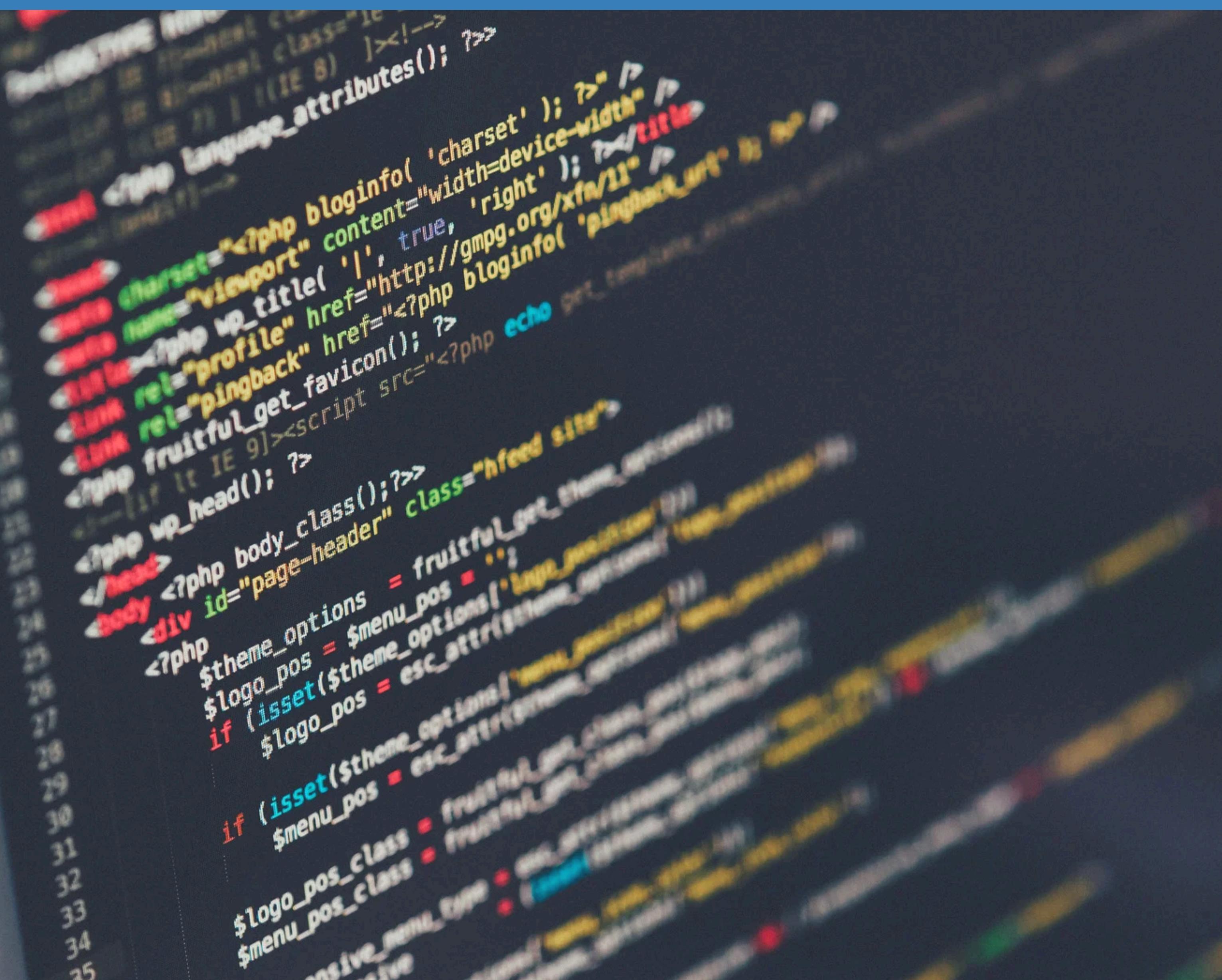


Python para Web



UTILIZANDO PYTHON PARA O DESENVOLVIMENTO WEB

by Ismael Roberto

Boas Vindas!!

Bem-vindo ao "Python para Web"!
Este eBook foi criado para aspirantes a desenvolvedores que desejam aprender como utilizar Python no desenvolvimento web. Vamos guiá-lo desde os conceitos básicos até técnicas mais avançadas, utilizando dois frameworks mais populares: Flask e Django.



CAPÍTULO 1: INTRODUÇÃO AO PYTHON

O QUE É PYTHON?

PYTHON É UMA LINGUAGEM DE PROGRAMAÇÃO DE ALTO NÍVEL, CONHECIDA POR SUA SINTAXE SIMPLES E FÁCIL DE APRENDER. CRIADA POR GUIDO VAN ROSSUM E LANÇADA PELA PRIMEIRA VEZ EM 1991, PYTHON SE TORNOU UMA DAS LINGUAGENS MAIS POPULARES DO MUNDO, USADA EM UMA AMPLA VARIEDADE DE APLICAÇÕES, INCLUINDO DESENVOLVIMENTO WEB, CIÊNCIA DE DADOS, INTELIGÊNCIA ARTIFICIAL E MAIS.

