O desenvolvimento web com Python Integra a versatilidade da linguagem às necessidades dinâmicas da internet contemporânea
Python é proeminente do desenvolvimento web, conhecido por sua legibilidade e aplicabilidade em ambas as camadas (front-end e back-end), permitindo a criação eficaz de aplicativos robustos.
No início, as páginas web eram estáticas, mas a demanda por interfaces dinâmicas impulsionou o desenvolvimento de linguagens e frameworks web, onde Python se destacou, principalmente no back-end, com frameworks com Django e Flask
Python é versátil no desenvolvimento web, sendo aplicado tanto no front-end quanto no back-end, permitindo uma integração eficiente entre interface de usuário e lógica de processamento
Oferece frameworks robustos como Django e flexíveis como Flask para back-end, além de suportar a integração fácil com bibliotecas front-end populares como React, Vue.js e Angular
Com FastAPI e Django Rest Framework, destaca-se em criar APIs eficientes e bem documentadas para facilitar a comunicação entre front-end e back-end
Python se integra a tendências web modernas, como arquiteturas sem servidor e containers, mantendo os desenvolvedores atualizados
desenvolvimento web em Python enfrenta desafios, como escalabilidade e segurança, mas oferecer oportunidade para inovação
Python destaca-se no desenvolvimento web, oferecendo ferramentas poderosas e sendo uma escolha excelente com uma comunidade ativa

Back-end
O back-end de uma aplicação web lida com a lógica, o processamento de dados e a interação com o servidor
O Flask é um framework web leve para Python, podemos usá-lo para criar um exemplo básico de servidor back-end.

Front-end
O front-end de uma aplicação web é a interface com o qual os usuários interagem diretamente
O HTML é a espinha dorsal do conteúdo web
pip install flask

```python
from flask import Flask

#Criando da aplicação Flask
app = Flask(__name__)

#Rota para a página inicial
@app.route('/')
def hello():
    return 'Bem vindo ao back-end simples com Flash!'

# Executa a aplicação no host e na porta expecificados
if __name__ == '__main__':
    app.run(host='localhost', port=5000)
```

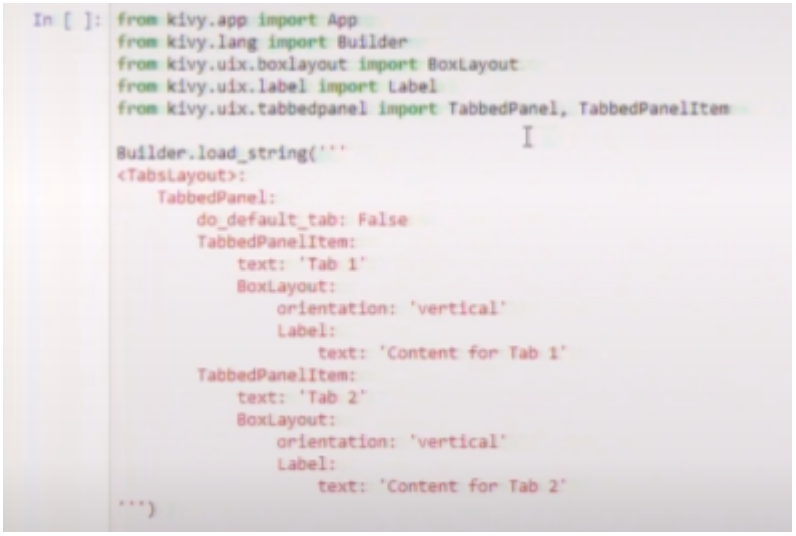
```
 * Serving Flask app '__main__'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a pr
oduction WSGI server instead.
 * Running on http://localhost:5000
Press CTRL+C to quit
127.0.0.1 - - [29/Apr/2025 10:53:07] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Apr/2025 10:53:08] "GET /favicon.ico HTTP/1.1" 404 -
```

