

VOCÇÃO



Código da Transformação

Back-end

Prefácio

Bem-vindo(a) ao **Código da Transformação - Python** 🚀🐍!

Este curso foi cuidadosamente planejado para guiá-lo(a) em uma jornada de aprendizado que vai desde os conceitos mais básicos da programação até o desenvolvimento de aplicações completas e funcionais. Seja você um(a) iniciante curioso(a) ou alguém que deseja aprimorar suas habilidades, este curso oferece um caminho claro e prático para dominar a linguagem Python e suas ferramentas essenciais.

🐍 Por que Python?

Python é uma das linguagens de programação mais populares e versáteis do mundo. Sua sintaxe simples e legível, combinada com uma vasta gama de bibliotecas e frameworks, faz dela a escolha ideal tanto para iniciantes quanto para profissionais experientes.

Com Python, você poderá desenvolver desde scripts simples até sistemas complexos, como:

- ✓ Aplicações web
 - ✓ Análise de dados
 - ✓ Automação de tarefas
 - ✓ Inteligência artificial
 - ✓ Machine learning
 - ✓ E muito mais!
-

📚 O que você vai aprender?

Este curso está dividido em **15 módulos**, cada um projetado para construir uma base sólida e progressiva de conhecimento.

Você começará com os fundamentos da programação, passando por tópicos como:

- ✓ Lógica de programação
- ✓ Estruturas de dados
- ✓ Funções
- ✓ Manipulação de arquivos
- ✓ Programação orientada a objetos (POO)

Em seguida, avançará para temas mais complexos, incluindo:

- ✓ Desenvolvimento de APIs com Flask e Django
- ✓ Integração com bancos de dados
- ✓ Testes automatizados

- ✓ Metodologias ágeis
- ✓ Git e GitHub para versionamento e colaboração

Por fim, você consolidará todo o conhecimento adquirido em um **Projeto Final**, onde criará uma aplicação completa do zero — simulando um ambiente profissional de desenvolvimento.

Metodologia Prática

Acreditamos que a melhor maneira de aprender a programar é colocando a mão na massa.

Por isso, cada módulo inclui **projetos práticos** que permitem aplicar imediatamente os conceitos aprendidos.

Além disso, você será introduzido(a) a ferramentas essenciais como:

- ✓ **Git e GitHub** – para versionamento de código e colaboração.
 - ✓ **Metodologias Ágeis** – para gerenciar suas tarefas e projetos de forma eficiente.
-

Para quem é este curso?

Iniciantes:

Se você nunca programou antes, este curso é para você! Começaremos do zero, explicando cada conceito de forma clara e acessível.

Estudantes e Profissionais:

Se você já tem alguma experiência, mas deseja aprofundar seus conhecimentos em Python e desenvolvimento de software, encontrará aqui um conteúdo rico e desafiador.

Entusiastas de Tecnologia:

Se você é apaixonado(a) por tecnologia e quer explorar o mundo da programação, este curso será um guia completo para transformar sua curiosidade em habilidades concretas.

Nosso Compromisso

Nosso objetivo não é apenas ensinar Python, mas também capacitar você a **pensar como um(a) desenvolvedor(a)**.

Queremos que você conclua este curso com a confiança e as habilidades necessárias para enfrentar desafios reais, seja no mercado de trabalho ou em projetos pessoais.



Prepare-se para Transformar sua Carreira!

Ao final desta jornada, você terá:

- ✓ Construído um portfólio de projetos práticos.
- ✓ Dominado ferramentas essenciais de desenvolvimento.
- ✓ Desenvolvido uma mentalidade ágil e colaborativa.
- ✓ Criado soluções reais usando Python e frameworks modernos.

Agora é com você! 💪

Dedique-se, pratique bastante e, acima de tudo, **divirta-se no processo**. A programação é uma habilidade poderosa, e estamos animados para ver o que você vai criar.

Boa sorte e bem-vindo(a) à sua transformação!  



Sumário do Curso

♦ Módulo 1: Introdução ao Git, GitHub e Metodologias Ágeis

Parte 1: Git e GitHub

- ✓ Introdução ao versionamento de código.
- ✓ Configuração inicial do Git.
- ✓ Comandos fundamentais: init, add, commit, status, log.
- ✓ Branches e controle de versão paralelo.
- ✓ Conectando-se ao GitHub: remote, push, pull, clone.
- ✓ Boas práticas de versionamento.

Parte 2: Metodologias Ágeis

- ✓ Introdução ao Manifesto Ágil.
 - ✓ **Scrum**: Papéis, cerimônias e artefatos.
 - ✓ **Kanban**: Quadros visuais e fluxo contínuo.
 - ✓ Aplicação no curso com **GitHub Projects**.
 - ✓ **Projeto Prático**:
 - Criar um repositório no GitHub e gerenciar tarefas com um quadro Kanban.
-

♦ Módulo 2: Introdução ao Python

- ✓ O que é Python e suas aplicações.
 - ✓ Instalação e configuração do ambiente.
 - ✓ Primeiros passos: sintaxe básica, print(), input(), variáveis e tipos de dados.
 - ✓ **Projeto Prático**:
 - Criar um programa que exibe uma mensagem personalizada.
-

♦ Módulo 3: Lógica de Programação com Python

- ✓ Operadores aritméticos, relacionais e lógicos.
 - ✓ Condicionais: if, elif, else.
 - ✓ Estruturas de repetição: for, while.
 - ✓ Manipulação de strings.
 - ✓ **Projeto Prático**:
 - Desenvolver uma calculadora simples.
-

♦ Módulo 4: Estruturas de Dados

- ✓ **Listas**: Criação, manipulação e métodos.
- ✓ **Tuplas**: Características e uso.
- ✓ **Dicionários**: Pares chave-valor e métodos.
- ✓ **Conjuntos**: Operações de união, interseção e diferença.

- ✓ **Projeto Prático:**
 - Criar um sistema de gerenciamento de lista de tarefas.
-

♦ **Módulo 5: Funções em Python**

- ✓ Definição e chamadas de funções.
 - ✓ Parâmetros e retorno de valores.
 - ✓ Argumentos posicionais e nomeados.
 - ✓ Escopo de variáveis.
 - ✓ Funções anônimas (lambda).
 - ✓ **Projeto Prático:**
 - Criar um programa de cálculo de notas.
-

♦ **Módulo 6: Manipulação de Arquivos**

- ✓ Abertura, leitura e escrita de arquivos.
 - ✓ Manipulação de arquivos CSV e JSON.
 - ✓ Gerenciamento de erros ao manipular arquivos.
 - ✓ **Projeto Prático:**
 - Criar um sistema para registrar e consultar dados de clientes em um arquivo JSON.
-

♦ **Módulo 7: Módulos e Pacotes**

- ✓ Importação de módulos: import, from ... import.
 - ✓ Criação de pacotes.
 - ✓ Introdução ao pip e instalação de bibliotecas externas.
 - ✓ Uso de bibliotecas populares: math, random, datetime.
 - ✓ **Projeto Prático:**
 - Criar um gerador de senhas seguras.
-

♦ **Módulo 8: Programação Orientada a Objetos (POO)**

- ✓ Conceitos básicos: classes, objetos, atributos e métodos.
 - ✓ Encapsulamento, herança e polimorfismo.
 - ✓ Métodos especiais: __init__, __str__.
 - ✓ **Projeto Prático:**
 - Criar um sistema de biblioteca para gerenciamento de livros.
-

♦ **Módulo 9: Tratamento de Erros**

- ✓ Blocos try, except, else, finally.
- ✓ Criação de exceções personalizadas.
- ✓ Exemplos de erros comuns e como tratá-los.

