada/<int:numero>", methods=("GET", ))
39  def tabuada(numero):
40
41      return render_template('tabuada.html', numero=numero)
```

Modifique o arquivo tabuada.html

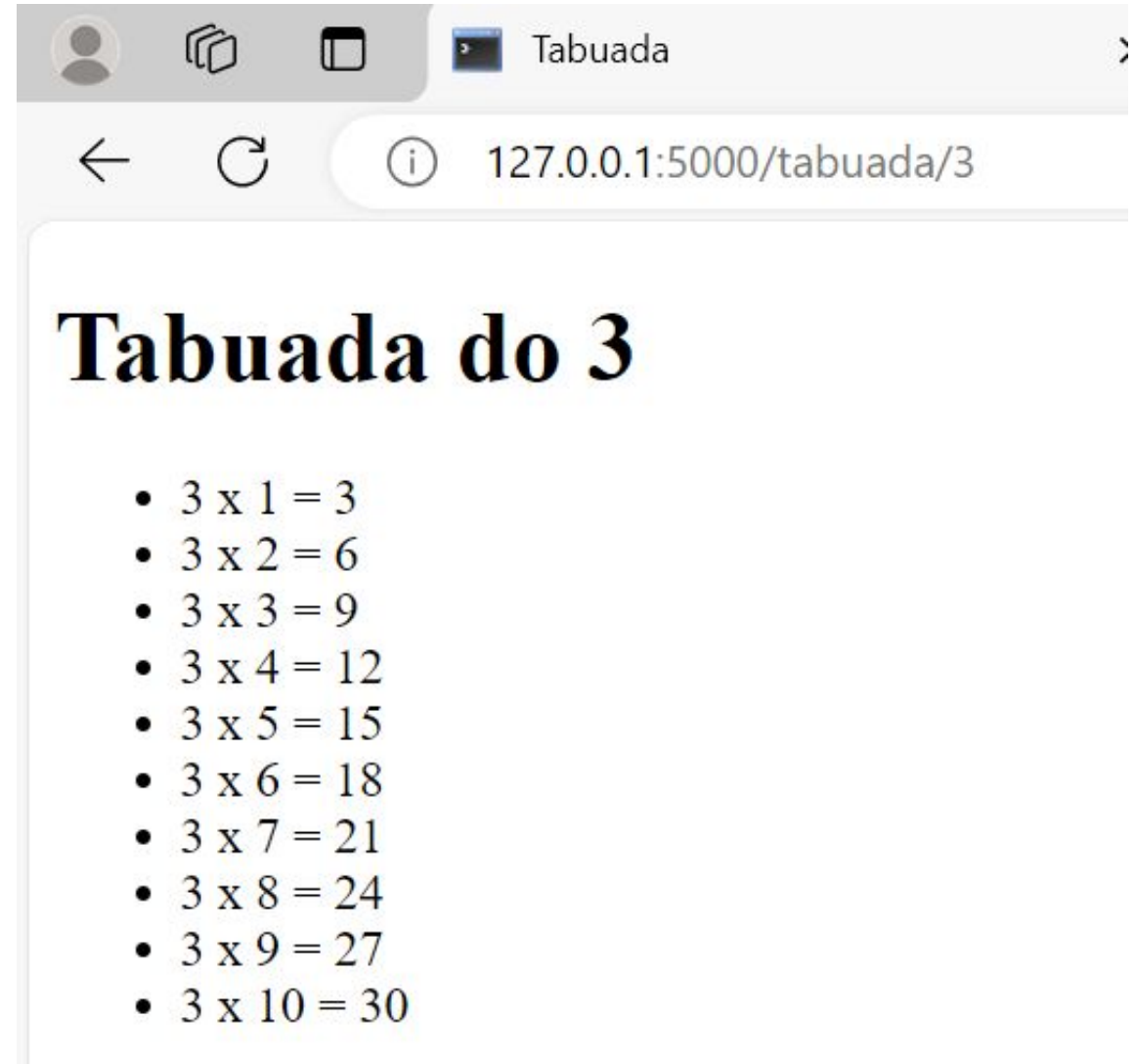
templates > <> tabuada.html >  html