O desenvolvimento mobile, tradicionalmente associado à linguagem nativas, como Swift e Kotlin, tem visto um aumento de popularidade do Python devido à sua versatilidade e simplicidade
Python, com sua sintaxe clara, destaca-se no desenvolvimento mobile, aproveitando frameworks com Kivy, BeeWare e Flutter, que oferecem suporte à linguagem
O Kivy destaca-se no desenvolvimento mobile em Python, oferecendo uma estrutura robusta e multi-touch para criação de aplicativos visualmente impressionantes, com suporte para diversas plataformas
A escolha de Python no desenvolvimento mobile oferece vantagens, como uma comunidade ativa, vasta biblioteca de módulos e agilidade proporcionada pela linguagem
Desenvolvedores devem estar cientes de desafios, como desempenho comparado a linguagens nativas e possíveis limitações de acesso a recursos específicos do dispositivo.
O uso de Python no desenvolvimento mobile permite que desenvolvedores familiarizados com a linguagem expandam para o universo mobile. A escolha entre Python e outra linguagens dependerá dos requisitos do projeto e das preferências da equipe, com frameworks robustos disponíveis para facilitar o desenvolvimento.

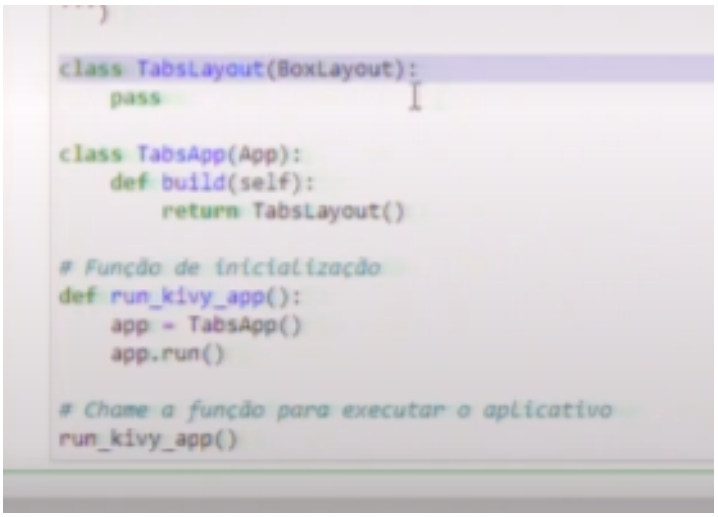
KivyMD - é uma extensão de Kivy, um framework Python para o desenvolvimento de aplicativos multi-touch. O KivyMD estende as capacidades do Kivy ao integrar os princípios de design do material Design do Google, proporcionando uma experiência atraente e consistente em dispositivos móveis
Adota os padrões de design do Material Design, oferecendo uma interface de usuário moderna, consistente e intuitiva. Isso inclui elementos como botões flutuantes, barras de navegação e efeitos de transição
Assim como o Kivy, o KivyMD é multiplataforma, permitindo que os aplicativos sejam executados em diferentes sistemas operacionais, incluindo iOS, Windows, Linux e macOS

Widgets - desempenham um papel fundamental no desenvolvimento de interfaces de usuário interativas. No contextos do KivuMD (Material Design Components for Kivy), os widgets são os blocos de construção essenciais para criar aplicativos visualmente atraentes

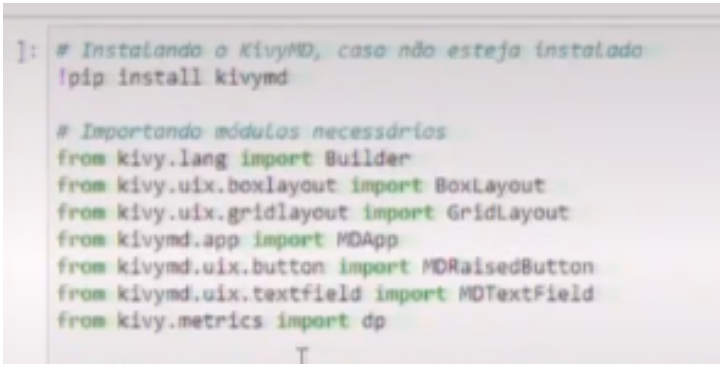
MDTabs é um widget poderoso do KivyMD projetado para facilitar a organização de conteúdo em diferentes abas. Isso é particularmente útil quando você precisa exibir várias seções ou funcionalidades em um aplicativo sem sobrecarregar a tela principal.