POR QUE USAR PYTHON PARA DESENVOLVIMENTO WEB?

PYTHON É UMA ESCOLHA POPULAR PARA O
DESENVOLVIMENTO WEB POR VÁRIAS RAZÕES:

- **FACILIDADE DE APRENDIZADO:** SUA SINTAXE CLARA E LEGÍVEL FACILITA A APRENDIZAGEM PARA INICIANTES.
- **BIBLIOTECAS E FRAMEWORKS:** PYTHON POSSUI UMA VASTA QUANTIDADE DE BIBLIOTECAS E FRAMEWORKS, COMO FLASK E DJANGO, QUE SIMPLIFICAM O DESENVOLVIMENTO WEB.
- **COMUNIDADE ATIVA:** A COMUNIDADE PYTHON É GRANDE E ATIVA, OFERECENDO SUPORTE, TUTORIAIS E RECURSOS.
- **ESCALABILIDADE E FLEXIBILIDADE:** PYTHON PODE SER USADO PARA CONSTRUIR DESDE PEQUENAS APLICAÇÕES WEB ATÉ GRANDES PLATAFORMAS ESCALÁVEIS.

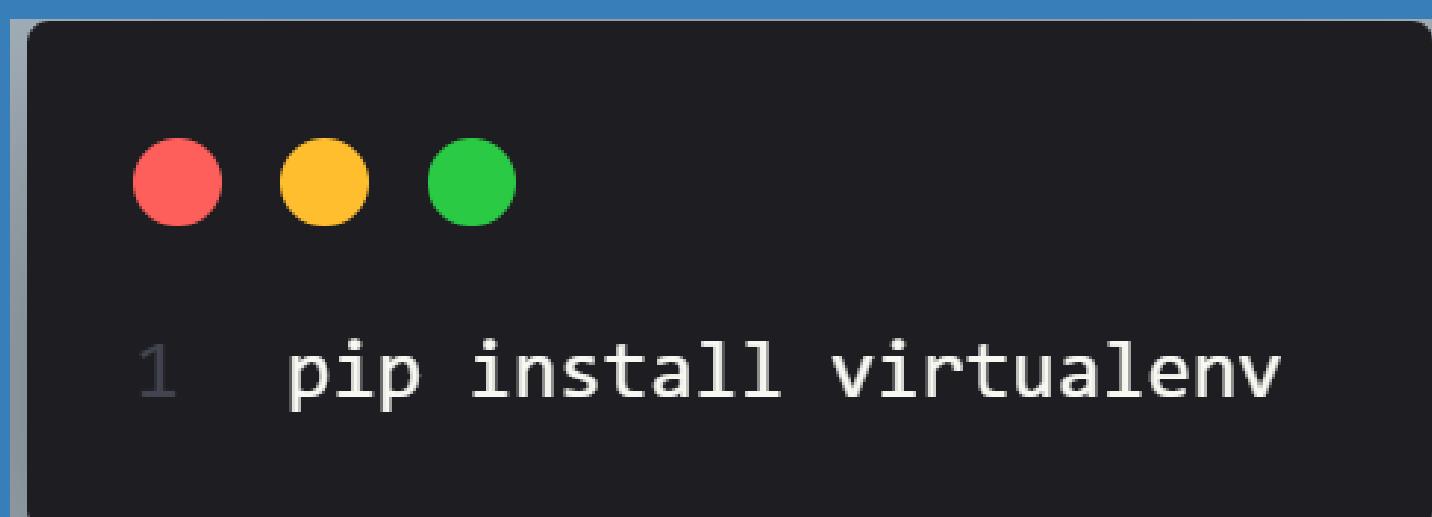
INSTALANDO PYTHON

PARA COMEÇAR A USAR PYTHON, VOCÊ PRECISA INSTALÁ-LO EM SEU SISTEMA. VISITE O SITE OFICIAL DO PYTHON E BAIXE A VERSÃO MAIS RECENTE. SIGA AS INSTRUÇÕES DE INSTALAÇÃO FORNECIDAS PARA SEU SISTEMA OPERACIONAL.

CONFIGURANDO UM AMBIENTE DE DESENVOLVIMENTO UM AMBIENTE DE DESENVOLVIMENTO BEM CONFIGURADO É CRUCIAL PARA A PRODUTIVIDADE. RECOMENDAMOS O USO DE UM AMBIENTE VIRTUAL PARA GERENCIAR AS DEPENDÊNCIAS DO SEU PROJETO.