- ✓ **Projeto Prático:**
 - ➡ Melhorar um sistema de cadastro de usuários com validação de entradas.
-

♦ **Módulo 10: Introdução às APIs**

- ✓ O que são APIs e métodos HTTP.
 - ✓ Consumo de APIs usando a biblioteca requests.
 - ✓ Tratamento de respostas em JSON.
 - ✓ **Projeto Prático:**
 - ➡ Criar um programa que exibe dados de previsão do tempo consumindo uma API pública.
-

♦ **Módulo 11: Banco de Dados com PostgreSQL**

- ✓ Introdução a bancos de dados relacionais.
 - ✓ Conexão com PostgreSQL usando a biblioteca psycopg2.
 - ✓ Criação de tabelas e inserção de dados.
 - ✓ Consultas e manipulação de dados com SQL.
 - ✓ **Projeto Prático:**
 - ➡ Criar um sistema de gerenciamento de vendas com PostgreSQL.
-

♦ **Módulo 12: Testes Automatizados**

- ✓ Introdução ao módulo unittest.
 - ✓ Criação de casos de teste.
 - ✓ Validação de entradas e saídas de funções.
 - ✓ **Projeto Prático:**
 - ➡ Implementar testes para validar um sistema de cálculo de frete.
-

♦ **Módulo 13: Desenvolvimento de APIs com Flask**

- ✓ Configuração básica do Flask.
 - ✓ Criação de rotas e manipulação de requisições HTTP.
 - ✓ Retorno de dados no formato JSON.
 - ✓ **Projeto Prático:**
 - ➡ Criar uma API para cadastro e consulta de produtos.
-

♦ **Módulo 14: Introdução ao Django**

- ✓ Configuração de projetos Django.
- ✓ Criação de modelos, views e templates.
- ✓ Configuração de rotas e administração.
- ✓ **Projeto Prático:**
- ➡ Criar um blog básico com Django.

♦ Módulo 15: Projeto Final

✓ **Objetivo:** Consolidar o aprendizado com um projeto integrado.

✓ **Descrição:**

→ Criar uma aplicação completa usando Flask ou Django.

→ Implementar funcionalidades como cadastro, login, manipulação de dados e consumo de APIs externas.

✓ **Entrega:**

→ Apresentação do projeto em sala ou upload em repositório GitHub.

Módulo 1: Introdução ao Git, GitHub e Metodologias Ágeis

Objetivo

Capacitar os alunos a utilizarem ferramentas essenciais para o desenvolvimento de projetos em equipe, como **Git** e **GitHub**, além de introduzir os principais conceitos de **Metodologias Ágeis**.

Este módulo prepara os alunos para:

- ✓ Versionar e organizar o código-fonte.
 - ✓ Colaborar em equipe de forma estruturada.
 - ✓ Gerenciar o ciclo de desenvolvimento de projetos.
-

Parte 1: Git e GitHub

1. Introdução ao Versionamento de Código

♦ O que é Git?

- Git é um sistema de controle de versão distribuído, usado para rastrear mudanças no código-fonte durante o desenvolvimento de software.
- Ele permite que várias pessoas trabalhem no mesmo projeto sem sobrescrever o trabalho umas das outras.

♦ Por que usar Git?

- ✓ Histórico de alterações (quem, o quê e quando).
- ✓ Colaboração eficiente em equipe.
- ✓ Backup e recuperação de código.

♦ O que é GitHub?

- Plataforma de hospedagem de código que usa Git para controle de versão.
 - Facilita a colaboração e o gerenciamento de projetos.
-

2. Configuração Inicial do Git

♦ Instalação do Git

- **Windows:** Baixar o instalador em git-scm.com.
- **Linux:**

```
sudo apt install git
```

- **MacOS:**

```
brew install git
```

◆ Configurações Básicas

- Definir nome e e-mail:

```
git config --global user.name "Seu Nome"
git config --global user.email "seuemail@exemplo.com"
```

- Verificar configurações:

```
git config --list
```

3. Comandos Fundamentais do Git

◆ Iniciar um repositório

```
git init
```

◆ Adicionar arquivos ao versionamento

```
git add nome_do_arquivo
git add . # Adiciona todos os arquivos
```

◆ Salvar mudanças (commit)

```
git commit -m "Mensagem descritiva"
```

◆ Verificar o estado do repositório

```
git status
```

◆ Exibir o histórico de commits

```
git log
```

4. Branches e Controle de Versão Paralelo

◆ O que são Branches?

- Branches são ramificações do projeto que permitem trabalhar em funcionalidades ou correções sem afetar o código principal.

◆ Criar e alternar entre branches

```
git branch nova-feature
git checkout nova-feature
# ou
git switch nova-feature
```

◆ Mesclar branches

```
git checkout main
git merge nova-feature
```

◆ Resolução de conflitos

- Conflitos ocorrem quando há alterações em uma mesma parte do código.
 - Editar manualmente os arquivos conflitantes e fazer um novo commit.
-

5. Conectando-se ao GitHub

◆ Criar um repositório remoto

- Acesse github.com e crie um novo repositório.

♦ Conectar o repositório local ao remoto

```
git remote add origin https://github.com/seu-usuario/nome-do-repositorio.git  
git push -u origin main
```

♦ Clonar um repositório existente

```
git clone https://github.com/seu-usuario/nome-do-repositorio.git
```

6. Boas Práticas de Versionamento

- ✓ Commits frequentes e organizados.
 - ✓ Mensagens de commit claras e diretas.
 - ✓ Evitar commits muito grandes (fragmentar o trabalho).
 - ✓ Trabalhar em branches para funcionalidades específicas.
 - ✓ Revisar código antes de mesclar na branch principal.
-



Parte 2: Metodologias Ágeis

1. Introdução às Metodologias Ágeis

♦ O que é o Manifesto Ágil?

- Indivíduos e interações acima de processos e ferramentas.
- Software funcionando acima de documentação abrangente.
- Colaboração com o cliente acima de negociação de contrato.
- Responder a mudanças acima de seguir um plano.

♦ Por que usar metodologias ágeis?