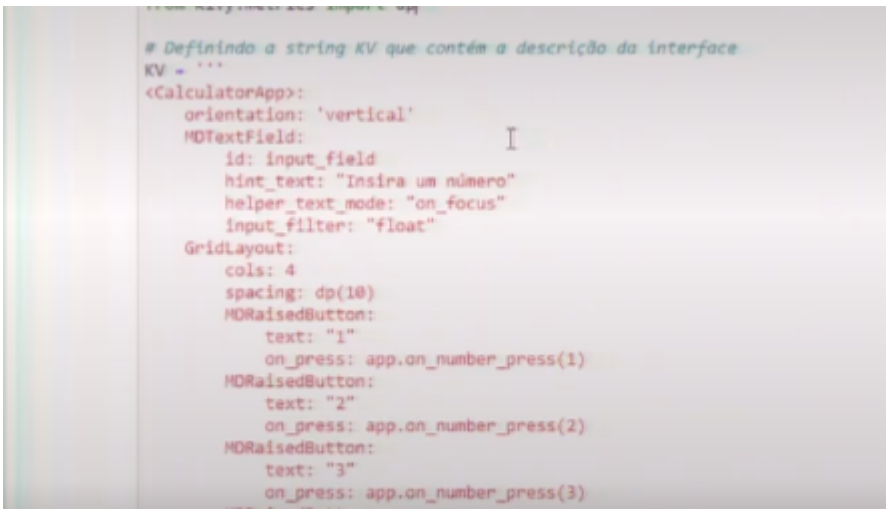
```
In [ ]: from kivy.app import App
        from kivy.lang import Builder
        from kivy.uix.boxlayout import BoxLayout
        from kivy.uix.label import Label
        from kivy.uix.tabbedpanel import TabbedPanel, TabbedPanelItem

        Builder.load_string('''
        <TabsLayout>:
            TabbedPanel:
                do_default_tab: False
                TabbedPanelItem:
                    text: 'Tab 1'
                    BoxLayout:
                        orientation: 'vertical'
                        Label:
                            text: 'Content for Tab 1'
                TabbedPanelItem:
                    text: 'Tab 2'
                    BoxLayout:
                        orientation: 'vertical'
                        Label:
                            text: 'Content for Tab 2'
        ''')
```



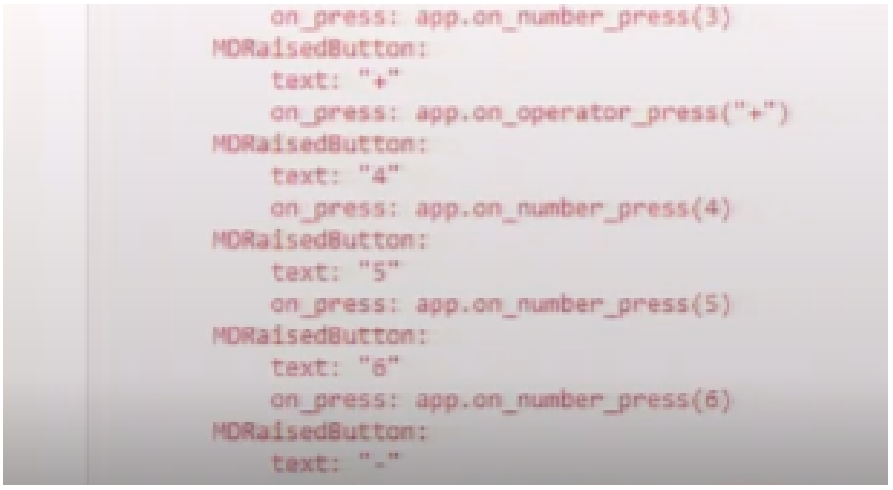
```
class TabsLayout(BoxLayout):
    pass

class TabsApp(App):
    def build(self):
        return TabsLayout()

# Função de inicialização
def run_kivy_app():
    app = TabsApp()
    app.run()

# Chame a função para executar o aplicativo
run_kivy_app()
```