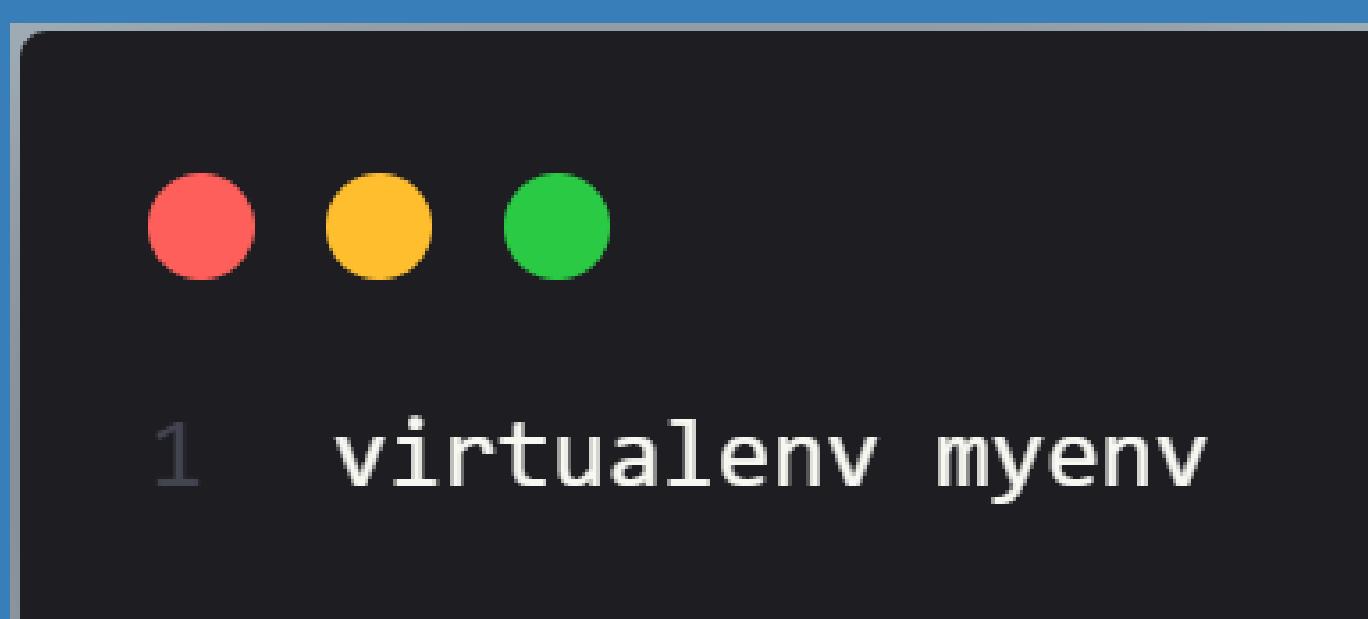
SIGA OS PASSOS ABAIXO PARA CONFIGURAR SEU AMBIENTE DE DESENVOLVIMENTO:

1 - INSTALE O VIRTUAL ENV:



```
1 pip install virtualenv
```

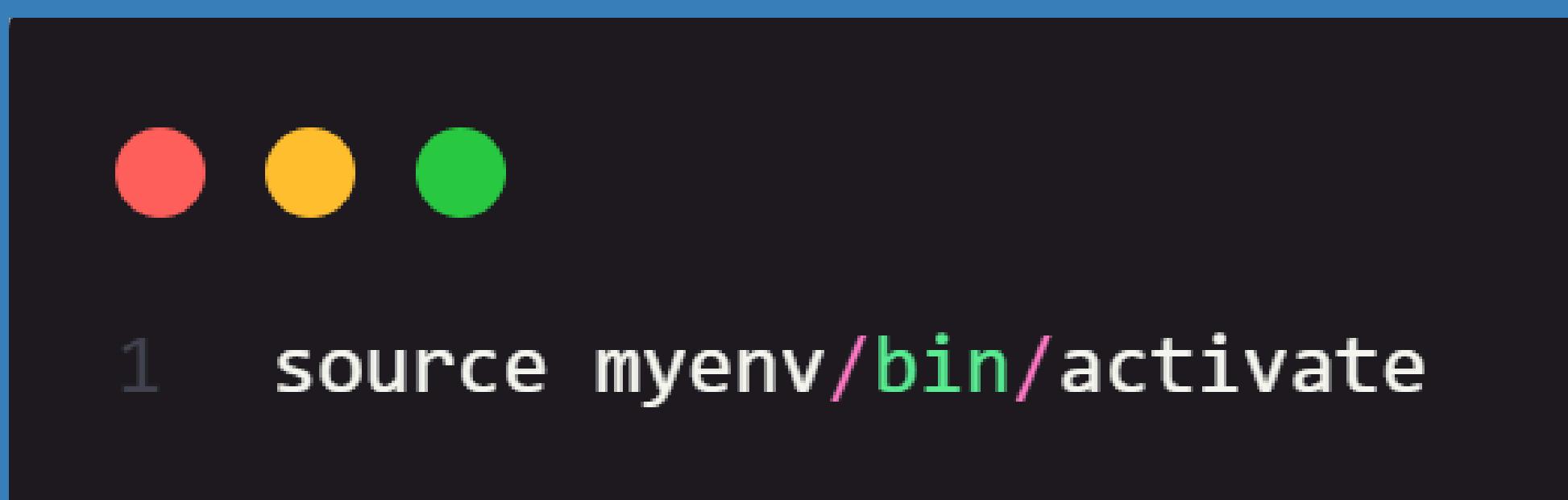
2 - CRIE UM AMBIENTE VIRTUAL:



```
1 virtualenv myenv
```