- ✓ Flexibilidade para mudanças.
 - ✓ Entregas contínuas e iterativas.
 - ✓ Foco na colaboração e no feedback constante.
-

2. Principais Métodos Ágeis

♦ Scrum

- **Papéis:**
 - **Product Owner** – Define o valor e as prioridades.
 - **Scrum Master** – Remove impedimentos e facilita o processo.
 - **Time de Desenvolvimento** – Executa o trabalho.
- **Cerimônias:**
 - **Sprint Planning** – Planejamento da sprint.
 - **Daily Scrum** – Reunião diária de alinhamento.
 - **Sprint Review** – Revisão da sprint.
 - **Retrospectiva** – Avaliação e melhorias no processo.
- **Artefatos:**
 - **Backlog** → Lista de tarefas.
 - **Incremento** → Produto funcional entregue após a sprint.

♦ Kanban

- **Quadro de tarefas:**
 - Colunas: "To Do", "In Progress", "Done".
- **Fluxo contínuo:**
 - Movimentação de tarefas à medida que são concluídas.
- **Limite de trabalho em progresso (WIP):**
 - Controla a quantidade de tarefas em execução.

3. Aplicação de Metodologias Ágeis no Curso

- ✓ Criação de um quadro Kanban no **GitHub Projects**.
 - ✓ Organização das tarefas em "To Do", "In Progress" e "Done".
 - ✓ Revisão semanal das tarefas com feedback em equipe.
 - ✓ Acompanhamento contínuo e adaptação das atividades.
-

Módulo 2: Introdução ao Python

Objetivo

Apresentar a linguagem Python, suas características e como configurar o ambiente de desenvolvimento. Ao final deste módulo, você será capaz de:

- ✓ Escrever seus primeiros programas em Python.
 - ✓ Utilizar variáveis e diferentes tipos de dados.
 - ✓ Realizar operações básicas e manipular strings.
 - ✓ Criar um programa funcional utilizando entrada e saída de dados.
-

1. O que é Python?

História e Aplicações

Python foi criado por **Guido van Rossum** e lançado em **1991**.

É uma linguagem de **alto nível**, **interpretada** e de **propósito geral**, o que significa que pode ser usada para criar uma ampla variedade de aplicações, como:

- ✓ Desenvolvimento web (Django, Flask)
- ✓ Ciência de dados (Pandas, NumPy)
- ✓ Automação e scripts
- ✓ Inteligência artificial e machine learning (TensorFlow, PyTorch)
- ✓ Aplicações desktop e mobile

Vantagens do Python

- ✓ **Sintaxe simples e legível** – Fácil de aprender e entender.
 - ✓ **Grande comunidade** – Ampla documentação e suporte.
 - ✓ **Portabilidade** – Funciona em Windows, Linux e MacOS.
 - ✓ **Bibliotecas e frameworks** – Suporte para diferentes áreas como web, IA, automação, etc.
-

2. Instalação e Configuração do Ambiente

Instalação do Python

- **Windows** – Baixe o instalador em python.org e selecione a opção "Add to PATH" durante a instalação.
- **Linux** – Instale via terminal:

```
bash
CopyEdit
sudo apt install python3
```

- **MacOS** – Já vem pré-instalado, mas pode ser atualizado via Homebrew:

```
bash
CopyEdit
brew install python
```

✓ Verificação da Instalação

No terminal ou prompt de comando, digite:

```
bash
CopyEdit
python --version
```

- Saída esperada: Python 3.x.x

Configuração do Ambiente

- **VS Code** – Baixe em code.visualstudio.com e instale a extensão Python.
 - **PyCharm** – Para desenvolvimento profissional, com suporte a projetos mais complexos.
 - **Jupyter Notebook** – Ideal para ciência de dados e testes rápidos.
-

3. Primeiros Passos em Python

Modo Interativo

Abra o terminal e digite:

```
bash
CopyEdit
python
```

- Você verá o interpretador Python ativo, pronto para receber comandos.

Escrevendo o Primeiro Programa

1. Crie um arquivo primeiro_programa.py.
2. Digite o código:

```
python
CopyEdit
print("Olá, mundo!")
```

3. Salve o arquivo e execute no terminal:

```
bash
CopyEdit
python primeiro_programa.py
→ Saída esperada:
```

```
bash
CopyEdit
Olá, mundo!
```



Entrada e Saída de Dados

- **input()** – Para capturar dados do usuário.
- **print()** – Para exibir resultados na tela.



Exemplo:

```
python
CopyEdit
nome = input("Digite seu nome: ")
print(f"Olá, {nome}!")
```

4. Variáveis e Tipos de Dados

♦ O que são Variáveis?

Variáveis são espaços na memória para armazenar dados que podem ser manipulados durante a execução do programa.



Em Python, você não precisa definir o tipo explicitamente — o interpretador identifica automaticamente.

♦ Tipos de Dados Básicos

Tipo	Descrição	Exemplo
int	Números inteiros	10, -5
float	Números decimais	3.14, -0.5
str	Texto	"Python", '123'
bool	Valores lógicos	True, False



Exemplo de código:

```
python
CopyEdit
idade = 25 # int
altura = 1.75 # float
nome = "Maria" # str
estudante = True # bool
```

5. Operações Básicas

+ Operadores Aritméticos

Operador	Descrição	Exemplo	Resultado
+	Adição	10 + 5	15
-	Subtração	10 - 5	5
*	Multiplicação	10 * 5	50
/	Divisão	10 / 5	2.0
//	Divisão inteira	10 // 3	3
%	Resto da divisão	10 % 3	1
**	Exponenciação	2 ** 3	8

➡ Exemplo:

```
python
CopyEdit
a = 10
b = 3
print(a + b) # Soma → 13
print(a // b) # Divisão inteira → 3
```

Concatenação de Strings

➡ Exemplo:

```
python
CopyEdit
nome = "Maria"
saudacao = "Olá, " + nome + "!"
print(saudacao)
```

➡ Usando f-strings (forma mais moderna):

```
python
CopyEdit
nome = "Maria"
print(f"Olá, {nome}!")
```

Projeto Prático: Programa de Boas-Vindas

Descrição

- ➡ Crie um programa que solicite o nome e a idade do usuário e exiba uma mensagem personalizada.
- ➡ Verifique se o usuário é maior de idade.

Exemplo de Código

```
python
CopyEdit
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))
```

```
print(f"Olá, {nome}! Você tem {idade} anos.")
```

```
if idade >= 18:  
    print("Você é maior de idade!")  
else:  
    print("Você é menor de idade.")
```

→ **Desafio:**

- Crie uma versão que também exiba o ano em que o usuário nasceu.
- Exiba uma mensagem diferente dependendo se o usuário for menor ou maior de idade.

Dicas para o Sucesso

- ✓ Pratique os conceitos usando o modo interativo do Python.
- ✓ Experimente modificar os exemplos para entender como o comportamento muda.
- ✓ Use **comentários** para documentar seu código e facilitar a leitura.

Conclusão

Parabéns! 🎉 Você concluiu o módulo introdutório e agora está pronto para avançar para o próximo nível!