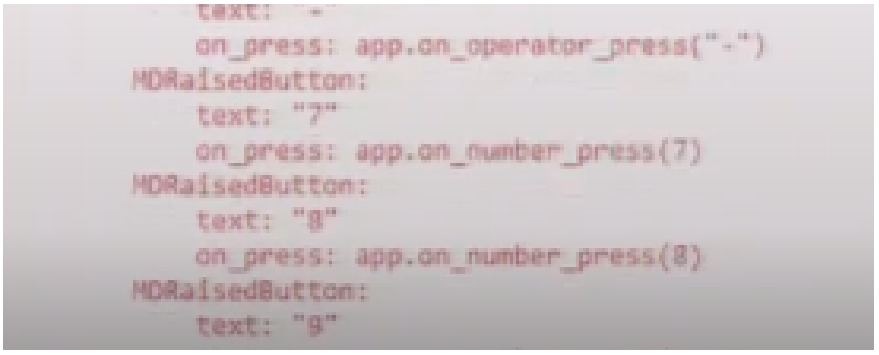
```
]: # Instalando o KivyMD, caso não esteja instalado
   !pip install kivymd

   # Importando módulos necessários
   from kivy.lang import Builder
   from kivy.uix.boxlayout import BoxLayout
   from kivy.uix.gridlayout import GridLayout
   from kivymd.app import MDApp
   from kivymd.uix.button import MDRaisedButton
   from kivymd.uix.textfield import MDTextField
   from kivy.metrics import dp
```



```
# Definindo a string KV que contém a descrição da interface
KV = '''
<CalculatorApp>:
    orientation: 'vertical'
    MDTextField:
        id: input_field
        hint_text: "Insira um número"
        helper_text_mode: "on_focus"
        input_filter: "float"
    GridLayout:
        cols: 4
        spacing: dp(10)
        MDRaisedButton:
            text: "1"
            on_press: app.on_number_press(1)
        MDRaisedButton:
            text: "2"
            on_press: app.on_number_press(2)
        MDRaisedButton:
            text: "3"
            on_press: app.on_number_press(3)
```



```
            on_press: app.on_number_press(3)
        MDRaisedButton:
            text: "+"
            on_press: app.on_operator_press("+")
        MDRaisedButton:
            text: "4"
            on_press: app.on_number_press(4)
        MDRaisedButton:
            text: "5"
            on_press: app.on_number_press(5)
        MDRaisedButton:
            text: "6"
            on_press: app.on_number_press(6)
        MDRaisedButton:
            text: "-"
```