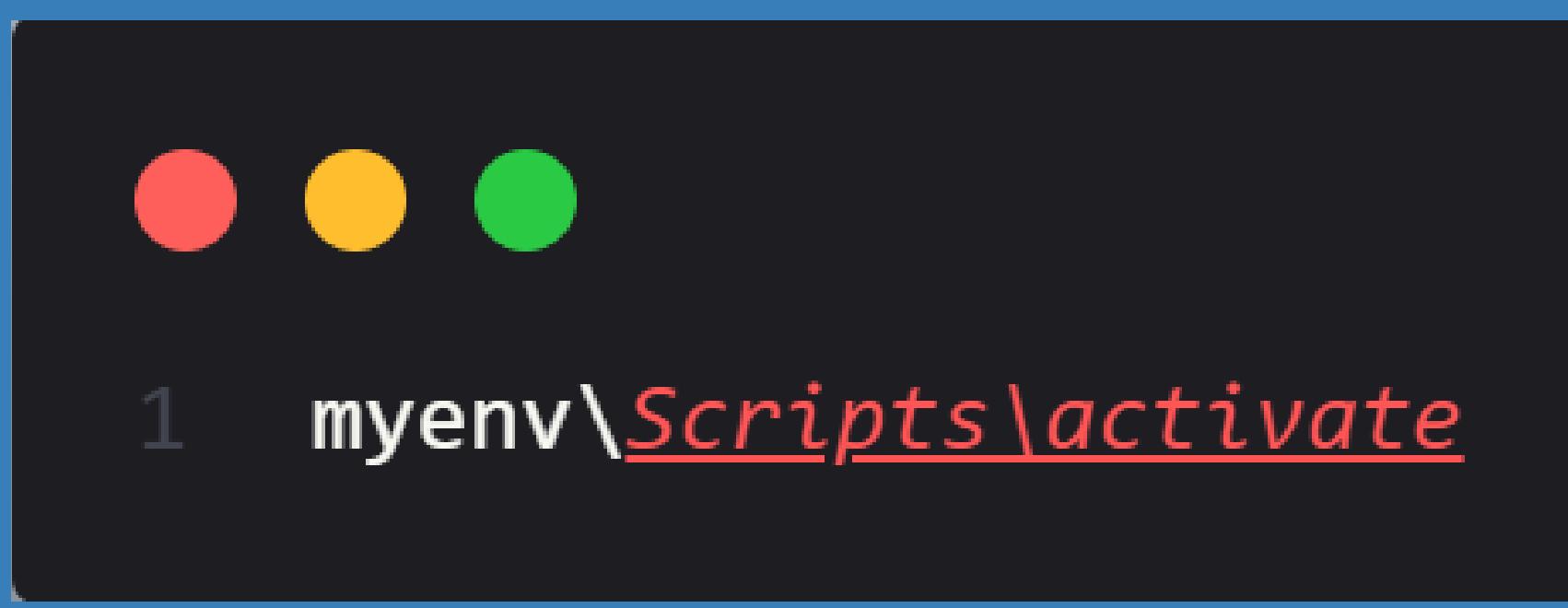
3 - ATIVE O AMBIENTE VIRTUAL

3.1 - NO WINDOWS:



```
1 source myenv/bin/activate
```

3.2 - NO MACOS E LINUX:



```
1 myenv\Scripts\activate
```

AGORA VOCÊ ESTÁ PRONTO PARA COMEÇAR A DESENVOLVER COM PYTHON!

CAPÍTULO 2: FUNDAMENTOS PARA WEB COM PYTHON

CONCEITOS BÁSICOS DE DESENVOLVIMENTO WEB

ANTES DE MÉRGENHARMOS NO
DESENVOLVIMENTO WEB COM PYTHON, É
IMPORTANTE ENTENDER ALGUNS CONCEITOS
FUNDAMENTAIS:

- **CLIENTE E SERVIDOR:** A WEB FUNCIONA NO MODELO CLIENTE-SERVIDOR, ONDE O CLIENTE (GERALMENTE UM NAVEGADOR) FAZ REQUISIÇÕES AO SERVIDOR, QUE RESPONDE COM OS RECURSOS SOLICITADOS (COMO PÁGINAS WEB, IMAGENS, ETC.).
- **HTTP:** HYPERTEXT TRANSFER PROTOCOL É O PROTOCOLO USADO PARA COMUNICAÇÃO ENTRE CLIENTES E SERVIDORES NA WEB.
- **URLS:** UNIFORM RESOURCE LOCATORS SÃO OS ENDEREÇOS USADOS PARA ACESSAR RECURSOS NA WEB.

HTTP E COMO A WEB FUNCIONA

HTTP É UM PROTOCOLO BASEADO EM REQUISIÇÕES E RESPOSTAS. QUANDO VOCÊ DIGITA UMA URL NO SEU NAVEGADOR, UMA REQUISIÇÃO HTTP É ENVIADA AO SERVIDOR, QUE RESPONDE COM O RECURSO SOLICITADO (COMO UMA PÁGINA HTML).