```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Tabuada</title>
7  </head>
8  <body>
9      <h1>Tabuada do {{numero}}</h1>
10     <ul>
11         {% for p in range(1,11) %}
12             <li>{{numero}} x {{p}} = {{numero*p}}</li>
13         {% endfor %}
14     </ul>
15 </body>
16 </html>
```

Explicando

- Perceba que dentro do **template .html** escrevemos as **variáveis** entre **{{...}}** **chaves**.
- E **blocos de código python** entre **{%...%}** **chave por cento**.
- Sempre finalizados com um **{%end...%}**

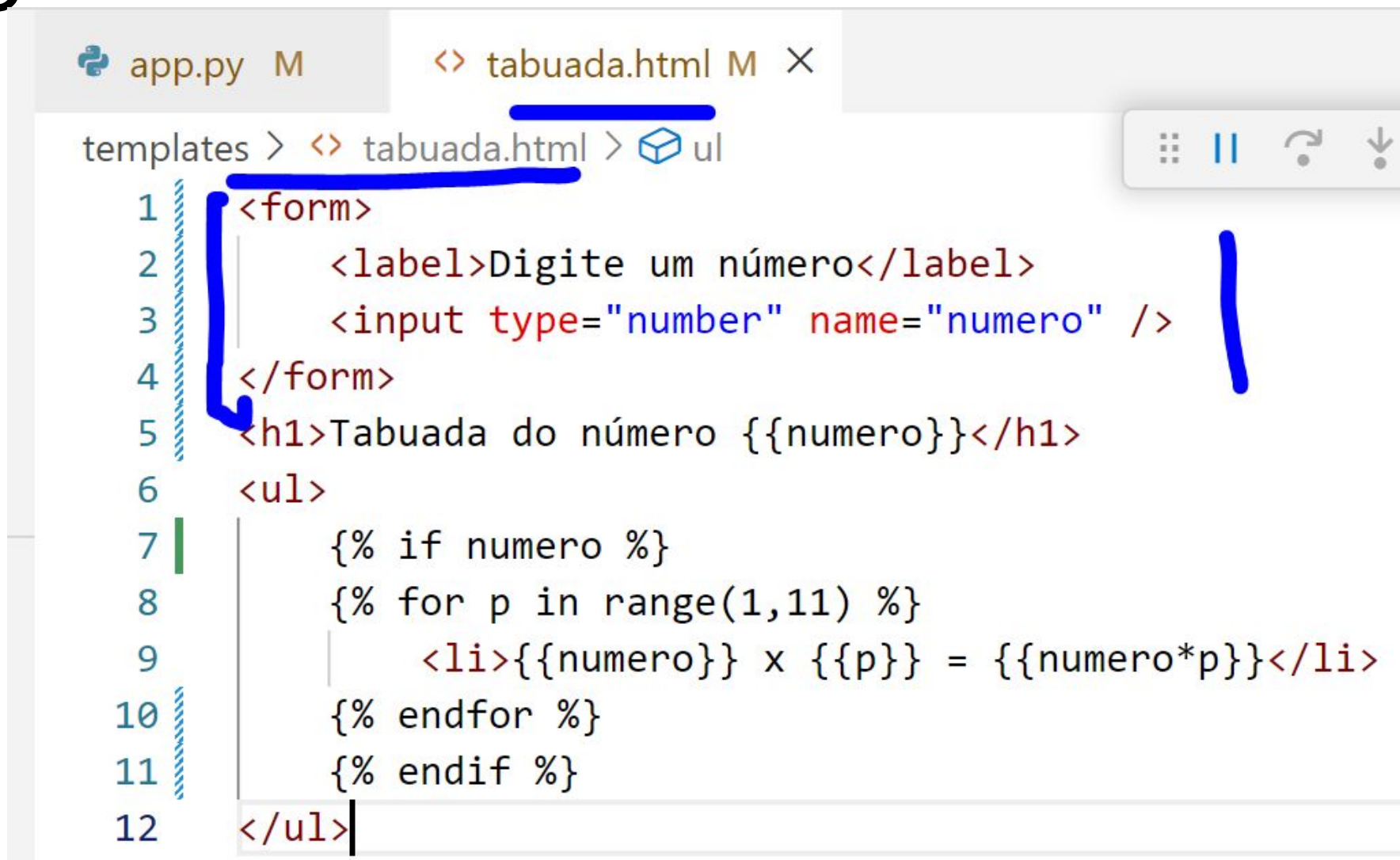


Desafio

- Crie templates para todos as rotas criadas até aqui.

Desafio

- Adicione o formulário ao tabuada.html



The screenshot shows a code editor with two tabs: 'app.py' and 'tabuada.html'. The 'tabuada.html' tab is active, showing a Jinja2 template. The code is as follows:

```
templates > <> tabuada.html > ul
1  <form>
2      <label>Digite um número</label>
3      <input type="number" name="numero" />
4  </form>
5  <h1>Tabuada do número {{numero}}</h1>
6  <ul>
7      {% if numero %}
8      {% for p in range(1,11) %}
9          <li>{{numero}} x {{p}} = {{numero*p}}</li>
10         {% endfor %}
11         {% endif %}
12 </ul>
```

Blue annotations highlight the new form structure (lines 1-4) and the existing h1 tag (line 5). A blue bracket on the right side of the code block groups the form and the h1 tag.

Desafio

- Descubra porque o formulário não calcula a tabuada do número informado no campo e acrescente um outro componente que está faltando nele.

Capturando requisições de formulário

- Quando fazemos o envio de um formulário precisamos dizer a ele qual o método usar.
- Repare que depois do envio a URL ficou assim -> <http://127.0.0.1:5000/tabuada?numero=11>
- Repare que ele criou um argumento de URL `numero=11`
- Podemos resolver isso de várias maneiras:
- 1º É usando `request.args.get('numero')` com um `if` para saber qual variável usar.

Capturando requisições de formulário

- **Por padrão todos os formulário usam o método GET, ou seja, passando argumentos expostos na URL como vimos.**
- **Se quiser saber mais sobre métodos (verbos)**
<https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods>
- **Vamos usar a primeira solução para resolver nosso problema de tabuada.**

Modifique a rota tabuada