No próximo módulo, você aprenderá sobre **Lógica de Programação com Python** — incluindo condicionais, loops e manipulação de strings.

Prepare-se para escrever códigos mais complexos e começar a criar programas poderosos! 🚀🐍

Módulo 3: Lógica de Programação com Python

Objetivo

Desenvolver a lógica de programação utilizando Python como base.

Ao final deste módulo, você será capaz de:

- ✓ Criar programas que tomam decisões com estruturas condicionais.
- ✓ Repetir tarefas de forma automatizada com loops.
- ✓ Manipular e formatar textos com eficiência.
- ✓ Resolver problemas simples de lógica e fluxo de execução.

1. Operadores

Os operadores em Python são ferramentas para realizar cálculos e comparações. Vamos explorar os principais tipos:

+ Operadores Aritméticos

Usados para realizar cálculos matemáticos.

Operador	Descrição	Exemplo	Resultado
+	Adição	10 + 5	15
-	Subtração	10 - 5	5
*	Multiplicação	10 * 5	50
/	Divisão	10 / 3	3.3333
//	Divisão inteira	10 // 3	3
%	Resto da divisão	10 % 3	1
**	Exponenciação	2 ** 3	8

➡ Exemplo:

```
python
CopyEdit
soma = 10 + 5
divisao_inteira = 20 // 3
resto = 20 % 3
potencia = 2 ** 3

print(soma) # 15
print(divisao_inteira) # 6
print(resto) # 2
print(potencia) # 8
```

◆ Operadores Relacionais

Usados para comparar valores e retornar um valor booleano (True ou False).

Operador	Descrição	Exemplo	Resultado
==	Igual	10 == 10	True
!=	Diferente	10 != 5	True
>	Maior que	10 > 5	True
<	Menor que	5 < 10	True
>=	Maior ou igual	10 >= 10	True
<=	Menor ou igual	5 <= 10	True

➡ Exemplo:

```
python
CopyEdit
a = 10
```

```
b = 5
resultado = a > b
print(resultado) # True
```

◆ Operadores Lógicos

Usados para combinar expressões booleanas.

Operador	Descrição	Exemplo	Resultado
and	Retorna True se ambas as condições forem verdadeiras	(10 > 5) and (20 < 30)	True
or	Retorna True se pelo menos uma condição for verdadeira	(10 > 5) or (20 > 30)	True
not	Inverte o valor lógico da expressão	not (10 > 5)	False

➡ Exemplo:

```
python
CopyEdit
x = 10
y = 5

print(x > 5 and y < 10) # True
print(x > 5 or y > 10) # True
print(not(x > 5))      # False
```

2. Condicionais

As estruturas condicionais permitem que o programa tome decisões com base em certas condições.

◆ Estrutura if

Executa um bloco de código se uma condição for verdadeira.

➡ Exemplo:

```
python
CopyEdit
idade = 18

if idade >= 18:
    print("Você é maior de idade!")
```

◆ Estrutura if-else

Permite executar um código caso a condição seja verdadeira e outro caso seja falsa.

➡ Exemplo:

```
python
CopyEdit
idade = 16

if idade >= 18:
    print("Você é maior de idade!")
else:
    print("Você é menor de idade.")
```

◆ Estrutura if-elif-else

Permite testar múltiplas condições em sequência.

➡ Exemplo:

```
python
CopyEdit
nota = 85

if nota >= 90:
    print("A")
elif nota >= 80:
    print("B")
elif nota >= 70:
    print("C")
else:
    print("Reprovado")
```

3. Estruturas de Repetição

As estruturas de repetição permitem repetir um bloco de código várias vezes.

◆ Loop for

Itera sobre uma sequência de elementos (listas, strings, etc.).

➡ Exemplo:

```
python
CopyEdit
for i in range(5):
    print(f"Iteração {i}")
```

◆ Loop while

Repete um bloco de código enquanto a condição for verdadeira.

➡ Exemplo:

```
python
CopyEdit
contador = 0

while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
```

◆ Controle de Loops

- ✓ **break** – Interrompe o loop imediatamente.
- ✓ **continue** – Pula para a próxima iteração do loop.

➡ Exemplo:

```
python
CopyEdit
for i in range(10):
    if i == 5:
        break
    print(i)
```

4. Manipulação de Strings

◆ Concatenação

➡ Exemplo:

```
python
CopyEdit
nome = "Maria"
mensagem = "Olá, " + nome + "!"
print(mensagem)
```

◆ Formatação de Strings com f-strings

➡ Exemplo:

```
python
CopyEdit
nome = "Maria"
idade = 25
print(f"{nome} tem {idade} anos.")
```

◆ Métodos Úteis

Método	Descrição	Exemplo	Resultado
upper()	Converte para maiúsculas	"python".upper()	PYTHON
lower()	Converte para minúsculas	"Python".lower()	python

Método	Descrição	Exemplo	Resultado
strip()	Remove espaços extras	" Python ".strip()	Python
split()	Divide a string	"Python é incrível".split()	['Python', 'é', 'incrível']
replace()	Substitui valores	"Python".replace("P", "J")	Jython

✓ Conclusão

Você concluiu o módulo sobre lógica de programação! 🙌

Agora você sabe como tomar decisões, repetir tarefas e manipular strings com Python.

No próximo módulo, você aprenderá sobre **Estruturas de Dados** — listas, tuplas, dicionários e conjuntos.

Vamos continuar essa jornada rumo à maestria em Python! 🚀🐍

Módulo 4: Estruturas de Dados

Objetivo

Ensinar a trabalhar com coleções de dados em Python, como **listas**, **tuplas**, **dicionários** e **conjuntos**.

Ao final deste módulo, você será capaz de:

- ✓ Escolher a estrutura de dados mais adequada para cada situação.
 - ✓ Manipular e organizar dados com eficiência.
 - ✓ Executar operações como ordenação, busca e filtragem de dados.
-

1. Listas

As listas são coleções **ordenadas** e **mutáveis** de elementos.

- Podem armazenar **diferentes tipos de dados** (números, strings, booleanos, etc.).
- Os elementos são organizados em uma **ordem definida** e podem ser acessados por índice.

➡ **Exemplo:**

```
python
CopyEdit
frutas = ["maçã", "banana", "laranja"]
print(frutas) # ['maçã', 'banana', 'laranja']
```

◆ Acessando Elementos

- Os índices começam em **0** (zero).
- Índices negativos contam de trás para frente.