INTRODUÇÃO AO FLASK: UM MICROFRAMEWORK PARA WEB

FLASK É UM MICROFRAMEWORK PARA DESENVOLVIMENTO WEB EM PYTHON. É CHAMADO DE "MICRO" PORQUE É LEVE E FORNECE APENAS AS FERRAMENTAS ESSENCIAIS PARA CONSTRUIR APLICAÇÕES WEB. É UMA ÓTIMA ESCOLHA PARA INICIANTES DEVIDO À SUA SIMPLICIDADE E FLEXIBILIDADE.

CONSTRUINDO SUA PRIMEIRA APLICAÇÃO WEB COM FLASK

VAMOS CRIAR UMA SIMPLES APLICAÇÃO "HELLO, WORLD!" COM FLASK PARA ENTENDER COMO ELE FUNCIONA.

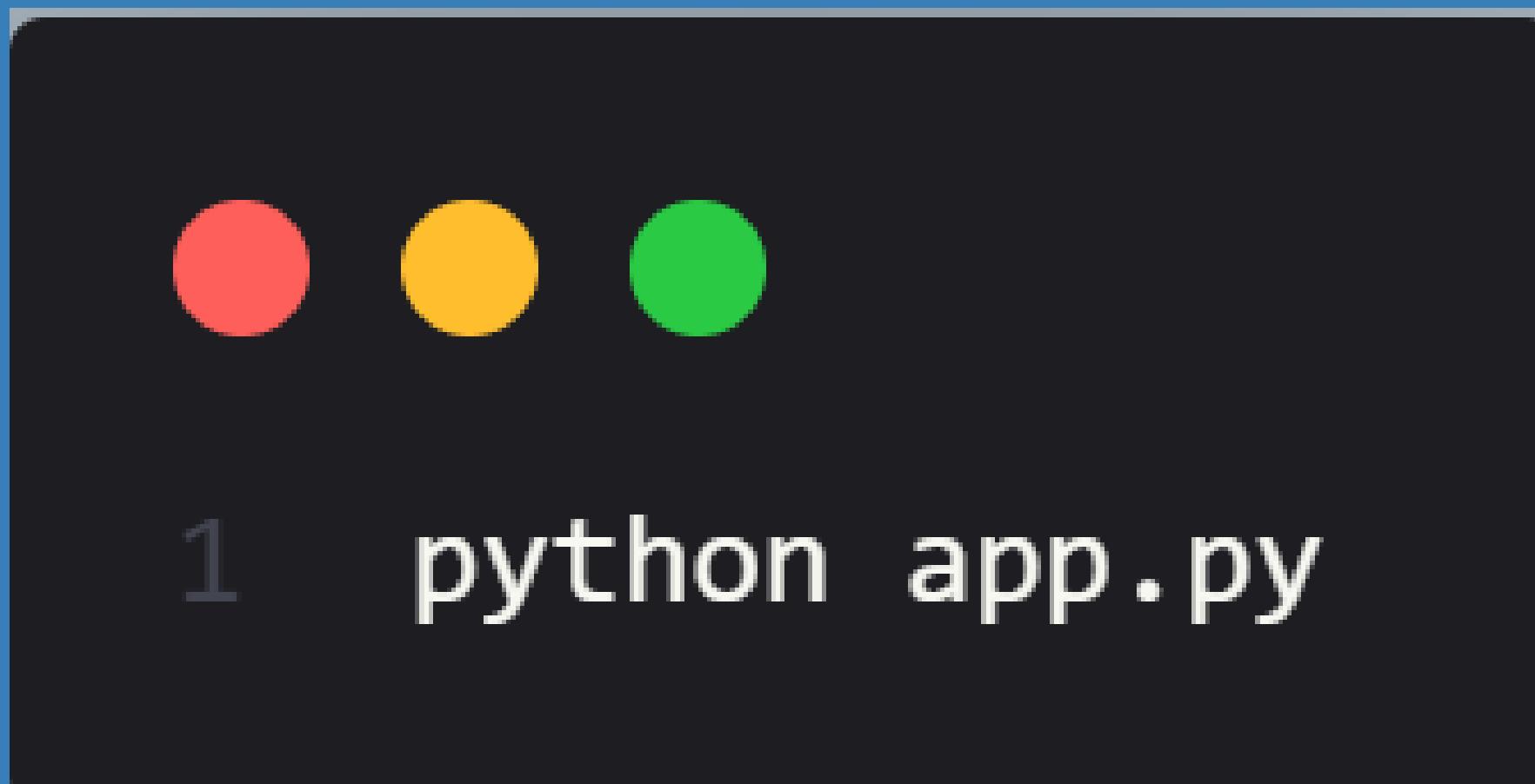
1 - INSTALE O FLASK:

```
1 pip install flask
```

2 - CRIE UM ARQUIVO "APP.PY" COM O SEGUINTE CONTEÚDO:

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5 @app.route('/')
6 def hello_world():
7     return 'Hello, World!'
8
9 if __name__ == '__main__':
10    app.run(debug=True)
```

3 - EXECUTE A APLICAÇÃO:



**ABRA SEU NAVEGADOR E ACESSSE
[HTTP://127.0.0.1:5000/](http://127.0.0.1:5000/). VOCÊ VERÁ A
MENSAGEM "HELLO, WORLD!".**

CAPÍTULO 3: DESENVOLVIMENTO COM FLASK

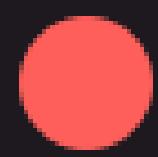
A dark blue background with blurred white text, likely code snippets, visible through a semi-transparent overlay. The text includes words like 'if', 'parseInt', 'header1.css('padding-top')', 'header2.css('padding-top')', and 'scrollTop()'.

if (parseInt(header1.css('padding-top'), 10) > header1_initialDistance) {
 header1.css('padding-top', '' + \$(window).scrollTop());
}

if (\$(window).scrollTop() > header2_initialDistance) {
 header2.css('padding-top', '' + \$(window).scrollTop());
}

ROTEAMENTO E URLs

O ROTEAMENTO EM FLASK É FEITO COM A FUNÇÃO ROUTE(). VAMOS ADICIONAR MAIS ROTAS À NOSSA APLICAÇÃO.



```
1 @app.route('/about')
2 def about():
3     return 'About Page'
4
5 @app.route('/contact')
6 def contact():
7     return 'Contact Page'
```

TEMPLATES E JINJA2
OFLASK UTILIZA O MOTOR DE TEMPLATES
JINJA2 PARA RENDERIZAR HTML DINÂMICO.
CRIE UM DIRETÓRIO CHAMADO TEMPLATES
E ADICIONE UM ARQUIVO INDEX.HTML:



```
1 <!doctype html>
2 <html>
3   <head>
4     <title>Home Page</title>
5   </head>
6   <body>
7     <h1>{{ message }}</h1>
8   </body>
9 </html>
```

NO ARQUIVO 'APP.PY':

```
● ● ●  
1 from flask import render_template  
2  
3 @app.route('/')  
4 def home():  
5     return render_template('index.html', message='Hello, World!')
```

MANIPULANDO FORMULÁRIOS E DADOS DO USUÁRIO

VAMOS CRIAR UM FORMULÁRIO SIMPLES
PARA ENTENDER COMO MANIPULAR DADOS
DO USUÁRIO EM FLASK. ADICIONE UM
ARQUIVO FORM.HTML NO DIRETÓRIO
TEMPLATES:



```
1 <!doctype html>
2 <html>
3 <head>
4     <title>Form</title>
5 </head>
6 <body>
7     <form method="POST">
8         <input type="text" name="name" placeholder="Enter your name">
9         <input type="submit">
10    </form>
11 </body>
12 </html>
```

NO ARQUIVO 'APP.PY':

```
● ● ●  
1 from flask import request  
2  
3 @app.route('/form', methods=['GET', 'POST'])  
4 def form():  
5     if request.method == 'POST':  
6         name = request.form['name']  
7         return f'Hello, {name}!'  
8     return render_template('form.html')
```

GERENCIAMENTO DE ERROS E DEPURAÇÃO

FLASK OFERECE SUPORTE PARA TRATAMENTO DE ERROS E MENSAGENS DE DEPURAÇÃO. ADICIONE UMA ROTA PARA SIMULAR UM ERRO 404:



```
1 app.errorhandler(404)
2 def page_not_found(e):
3     return 'Page not found!', 404
```

CAPÍTULO 4: INTRODUÇÃO AO DJANGO

```
if ($window.scrollTop() > header1_initialDistance) {
    header1.css('padding-top', '' + $window.scrollTop());
}
if ($window.scrollTop() > header2_initialDistance) {
    header2.css('padding-top', '' + $window.scrollTop());
}
```

O QUE É O DJANGO E POR QUE USA-LO?

DJANGO É UM FRAMEWORK DE ALTO NÍVEL PARA DESENVOLVIMENTO WEB EM PYTHON, CONHECIDO POR SUA "BATERIA INCLUSA". ELE OFERECE UMA SÉRIE DE FERRAMENTAS E FUNCIONALIDADES INTEGRADAS QUE FACILITAM O DESENVOLVIMENTO DE APLICAÇÕES WEB COMPLEXAS.

CONFIGURANDO UM PROJETO DJANGO

1 - INSTALE O DJANGO



```
1 pip install django
```

2 - CRIE UM NOVO PROJETO:



```
1 django-admin startproject myproject  
2 cd myproject
```

3 - INICIE O SERVIDOR DE DESENVOLVIMENTO:



```
1 python manage.py runserver
```

ACESSE [HTTP://127.0.0.1:8000/](http://127.0.0.1:8000/) NO NAVEGADOR PARA VER A PÁGINA INICIAL DO DJANGO.

MODELOS, VIZUALIZAÇÕES E URLs

DJANGO SEGUE O PADRÃO MVT (MODEL-VIEW-TEMPLATE). VAMOS CRIAR UMA APLICAÇÃO SIMPLES PARA ENTENDER ESSES CONCEITOS.

1 - CRIE UMA APLICAÇÃO:



```
1 python manage.py startapp myapp
```

2 - DEFINA UM MODELO EM 'MYAPP/MODELS.PY':