```
57 @app.route("/tabuada")
58 @app.route("/tabuada/<numero>", methods=("GET", ))
59 def tabuada(numero = None): # None desobriga o valor
60
61     if 'numero' in request.args: # se argumento existir
62         numero = request.args.get('numero') # atualiza numero
63
64     return render_template('tabuada.html', numero=numero)
```

← ↻ ⓘ 127.0.0.1:5000/tabuada?numero=12

Digite um número

Tabuada do número 12

- $12 \times 1 = 12$
- $12 \times 2 = 1212$
- $12 \times 3 = 121212$
- $12 \times 4 = 12121212$
- $12 \times 5 = 1212121212$
- $12 \times 6 = 121212121212$
- $12 \times 7 = 12121212121212$
- $12 \times 8 = 1212121212121212$
- $12 \times 9 = 121212121212121212$
- $12 \times 10 = 12121212121212121212$

Padronizando o HTML

- **A grande vantagem de usar uma template engine como o Jinja2 é poder criar um arquivo HTML padrão que poderemos usar em qualquer parte do código.**
- **Se observarmos as páginas criadas a única coisa que muda é o conteúdo do body.**
- **As tags HEAD por exemplo se repetem em todas as páginas.**
- **Podemos criar um template padrão para reutilizar código.**

Crie um arquivo layout.html



```
app.py layout.html potencial.html tabuada.html
templates > layout.html > html
1 <!DOCTYPE html>
2 <html lang="pt-br">
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content="width=device-width, initial-scale=1.0">
6     <title>Meu site</title>
7 </head>
8 <body>
9     {% block content %}
10    {% endblock %}
11 </body>
12 </html>
```

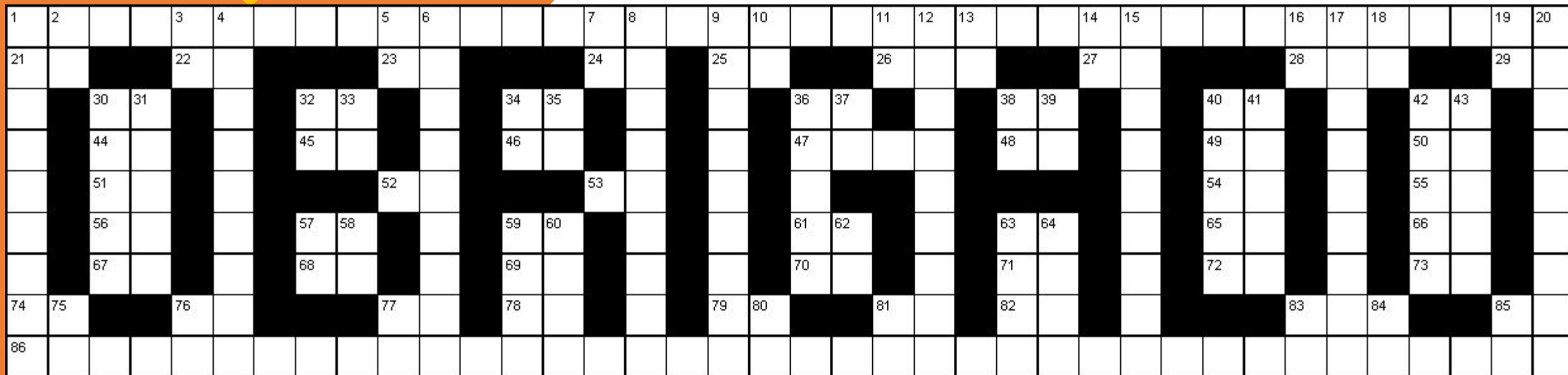
Modifique os demais arquivos

templates > <> tabuada.html >  ul

```
1  {% extends 'layout.html' %}  
2  {% block content %}  
3      <h1>Tabuada do {{numero}}</h1>  
4      <ul>  
5          {% for p in range(1,11) %}  
6              <li>{{numero}} x {{p}} = {{numero*p}}</li>  
7          {%endfor%}  
8  
9      {% endblock %}
```

Pronto!

- Agora basta inserir todas as informações “estáticas” do site no arquivo Layout.html.
- E o conteúdo que sempre muda ficará em arquivos separados.
- Podemos ter quantos templates desejarmos e quantos padrões quiser.
- Podemos inclusive **injetar** html em outros arquivos.



Esta Foto de Autor Desconhecido está licenciado em [CC BY-NC](https://creativecommons.org/licenses/by-nc/4.0/)