# PARA DIZER QUE SABE PYTHON TEM QUE APRENDER ESTES 10 TEMAS

1. **- Textos, Números e Variáveis**

## Variáveis

# Definindo variáveis

texto = "Olá, Mundo!" inteiro = 42

flutuante = 3.14

print(texto, inteiro, flutuante)

## Strings

# Trabalhando com strings nome = "Alice"

saudacao = "Olá, " + nome + "!" print(saudacao)

## Inteiros

# Trabalhando com inteiros idade = 30

ano\_atual = 2024

ano\_nascimento = ano\_atual - idade print(ano\_nascimento)

## Floats

# Trabalhando com floats pi = 3.14159

raio = 5

area\_circulo = pi \* (raio \*\* 2) print(area\_circulo)

# - Operações com Elementos Básicos

## Operações Matemáticas

# Operações básicas soma = 10 + 5

subtracao = 10 - 5

multiplicacao = 10 \* 5

divisao = 10 / 5

print(soma, subtracao, multiplicacao, divisao)

## Substituir um texto por um número

# Convertendo string para número numero\_texto = "100"

numero = int(numero\_texto) print(numero)

## Concatenar textos

# Concatenando strings texto1 = "Bom"

texto2 = " dia!"

resultado = texto1 + texto2 print(resultado)

## Potenciação

# Potenciação base = 2

expoente = 3

potencia = base \*\* expoente print(potencia)

## Porcentagem

# Calculando porcentagem total = 200

percentual = 15

porcentagem = (percentual / 100) \* total print(porcentagem)

# - Condicionais e Comparações

## If

# Estrutura condicional if idade = 20

if idade >= 18: print("Maior de idade")

## Else

# Estrutura condicional else idade = 16

if idade >= 18: print("Maior de idade")

else:

print("Menor de idade")

# - Estrutura de Repetição

## For e While

# Loop for

for i in range(5): print(i)

# Loop while contador = 0

while contador < 5: print(contador) contador += 1

## Analisar todos os funcionários

# Iterando sobre uma lista de funcionários funcionarios = ["Alice", "Bob", "Carol"] for funcionario in funcionarios:

print(funcionario)

## Analisar todos os meses de vendas

# Iterando sobre uma lista de meses

meses = ["Janeiro", "Fevereiro", "Março"]

for mes in meses: print(mes)

## Analisar todos os preços dos produtos

# Iterando sobre uma lista de preços precos = [10.99, 20.99, 30.99]

for preco in precos: print(preco)

# - Iterables

## Listas

# Trabalhando com listas lista = [1, 2, 3, 4, 5]

print(lista)

## Tuplas

# Trabalhando com tuplas tupla = (1, 2, 3) print(tupla)

## Dicionários

# Trabalhando com dicionários

dicionario = {"nome": "Alice", "idade": 30} print(dicionario)

## Sets

# Trabalhando com sets conjunto = {1, 2, 3, 4, 5} print(conjunto)

# - Bibliotecas & Módulos

## Saber importar

# Importando bibliotecas import math

import numpy as np import pandas as pd

## Ler a documentação

# Ler a documentação é essencial

# Por exemplo, a documentação do Pandas: https://pandas.pydata.org/pandas-docs/stable/

## Exemplo

# Análise de dados com Pandas e NumPy import numpy as np

import pandas as pd

dados = {'coluna1': [1, 2, 3], 'coluna2': [4, 5, 6]} df = pd.DataFrame(dados)

print(df)

# - Aprender a Ler uma Documentação / API

## APIs prontas

# Utilizando uma API existente import requests

resposta = requests.get('https://api.exchangerate-api.com/v4/latest/USD') dados = resposta.json()

print(dados)

## Interpretar documentação

# Interpretação de documentação é essencial para trabalhar com APIs # Exemplo: https://requests.readthedocs.io/en/latest/

# - Funções, Expressões Lambda e List Comprehension

## Funções

# Definindo uma função def saudacao(nome):

return f"Olá, {nome}!" print(saudacao("Alice"))

## Expressões Lambda

# Funções lambda

soma = lambda a, b: a + b print(soma(3, 4))

## List Comprehension

# List comprehension

quadrados = [x \*\* 2 for x in range(10)] print(quadrados)