→ Exemplo:

```
python
CopyEdit
print(frutas[0]) # maçã
print(frutas[-1]) # laranja
```

◆ Métodos Úteis

Método	Descrição	Exemplo	Resultado
append()	Adiciona um elemento ao final da lista	frutas.append("uva")	['maçã', 'banana', 'laranja', 'uva']
insert()	Insere um elemento em uma posição específica	frutas.insert(1, "abacaxi")	['maçã', 'abacaxi', 'banana', 'laranja']
remove()	Remove a primeira ocorrência de um valor	frutas.remove("banana")	['maçã', 'laranja']
pop()	Remove e retorna o elemento em uma posição específica	frutas.pop(1)	'banana'
sort()	Ordena a lista em ordem crescente	frutas.sort()	['laranja', 'maçã', 'uva']
reverse()	Inverte a ordem dos elementos	frutas.reverse()	['uva', 'maçã', 'laranja']

→ Exemplo Completo:

```
python
CopyEdit
frutas.append("uva")
frutas.remove("banana")
frutas.sort()
print(frutas) # ['laranja', 'maçã', 'uva']
```

2. Tuplas

As tuplas são coleções **ordenadas** e **imutáveis** de elementos.

- Não podem ser modificadas após a criação.
- Usadas para armazenar dados que não devem ser alterados.

→ Exemplo:

```
python
```

```
CopyEdit
coordenadas = (10, 20)
print(coordenadas[0]) # 10
```

◆ Quando Usar Tuplas

- ✓ Dados que não devem ser alterados (exemplo: coordenadas geográficas).
- ✓ Melhor desempenho em comparação com listas.

➔ Exemplo:

```
python
CopyEdit
dias_da_semana = ("Segunda", "Terça", "Quarta", "Quinta", "Sexta")
print(dias_da_semana[2]) # Quarta
```

3. Dicionários

Os dicionários são coleções **não ordenadas** de pares **chave-valor**.

- As chaves devem ser **únicas e imutáveis** (exemplo: strings ou números).
- Muito úteis para armazenar dados estruturados.

➔ Exemplo:

```
python
CopyEdit
pessoa = {"nome": "Maria", "idade": 25, "cidade": "São Paulo"}
print(pessoa["nome"]) # Maria
```

◆ Métodos Úteis

Método	Descrição	Exemplo	Resultado
keys()	Retorna as chaves	pessoa.keys()	['nome', 'idade', 'cidade']
values()	Retorna os valores	pessoa.values()	['Maria', 25, 'São Paulo']
items()	Retorna pares chave-valor	pessoa.items()	[('nome', 'Maria'), ('idade', 25), ('cidade', 'São Paulo')]
get()	Retorna o valor de uma chave (ou valor padrão)	pessoa.get('idade', 30)	25
update()	Atualiza ou adiciona pares chave-valor	pessoa.update({'idade': 26})	{'nome': 'Maria', 'idade': 26, 'cidade': 'São Paulo'}

➔ Exemplo Completo:

```
python
CopyEdit
pessoa = {"nome": "Maria", "idade": 25}
pessoa["cidade"] = "São Paulo"
pessoa.update({"idade": 26})
print(pessoa)
```

4. Conjuntos

Os conjuntos são coleções **não ordenadas** de elementos **únicos**.

- Muito úteis para eliminar duplicatas e realizar operações matemáticas (união, interseção, etc.).

➔ Exemplo:

```
python
CopyEdit
numeros = {1, 2, 3, 4, 5}
print(numeros) # {1, 2, 3, 4, 5}
```

◆ Operações com Conjuntos

Operação	Símbolo	Exemplo	Resultado
----------	---------	---------	-----------

União	`	`	`{1, 2, 3}
-------	---	---	------------

Interseção	&	{1, 2, 3} & {3, 4, 5}	{3}
------------	---	-----------------------	-----

Diferença	-	{1, 2, 3} - {3, 4, 5}	{1, 2}
-----------	---	-----------------------	--------

➔ Exemplo Completo:

```
python
CopyEdit
conjunto1 = {1, 2, 3}
conjunto2 = {3, 4, 5}

print(conjunto1 | conjunto2) # {1, 2, 3, 4, 5}
print(conjunto1 & conjunto2) # {3}
print(conjunto1 - conjunto2) # {1, 2}
```

5. Escolhendo a Estrutura de Dados Correta

- ✓ **Listas** → Quando a **ordem** e a **mutabilidade** são importantes.
- ✓ **Tuplas** → Quando os dados são **imutáveis** e a **ordem** importa.
- ✓ **Dicionários** → Quando você precisa associar **chaves a valores**.
- ✓ **Conjuntos** → Quando você precisa garantir a **unicidade** dos elementos.

🎯 Projeto Prático: Sistema de Lista de Tarefas

Descrição

Crie um sistema que permita:

- ✓ Adicionar tarefas a uma lista.
- ✓ Remover tarefas.
- ✓ Exibir as tarefas em ordem alfabética.

➔ Exemplo de Código:

```
python
CopyEdit
tarefas = []
```



```
while True:
    tarefa = input("Digite uma tarefa (ou 'sair' para finalizar): ")
    if tarefa.lower() == "sair":
        break
    tarefas.append(tarefa)

tarefas.sort()
print("\nLista de tarefas:")
for t in tarefas:
    print(f"- {t}")
```

✓ Conclusão

Você concluiu o módulo sobre **Estruturas de Dados!** 🙌

Agora você sabe como organizar e manipular diferentes tipos de dados em Python.

No próximo módulo, você aprenderá a criar **funções** para tornar seu código mais eficiente e reutilizável. Vamos continuar essa jornada! 🚀🐍

Módulo 5: Funções em Python

Objetivo

Ensinar como criar funções reutilizáveis e entender o papel das funções na organização do código.

Ao final deste módulo, você será capaz de:

✓ Definir funções e reutilizá-las em diferentes partes do programa.

- ✓ Passar parâmetros e retornar valores.
 - ✓ Entender o escopo de variáveis e como ele afeta o comportamento das funções.
 - ✓ Aplicar boas práticas para tornar seu código mais limpo e organizado.
-

1. O que são Funções?

As funções são blocos de código que realizam uma tarefa específica e podem ser chamados em diferentes partes do programa.

- Elas ajudam a **evitar repetição de código** e facilitam a **manutenção**.
- Podem receber **valores de entrada (parâmetros)** e retornar **valores de saída**.

➡ Sintaxe Básica:

```
python
CopyEdit
def nome_da_funcao():
    # bloco de código
```

➡ Exemplo:

```
python
CopyEdit
def saudacao():
    print("Olá, mundo!")
```

saudacao() # Chamada da função

Saída:

```
css
CopyEdit
Olá, mundo!
```

2. Parâmetros e Retorno de Valores

Parâmetros

Você pode definir parâmetros para passar valores para a função.

- Os parâmetros tornam a função **mais flexível** e reutilizável.

➡ Exemplo:

```
python
CopyEdit
def saudacao(nome):
    print(f"Olá, {nome}!")
```

```
saudacao("Maria")
saudacao("João")
```

Saída:

```
css
CopyEdit
Olá, Maria!
```

Retorno de Valores

Use a palavra-chave `return` para retornar um valor da função.

- Após o `return`, o código da função é encerrado.