```
1 from django.db import models
2
3 class Post(models.Model):
4     title = models.CharField(max_length=100)
5     content = models.TextField()
6     created_at = models.DateTimeField(auto_now_add=True)
```

3 - CRIE UMA VIEW EM 'MYAPP/VIEWS.PY':



```
1 from django.shortcuts import render
2 from .models import Post
3
4 def home(request):
5     posts = Post.objects.all()
6     return render(request, 'home.html', {'posts': posts})
```

4 - ADICIONE URLs EM 'MYAPP/URLS.PY':

```
● ● ●  
1 from django.urls import path  
2 from .views import home  
3  
4 urlpatterns = [  
5     path('', home, name='home'),  
6 ]
```

5 - INCLUA AS URLs DA APLIICAÇÃO NO 'MYPROJECT/URLS.PY':

```
● ● ●  
1 from django.contrib import admin  
2 from django.urls import path, include  
3  
4 urlpatterns = [  
5     path('admin/', admin.site.urls),  
6     path('', include('myapp.urls')),  
7 ]
```

6 - CRIE UM TEMPLATE 'HOME.HTML' NO DIRETÓRIO 'TEMPLATES':



```
1 <!doctype html>
2 <html>
3 <head>
4     <title>Form</title>
5 </head>
6 <body>
7     <form method="POST">
8         <input type="text" name="name"
9             placeholder="Enter your name">
10        <input type="submit">
11    </form>
12 </body>
13 </html>
```

TEMPLATES E STATIC FILES

OS TEMPLATES EM DJANGO SÃO ARQUIVOS HTML QUE PODEM INCLUIR CÓDIGO DJANGO TEMPLATE LANGUAGE (DTL). PARA SERVIR ARQUIVOS ESTÁTICOS, ADICIONE O SEGUINTE AO SETTINGS.PY:



```
1 STATIC_URL = '/static/'
```

CAPÍTULO 5: DESENVOLVENDO COM DJANGO

SISTEMAS DE ADMINISTRAÇÃO COM DJANGO

DJANGO VEM COM UM SISTEMA DE ADMINISTRAÇÃO PRONTO PARA USO. PARA USÁ-LO, REGISTRE SEUS MODELOS EM **MYAPP/ADMIN.PY**:



```
1 from django.contrib import admin
2 from .models import Post
3
4 admin.site.register(Post)
```

ACESSE [HTTP://127.0.0.1:8000/admin/](http://127.0.0.1:8000/admin/) E USE O ADMIN PARA GERENCIAR SEUS MODELOS.

FORMULÁRIOS E VALIDAÇÃO

DJANGO FACILITA O TRABALHO COM FORMULÁRIOS E VALIDAÇÃO. VAMOS CRIAR UM FORMULÁRIO SIMPLES PARA ADICIONAR NOVOS POSTS.

1 - CRIE UM FORMULÁRIO EM 'MYAPP/FORMS.PY':

```
● ● ●  
1 from django import forms  
2 from .models import Post  
3  
4 class PostForm(forms.ModelForm):  
5     class Meta:  
6         model = Post  
7         fields = ['title', 'content']
```

2 - ADICIONE UMA VIZUALIZAÇÃO PARA O FORMULÁRIO EM 'MYAPP/VIEWS.PY':



```
1 from .forms import PostForm
2
3 def new_post(request):
4     if request.method == 'POST':
5         form = PostForm(request.POST)
6         if form.is_valid():
7             form.save()
8             return redirect('home')
9     else:
10        form = PostForm()
11    return render(request, 'new_post.html', {'form': form})
```

FORMULÁRIOS E VALIDAÇÃO

DJANGO TEM UM SISTEMA DE AUTENTICAÇÃO ROBUSTO. VAMOS ADICIONAR FUNCIONALIDADES DE LOGIN E LOGOUT.

1 - ADICIONE URLs DE AUTENTICAÇÃO EM 'MYPROJECT/URLS.PY'



```
1 from django.contrib.auth import views as auth_views
2
3 urlpatterns = [
4     # ...
5     path('login/', auth_views.LoginView.as_view(), name='login'),
6     path('logout/', auth_views.LogoutView.as_view(), name='logout'),
7 ]
```

DEPLOYING UM APLICATIVO DJANGO

O DEPLOY DE UMA APLICAÇÃO DJANGO ENVOLVE A CONFIGURAÇÃO DE UM SERVIDOR DE PRODUÇÃO, COMO O GUNICORN, E UM SERVIDOR WEB, COMO O NGINX. AQUI ESTÃO OS PASSOS BÁSICOS:

1 - INSTALE O GUNICORN:

```
1 pip install gunicorn
```

2 - EXECUTE O GUNICORN

```
1 gunicorn myproject.wsgi
```

CAPÍTULO 6: AVANÇANDO COM PYTHON PARA WEB

APIS RESTFUL COM FLASK E DJANGO

APIS RESTFUL PERMITEM A COMUNICAÇÃO ENTRE SISTEMAS. VAMOS CRIAR UMA SIMPLES API RESTFUL COM FLASK E DJANGO.

COM FLASK:

```
● ● ●  
1  from flask import Flask, jsonify, request  
2  
3  app = Flask(__name__)  
4  
5  posts = []  
6  
7  @app.route('/api/posts', methods=['GET'])  
8  def get_posts():  
9      return jsonify(posts)  
10  
11 @app.route('/api/posts', methods=['POST'])  
12 def add_post():  
13     post = request.get_json()  
14     posts.append(post)  
15     return jsonify(post), 201  
16  
17 if __name__ == '__main__':  
18     app.run(debug=True)
```

COM DJANGO:

1 - INSTALE O DJANGO REST FRAMEWORK

```
1 pip install djangorestframework
```

2 - ADICIONE 'REST_FRAMEWORK' A 'INSTALLED_APPS' EM 'SETTINGS.PY':

```
1 INSTALLED_APPS = [
2     # ...
3     'rest_framework',
4 ]
```

3 - CRIE UMA API EM 'MYAPP/VIEWS.PY':

```
1 from rest_framework import viewsets
2 from .models import Post
3 from .serializers import PostSerializer
4
5 class PostViewSet(viewsets.ModelViewSet):
6     queryset = Post.objects.all()
7     serializer_class = PostSerializer
```

4 - DEFINA UM ROTEADOR EM 'MYAPP/URLS.PY':

```
● ● ●  
1 from rest_framework import routers  
2 from .views import PostViewSet  
3  
4 router = routers.DefaultRouter()  
5 router.register(r'posts', PostViewSet)  
6  
7 urlpatterns = router.urls
```

WEBSOCKETS E APLICAÇÕES EM TEMPO REAL

WEBSOCKETS PERMITEM A COMUNICAÇÃO BIDIRECIONAL EM TEMPO REAL. PARA USÁ-LOS COM DJANGO, VOCÊ PODE UTILIZAR DJANGO CHANNELS.

1 - INSTALE O DJANGO CHANNELS:

```
● ● ●  
1 pip install channels
```

2- ADICIONE CHANNELS A INSTALLED_APPS E CONFIGURE ASGI_APPLICATION EM SETTINGS.PY:

```
● ● ●  
1 INSTALLED_APPS = [  
2     # ...  
3     'channels',  
4 ]  
5  
6 ASGI_APPLICATION = 'myproject.asgi.application'
```

3 - CONFIGURE O ROTEAMENTO EM MYPROJECT/ASGI.PY:

```
● ● ●  
1 import os  
2 from channels.routing import ProtocolTypeRouter, URLRouter  
3 from channels.auth import AuthMiddlewareStack  
4 from django.core.asgi import get_asgi_application  
5 from myapp import routing  
6  
7 os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'myproject.settings')  
8  
9 application = ProtocolTypeRouter({  
10     "http": get_asgi_application(),  
11     "websocket": AuthMiddlewareStack(  
12         URLRouter(  
13             routing.websocket_urlpatterns  
14         )  
15     ),  
16 })
```

4 - DEFINA ROTAS DE WEBSOCKET EM 'MYAPP/ROUTING.PY':



```
1 from django.urls import re_path
2 from . import consumers
3
4 websocket_urlpatterns = [
5     re_path(r'ws/somepath/$', consumers.MyConsumer.as_asgi()),
6 ]
```

5 - CRIE UM CONSUMER EM 'MYAPP/CONSUMERS.PY':



```
1 import json
2 from channels.generic.websocket import WebsocketConsumer
3
4 class MyConsumer(WebsocketConsumer):
5     def connect(self):
6         self.accept()
7
8     def disconnect(self, close_code):
9         pass
10
11    def receive(self, text_data):
12        data = json.loads(text_data)
13        self.send(text_data=json.dumps({
14            'message': data['message']
15        }))
```

TESTES E QUALIDADE DE CÓDIGO

ESCREVER TESTES É CRUCIAL PARA MANTER A QUALIDADE DO CÓDIGO. TANTO FLASK QUANTO DJANGO SUPORTAM TESTES DE UNIDADE E TESTES FUNCIONAIS.

TESTES COM FLASK:



```
1 import unittest
2 from app import app
3
4 class FlaskTestCase(unittest.TestCase):
5     def test_home(self):
6         tester = app.test_client(self)
7         response = tester.get('/')
8         self.assertEqual(response.status_code, 200)
9         self.assertEqual(response.data, b'Hello, World!')
10
11 if __name__ == '__main__':
12     unittest.main()
```

TESTES COM DJANGO:



```
1 from django.test import TestCase
2 from .models import Post
3
4 class PostModelTest(TestCase):
5     def setUp(self):
6         Post.objects.create(title='Test', content='This is a test post.')
7
8     def test_post_content(self):
9         post = Post.objects.get(id=1)
10        self.assertEqual(post.title, 'Test')
11        self.assertEqual(post.content, 'This is a test post.')
```

Conclusão!!

Este eBook oferece uma introdução abrangente ao desenvolvimento web com Python, utilizando Flask e Django. Com esses conhecimentos, você estará pronto para construir suas próprias aplicações web e continuar explorando o mundo do desenvolvimento com Python.