# e 10 - Áreas Específicas

## Ciência de Dados

### Pandas, NumPy e Bibliotecas de Gráficos

# Análise de dados com Pandas e NumPy import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dados = np.random.rand(10, 3)

df = pd.DataFrame(dados, columns=['A', 'B', 'C']) df.plot(kind='bar')

plt.show()

### Obtenção e Tratamento de Dados

# Obtenção e tratamento de dados import pandas as pd

df = pd.read\_csv('dados.csv') df.fillna(0, inplace=True)

print(df)

## Desenvolvimento Web

### Orientação a Objetos

# Fundamentos de orientação a objetos class Pessoa:

def init (self, nome, idade): self.nome = nome

self.idade = idade

def saudacao(self):

return f"Olá, meu nome é {self.nome} e tenho {self.idade} anos."

pessoa = Pessoa("Alice", 30) print(pessoa.saudacao())

### Integração Backend Frontend

# Uso de Flask para desenvolvimento web from flask import Flask, render\_template

app = Flask( name ) @app.route('/')

def home():

return "Olá, Mundo!"

if name == ' main ': app.run(debug=True)

## Automação

### Trabalhar com Arquivos, Caminhos, Tabelas

# Manipulação de arquivos import os

caminho = 'caminho/para/arquivo.txt' with open(caminho, 'r') as arquivo:

conteudo = arquivo.read() print(conteudo)

### Ferramentas de Automação

# Web Scraping com BeautifulSoup import requests

from bs4 import BeautifulSoup

url = 'https://example.com' resposta = requests.get(url)

soup = BeautifulSoup(resposta.content, 'html.parser') print(soup.title.text)

Para fazer a conexão com um banco de dados usando Python em um ambiente de desenvolvimento backend e frontend, você pode usar o framework Flask para criar uma aplicação web simples. Vou mostrar um exemplo usando Flask e SQLite, que é um banco de dados leve e embutido. O exemplo inclui um backend básico que conecta ao banco de dados e uma interface frontend para interagir com os dados.

### Backend com Flask e SQLite

1. **Instalação das Dependências** Primeiro, instale as bibliotecas necessárias:

pip install flask flask\_sqlalchemy

1. **Código do Backend**

python

Copiar código

from flask import Flask, render\_template, request, redirect, url\_for

from flask\_sqlalchemy import SQLAlchemy

app = Flask(\_\_name\_\_)

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///database.db'

db = SQLAlchemy(app)

class User(db.Model):

id = db.Column(db.Integer, primary\_key=True)

name = db.Column(db.String(50), nullable=False)

email = db.Column(db.String(50), nullable=False)

def \_\_repr\_\_(self):

return f'<User {self.name}>'

@app.route('/')

def index():

users = User.query.all()

return render\_template('index.html', users=users)

@app.route('/add', methods=['POST'])

def add\_user():

name = request.form.get('name')

email = request.form.get('email')

new\_user = User(name=name, email=email)

db.session.add(new\_user)

db.session.commit()

return redirect(url\_for('index'))

if \_\_name\_\_ == '\_\_main\_\_':

db.create\_all()

app.run(debug=True)

1. **Frontend (Templates)** Crie uma pasta chamada templates no mesmo diretório do script acima e adicione um arquivo chamado index.html com o seguinte conteúdo:

<!DOCTYPE html>

<html>

<head>

<title>Users</title>

</head>

<body>

<h1>Users</h1>

<form action="/add" method="post">

<input type="text" name="name" placeholder="Name" required>

<input type="email" name="email" placeholder="Email" required>

<button type="submit">Add User</button>

</form>

<ul>

{% for user in users %}

<li>{{ user.name }} - {{ user.email }}</li>

{% endfor %}

</ul>

</body>

</html>

### Explicação do Código

1. **Backend**
   * A aplicação Flask é configurada com uma URI de banco de dados SQLite (sqlite:///database.db).
   * A classe User define um modelo de tabela com colunas para id, name e email.
   * A rota / renderiza uma lista de usuários armazenados no banco de dados.
   * A rota /add permite adicionar novos usuários ao banco de dados via formulário.
2. **Frontend**
   * O template index.html inclui um formulário para adicionar novos usuários e uma lista para exibir os usuários existentes.

### Executando a Aplicação

1. Salve o código do backend em um arquivo chamado app.py.
2. Crie a pasta templates e adicione o arquivo index.html conforme mostrado.
3. Execute o backend com o comando:

bash

Copiar código

python app.py

1. Abra o navegador e acesse http://127.0.0.1:5000/ para ver a interface web.