➡ Exemplo:

```
python
CopyEdit
def soma(a, b):
    return a + b
```

```
resultado = soma(5, 3)
print(resultado) # 8
```

Saída:

```
CopyEdit
8
```

➡ Exemplo com múltiplos retornos:

```
python
CopyEdit
def calcula_media(nota1, nota2):
    media = (nota1 + nota2) / 2
    if media >= 7:
        return media, "Aprovado"
    else:
        return media, "Reprovado"
```

```
resultado, status = calcula_media(8, 6)
print(f"Média: {resultado}, Status: {status}")
```

Saída:

```
makefile
CopyEdit
Média: 7.0, Status: Aprovado
```

3. Argumentos Posicionais e Nomeados

Argumentos Posicionais

Os argumentos são passados na **ordem em que foram definidos** na função.

➡ Exemplo:

```
python
CopyEdit
def subtracao(a, b):
    return a - b
```

```
print(subtracao(10, 5)) # 5
```

Argumentos Nomeados

Você pode especificar o nome do parâmetro ao passar o valor.

- Isso permite passar os argumentos em **qualquer ordem**.

➔ **Exemplo:**

```
python
CopyEdit
print(subtracao(b=5, a=10)) # 5
```

4. Valores Padrão para Parâmetros

Você pode definir valores padrão para os parâmetros.

- Se nenhum valor for passado, o valor padrão será usado.

➔ **Exemplo:**

```
python
CopyEdit
def saudacao(nome="mundo"):
    print(f"Olá, {nome}!")

saudacao() # Usa o valor padrão "mundo"
saudacao("Maria") # Usa o valor fornecido
```

Saída:

```
css
CopyEdit
Olá, mundo!
Olá, Maria!
```

5. Escopo de Variáveis

Variáveis Locais

São definidas dentro de uma função e só podem ser acessadas dentro dela.

➔ **Exemplo:**

```
python
CopyEdit
def minha_funcao():
    x = 10 # Variável local
    print(x)

minha_funcao()
print(x) # Erro: x não está definido
```

Variáveis Globais

São definidas fora de qualquer função e podem ser acessadas em qualquer parte do código.

➔ **Exemplo:**

```
python
CopyEdit
x = 10 # Variável global
```

```
def minha_funcao():
    print(x)
```

```
minha_funcao() # 10
print(x) # 10
```

➡ **Modificando uma variável global dentro de uma função:**

Use a palavra-chave global para modificar uma variável global.

```
python
CopyEdit
x = 10
```

```
def alterar_valor():
    global x
    x = 20
```

```
alterar_valor()
print(x) # 20
```

6. Funções Anônimas (Lambda)

As funções lambda são funções simples que podem ser definidas em uma única linha.

- Muito úteis para operações rápidas e simples.

➡ **Exemplo:**

```
python
CopyEdit
quadrado = lambda x: x ** 2
print(quadrado(5)) # 25
```

➡ **Exemplo com map() e lambda:**

```
python
CopyEdit
numeros = [1, 2, 3, 4, 5]
quadrados = list(map(lambda x: x ** 2, numeros))
print(quadrados) # [1, 4, 9, 16, 25]
```

7. Boas Práticas com Funções

✓ **Nomes Descritivos**

- Escolha nomes que descrevam claramente o que a função faz.
 - ✓ **Funções Pequenas e Específicas**
- Cada função deve realizar uma única tarefa.
 - ✓ **Documentação com docstrings**
- Use docstrings para documentar o propósito da função.

➡ **Exemplo:**

```
python
CopyEdit
def soma(a, b):
    """
    Retorna a soma de dois números.
    :param a: Primeiro número.
    :param b: Segundo número.
    :return: Soma de a e b.
    """
    return a + b
```

Projeto Prático: Sistema de Cálculo de Notas

Descrição

Crie um programa que:

- ✓ Solicite as notas de um aluno.
- ✓ Calcule a média usando uma função.
- ✓ Retorne se o aluno foi aprovado ou reprovado.

Exemplo de Código:

```
python
CopyEdit
def calcula_media(notas):
    return sum(notas) / len(notas)

def verificar_aprovacao(media):
    if media >= 7:
        return "Aprovado"
    else:
        return "Reprovado"

def exibir_resultado(notas):
    media = calcula_media(notas)
    status = verificar_aprovacao(media)
    print(f"Média: {media:.2f} - Situação: {status}")

notas = [8.5, 7.0, 9.0]
exibir_resultado(notas)
```

Saída:

```
makefile
CopyEdit
Média: 8.17 - Situação: Aprovado
```

Conclusão

Você concluiu o módulo sobre **Funções em Python!** 🙌

Agora você sabe como criar e reutilizar funções para simplificar e organizar seu código.

No próximo módulo, você aprenderá a **manipular arquivos** de texto, CSV e JSON em Python.

Vamos continuar essa jornada! 🚀🐍

Módulo 6: Manipulação de Arquivos

Objetivo

Ensinar como ler, escrever e manipular arquivos de texto, CSV e JSON em Python.

Ao final deste módulo, você será capaz de:

- ✓ Abrir, ler e escrever em arquivos de texto.
- ✓ Trabalhar com arquivos estruturados em formato CSV.
- ✓ Manipular dados em arquivos JSON.
- ✓ Lidar com erros e exceções ao trabalhar com arquivos.

1. Trabalhando com Arquivos de Texto

Abrindo Arquivos

Use a função `open()` para abrir um arquivo.

- **Sintaxe:**

```
python
CopyEdit
arquivo = open("nome_do_arquivo.txt", "modo")
```

→ **Modos de Abertura:**

Modo	Descrição
r	Leitura (modo padrão).
w	Escrita (cria um novo arquivo ou sobrescreve o existente).
a	Anexar (adiciona dados ao final do arquivo sem sobrescrever).
b	Modo binário (para arquivos não textuais, como imagens).
r+	Leitura e escrita.

→ **Exemplo:**

```
python
CopyEdit
arquivo = open("exemplo.txt", "r") # Abre para leitura
conteudo = arquivo.read()
print(conteudo)
arquivo.close() # Fecha o arquivo
```

Lendo Arquivos

→ **Funções para leitura:**

Função

Descrição

`read()` Lê todo o conteúdo do arquivo como uma string.

`readline()` Lê uma linha por vez.

`readlines()` Retorna uma lista com todas as linhas do arquivo.