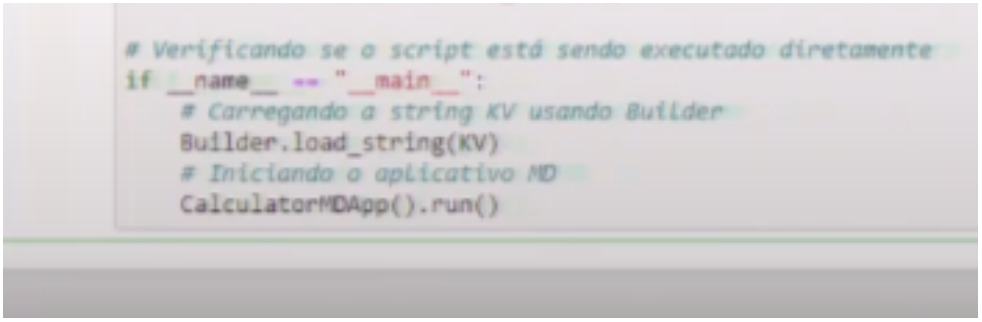


```
            text: "-"
            on_press: app.on_operator_press("-")
        MDRaisedButton:
            text: "7"
            on_press: app.on_number_press(7)
        MDRaisedButton:
            text: "8"
            on_press: app.on_number_press(8)
        MDRaisedButton:
            text: "9"
```

```
            text: "9"
            on_press: app.on_number_press(9)
        MDRaisedButton:
            text: "*"
            on_press: app.on_operator_press("*")
        MDRaisedButton:
            text: "C"
            on_press: app.clear_input()
        MDRaisedButton:
            text: "0"
            on_press: app.on_number_press(0)
        MDRaisedButton:
            text: "="
            on_press: app.calculate_result()
        MDRaisedButton:
            text: "/"
            on_press: app.on_operator_press("/")
...
```

```python
# Definindo a classe CalculatorApp que herda de BoxLayout
class CalculatorApp(BoxLayout):
    # Método chamado quando um número é pressionado
    def on_number_press(self, number):
        current_text = self.ids.input_field.text
        new_text = f"{current_text}{number}"
        self.ids.input_field.text = new_text

    # Método chamado quando um operador é pressionado
    def on_operator_press(self, operator):
        current_text = self.ids.input_field.text
        new_text = f"{current_text} {operator} "
        self.ids.input_field.text = new_text

    # Método chamado para limpar a entrada
    def clear_input(self):
        self.ids.input_field.text = ""

    # Método chamado para calcular o resultado da expressão
    def calculate_result(self):
        try:
            result = eval(self.ids.input_field.text)
            self.ids.input_field.text = str(result)
        except Exception as e:
            self.ids.input_field.text = "Erro"
```

```python
# Definindo a classe CalculatorMDApp que herda de MDApp
class CalculatorMDApp(MDApp):
    # Método chamado para construir o aplicativo
    def build(self):
        return CalculatorApp()

    # Métodos chamados para interagir com a instância de CalculatorApp
    def on_number_press(self, number):
        self.root.on_number_press(number)

    def on_operator_press(self, operator):
        self.root.on_operator_press(operator)

    def clear_input(self):
        self.root.clear_input()

    def calculate_result(self):
        self.root.calculate_result()
```



```
# Verificando se o script está sendo executado diretamente
if __name__ == "__main__":
    # Carregando a string KV usando Builder
    Builder.load_string(KV)
    # Iniciando o aplicativo MD
    CalculatorMDApp().run()
```

Teste no Python

Assertions - As Assertions são expressões utilizadas para verificar as condições de verdade durante a execução do código. Elas são fundamentais para a detecção precoce de erros, garantindo que as suposições sobre o comportamento do programa sejam atendidas.

Doctest - O doctest é um módulo em Python que permite incorporar testes diretamente na documentação do código, aproveitando os exemplos presentes na documentação para verificar se o código funciona conforme o esperado

Unittest - O módulo unittest oferece uma estrutura de teste mais avançada, permitindo a organização de testes em classes e métodos, além de fornecer assertions mais poderosos.

Machine Learning (Aprendizado de Máquina) é uma área da inteligência artificial que visa desenvolver algoritmos capazes de aprender com dados e realizar tarefas sem programação explícita
Envolve abordagens como aprendizado supervisionado, não supervisionado e por esforço
Suas aplicações abrangem desde reconhecimento de imagem até tomada de decisões complexas em diversos setores
É uma área em constante evolução impulsionada por avanços tecnológicos e disponibilidade de dados

Modelos

Árvores de Decisão: Modelo que toma decisões com base em condições
Redes Neurais: inspiradas no funcionamento do cérebro, são usadas para problemas complexos.
SVM (Support Vector Machines): Para classificação e regressão
K-Means: Algoritmo de agrupamento usado no aprendizado não supervisionado

Os tipos de treinamento são estratégias fundamentais no desenvolvimento de modelos de Machine Learning
No treinamento supervisionado, o modelo é alimentado com um conjunto de dados rotulado, consistindo em pares de entrada e saída esperada. exemplo prático: classificação de e-mails como “spam” ou “não spam”

Ao contrário do supervisionado, o treinamento não supervisionado lida com dados não rotulados. Exemplo prático: agrupamento de clientes com base em padrões de compra
O treinamento por reforço envolve um agente interagindo com um ambiente dinâmico. Exemplo prático: treinar um agente de IA para jogar um jogo e ganhar pontos ao realizar ações corretivas

TensorFlow é uma biblioteca de código aberto desenvolvida pela Google que facilita a implementação de modelos de Machine Learning e Deep Learning. Sua estrutura flexível permite a criação e treinamento de modelos complexos, sendo amplamente utilizada na comunidade de aprendizado de máquina.