➡ Exemplo:

```
python
CopyEdit
with open("exemplo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```

➡ Exemplo com `readlines()`:

```
python
CopyEdit
with open("exemplo.txt", "r") as arquivo:
    linhas = arquivo.readlines()
    for linha in linhas:
        print(linha.strip())
```

Escrevendo em Arquivos

➡ Exemplo:

```
python
CopyEdit
with open("exemplo.txt", "w") as arquivo:
    arquivo.write("Este é um exemplo de escrita em arquivo.\n")
    arquivo.write("Python é incrível!")
```

➡ Exemplo com a (anexar):

```
python
CopyEdit
with open("exemplo.txt", "a") as arquivo:
    arquivo.write("\nEste texto foi adicionado ao arquivo.")
```

Fechando Arquivos

- Use o método `close()` para fechar arquivos manualmente.
- A estrutura `with` fecha o arquivo automaticamente após o bloco de código.

➡ Exemplo:

```
python
CopyEdit
arquivo = open("exemplo.txt", "r")
print(arquivo.read())
arquivo.close()
```

➡ Exemplo com `with` (fechamento automático):

```
python
CopyEdit
with open("exemplo.txt", "r") as arquivo:
    print(arquivo.read())
```


2. Manipulação de Arquivos CSV

O que é CSV?

- CSV (Comma-Separated Values) é um formato para armazenar dados tabulares.
- Cada linha representa um registro e os valores são separados por vírgulas.

➡ Exemplo de arquivo CSV:

```
csv
CopyEdit
Nome,Idade,Profissão
Maria,25,Engenheira
João,30,Professor
```

Lendo Arquivos CSV

Use a biblioteca `csv` para ler arquivos CSV.

➡ Exemplo:

```
python
CopyEdit
import csv

with open("dados.csv", "r") as arquivo:
    leitor = csv.reader(arquivo)
    for linha in leitor:
        print(linha)
```

Saída:

```
css
CopyEdit
['Nome', 'Idade', 'Profissão']
['Maria', '25', 'Engenheira']
['João', '30', 'Professor']
```

Escrevendo em Arquivos CSV

Use `csv.writer()` para escrever em arquivos CSV.

➡ Exemplo:

```
python
CopyEdit
import csv

dados = [{"Nome": "Maria", "Idade": 25}, {"Nome": "João", "Idade": 30}]

with open("dados.csv", "w", newline="") as arquivo:
    escritor = csv.writer(arquivo)
    escritor.writerows(dados)
```

➡ Exemplo com DictWriter (usando dicionário):

```
python
CopyEdit
import csv

dados = [
    {"Nome": "Maria", "Idade": 25},
    {"Nome": "João", "Idade": 30}
]

with open("dados.csv", "w", newline="") as arquivo:
    campos = ["Nome", "Idade"]
    escritor = csv.DictWriter(arquivo, fieldnames=campos)
    escritor.writeheader()
    escritor.writerows(dados)
```

3. Manipulação de Arquivos JSON

O que é JSON?

- JSON (JavaScript Object Notation) é um formato leve para troca de dados.
- Estrutura semelhante a **dicionários em Python**.

➡ Exemplo de JSON:

```
json
CopyEdit
{
    "nome": "Maria",
    "idade": 25,
    "cidade": "São Paulo"
}
```

Lendo Arquivos JSON

Use a biblioteca json para ler arquivos JSON.

➡ Exemplo:

```
python
CopyEdit
import json

with open("dados.json", "r") as arquivo:
    dados = json.load(arquivo)
    print(dados)
```

Escrevendo em Arquivos JSON

Use json.dump() para salvar dados em formato JSON.

➡ Exemplo:

```
python
CopyEdit
import json

dados = {"nome": "Maria", "idade": 25}
```

```
with open("dados.json", "w") as arquivo:
    json.dump(dados, arquivo, indent=4)
```

4. Gerenciamento de Erros

Use try e except para lidar com erros ao manipular arquivos.

➡ Exemplo:

```
python
CopyEdit
try:
    with open("arquivo_inexistente.txt", "r") as arquivo:
        print(arquivo.read())
except FileNotFoundError:
    print("Arquivo não encontrado!")
```

➡ Exemplo com múltiplos erros:

```
python
CopyEdit
try:
    with open("dados.json", "r") as arquivo:
        dados = json.load(arquivo)
except FileNotFoundError:
    print("Arquivo não encontrado!")
except json.JSONDecodeError:
    print("Erro ao decodificar JSON!")
```

Projeto Prático: Sistema de Cadastro de Clientes

Descrição

Crie um sistema para gerenciar o cadastro de clientes em um arquivo JSON.

- ✓ Adicione clientes (nome, idade, cidade).
- ✓ Liste os clientes cadastrados.
- ✓ Salve os dados em um arquivo JSON.

➡ Exemplo de Código:

```
python
CopyEdit
import json

clientes = []

def adicionar_cliente(nome, idade, cidade):
    clientes.append({"nome": nome, "idade": idade, "cidade": cidade})
    with open("clientes.json", "w") as arquivo:
        json.dump(clientes, arquivo, indent=4)

adicionar_cliente("Maria", 25, "São Paulo")
adicionar_cliente("João", 30, "Rio de Janeiro")

with open("clientes.json", "r") as arquivo:
    print(json.load(arquivo))
```

✓ Conclusão

Agora você sabe como **manipular arquivos** de texto, CSV e JSON em Python!



No próximo módulo, você aprenderá a **criar módulos e pacotes** para organizar melhor o código e reutilizar funcionalidades.

Vamos continuar essa jornada! 🚀🐍

Módulo 7: Módulos e Pacotes

Objetivo

Ensinar como organizar o código em módulos e pacotes, além de explorar bibliotecas externas.

Ao final deste módulo, você será capaz de:

- ✓ Criar e importar módulos e pacotes.
- ✓ Organizar o código em arquivos reutilizáveis.
- ✓ Instalar e usar bibliotecas externas com pip.
- ✓ Trabalhar com módulos padrão do Python, como math, random e datetime.

1. O que são Módulos e Pacotes?

Módulos

- Um **módulo** é um arquivo .py que contém código Python.
- Pode incluir **funções, classes e variáveis**.
- Permite reutilização de código e divisão de responsabilidades.

➡ Exemplo de módulo (meu_modulo.py)

```
python
CopyEdit
# meu_modulo.py
```

```
def saudacao(nome):
    return f"Olá, {nome}!"
```

```
PI = 3.14159
```

➡ Vantagens de Usar Módulos:

- ✓ Organização e manutenção do código.
- ✓ Reutilização de código em diferentes projetos.
- ✓ Evita duplicação de código.

Pacotes

- Um **pacote** é uma coleção de módulos organizados em diretórios.
- Deve conter um arquivo `__init__.py` para ser reconhecido como um pacote.
- A organização em pacotes facilita a modularização de projetos maiores.

➡ Exemplo de estrutura de pacote:

```
markdown
CopyEdit
meu_pacote/
├── __init__.py
├── modulo1.py
└── modulo2.py
```

➡ Exemplo (modulo1.py):

```
python
CopyEdit
def funcao():
    return "Função do módulo 1"
```

➡ Exemplo (__init__.py)

```
python
CopyEdit
from .modulo1 import funcao
➡ Importação do pacote:
```

```
python
CopyEdit
from meu_pacote import funcao
print(funcao()) # "Função do módulo 1"
```

2. Criando e Importando Módulos

Criando um Módulo

1. Crie um arquivo `.py` com funções ou classes.
2. Implemente a lógica desejada.

➡ Exemplo (meu_modulo.py):

```
python
CopyEdit
def saudacao(nome):
    return f"Olá, {nome}!"
```

Importando um Módulo

1. Use a palavra-chave `import` para importar o módulo.
2. Acesse funções e variáveis usando a notação `modulo.funcao`.

➡ Exemplo:

```
python
CopyEdit
import meu_modulo
```

```
print(meu_modulo.saudacao("Maria")) # Saída: Olá, Maria!
```

Importando Funções Específicas

➡ Exemplo:

```
python
CopyEdit
from meu_modulo import saudacao

print(saudacao("João")) # Saída: Olá, João!
```

Importando com Alias (apelido)

➡ Exemplo:

```
python
CopyEdit
import meu_modulo as mm

print(mm.saudacao("Carlos")) # Saída: Olá, Carlos!
```

3. Criando e Usando Pacotes

Estrutura de um Pacote

1. Crie um diretório com um arquivo `__init__.py`.
2. Adicione módulos ao pacote.

➡ Exemplo de estrutura de pacote:

```
markdown
CopyEdit
meu_pacote/
├── __init__.py
├── modulo1.py
└── modulo2.py
```

➡ Exemplo (modulo1.py)

```
python
CopyEdit
def funcao1():
    return "Função 1 do módulo 1"
```

➡ Exemplo (modulo2.py)

```
python
CopyEdit
def funcao2():
    return "Função 2 do módulo 2"
```

➡ Exemplo (__init__.py)

```
python
CopyEdit
from .modulo1 import funcao1
from .modulo2 import funcao2
```

Importando de um Pacote

➡ Exemplo:

```
python
CopyEdit
from meu_pacote import funcao1, funcao2

print(funcao1()) # "Função 1 do módulo 1"
print(funcao2()) # "Função 2 do módulo 2"
```

4. Bibliotecas Padrão do Python

O Python oferece diversas bibliotecas padrão que facilitam tarefas comuns.

math (*operações matemáticas*)

➡ Exemplo:

```
python
CopyEdit
import math

print(math.sqrt(16)) # 4.0
print(math.pi) # 3.14159
```

random (*geração de números aleatórios*)

➡ Exemplo:

```
python
CopyEdit
import random

print(random.randint(1, 10)) # Número entre 1 e 10
print(random.choice(["Python", "Java", "C++"])) # Escolhe um valor aleatório
```

datetime (*manipulação de datas e horas*)

➡ Exemplo:

```
python
CopyEdit
from datetime import datetime

agora = datetime.now()
print(agora.strftime("%d/%m/%Y %H:%M"))
```

5. Instalando Bibliotecas Externas com pip

O que é pip?

- pip é o gerenciador de pacotes do Python.
- Usado para instalar e gerenciar bibliotecas externas.

Instalando uma Biblioteca

→ Exemplo:

```
bash
CopyEdit
pip install requests
```

Usando uma Biblioteca Instalada

→ Exemplo:

```
python
CopyEdit
import requests
```

```
resposta = requests.get("https://api.github.com")
print(resposta.status_code) # 200
```

Desinstalando uma Biblioteca

→ Exemplo:

```
bash
CopyEdit
pip uninstall requests
```

6. Boas Práticas com Módulos e Pacotes

- ✓ **Nomes Descritivos:** Use nomes claros e diretos para módulos e pacotes.
- ✓ **Organização:** Divida o código em módulos e pacotes com responsabilidades específicas.
- ✓ **Documentação:** Use docstrings para explicar o propósito de cada módulo e função.
- ✓ **Evite Nomes de Bibliotecas:** Não nomeie módulos com nomes de bibliotecas padrão (math, random).

→ Exemplo de Docstring:

```
python
CopyEdit
def saudacao(nome):
    """
    Retorna uma mensagem de saudação.

    :param nome: Nome da pessoa.
    :return: Mensagem de saudação.
    """
    return f"Olá, {nome}!"
```

Projeto Prático: Gerenciador de Senhas

Descrição

Crie um sistema para gerar senhas seguras usando a biblioteca random e organize o código em módulos e pacotes.

- ✓ Crie um módulo gerador.py para gerar senhas.

- ✓ Crie um módulo `validador.py` para validar a segurança das senhas.
- ✓ Crie um pacote `seguranca` para armazenar os módulos.

➡ Exemplo de Código:

```
python
CopyEdit
import random
import string

def gerar_senha(tamanho):
    caracteres = string.ascii_letters + string.digits + string.punctuation
    senha = "".join(random.choice(caracteres) for _ in range(tamanho))
    return senha

senha = gerar_senha(12)
print(f"Senha gerada: {senha}")
```

✓ Conclusão

Agora você sabe como **organizar seu código** em módulos e pacotes, e como **usar bibliotecas externas** para ampliar as funcionalidades do Python. 🙌
No próximo módulo, você mergulhará na **Programação Orientada a Objetos (POO)**! 🚀🐍

Módulo 8: Programação Orientada a Objetos (POO)

🎯 Objetivo

Introduzir os conceitos de **Programação Orientada a Objetos (POO)** e sua aplicação em Python. Ao final deste módulo, você será capaz de criar classes, instanciar objetos e aplicar os princípios fundamentais da POO, como **encapsulamento**, **herança** e **polimorfismo**.

📖 Conteúdo

1. O que é Programação Orientada a Objetos?

✓ Definição

- POO é um **paradigma de programação** que organiza o código em "**objetos**", que são instâncias de "**classes**".
- Foca na criação de objetos que contêm dados (atributos) e comportamentos (métodos).
- Os principais pilares da POO são:
 - **Abstração** – Simplificar a representação do mundo real.
 - **Encapsulamento** – Proteger os dados e expor apenas o necessário.

- **Herança** – Reutilizar e estender o comportamento de classes.
- **Polimorfismo** – Tratar objetos de diferentes classes de forma uniforme.

Vantagens

- ✓ Reutilização de código.
 - ✓ Maior organização e modularidade.
 - ✓ Facilidade de manutenção e escalabilidade.
-

2. Classes e Objetos

✓ Classes

- Uma **classe** é um modelo (ou blueprint) para criar objetos.
- Define os **atributos** (dados) e **métodos** (comportamentos).
- Sintaxe:

python

CopyEdit

```
class Carro:

    def __init__(self, marca, modelo):

        self.marca = marca

        self.modelo = modelo

    def exibir_info(self):

        return f"{self.marca} {self.modelo}"
```

✓ Objetos

- Um **objeto** é uma instância de uma classe.
- Os atributos e métodos definidos na classe ficam disponíveis no objeto.
- Exemplo:

python

CopyEdit

```
meu_carro = Carro("Toyota", "Corolla")

print(meu_carro.exibir_info()) # Saída: Toyota Corolla
```

3. Método `__init__` e Atributos

✓ Método `__init__`

- O método `__init__` é o **construtor** da classe.
- É chamado automaticamente quando um objeto é criado.
- Define os valores iniciais dos atributos do objeto.
- Exemplo:

python

CopyEdit

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade
```

✓ Atributos

- Os atributos pertencem a um objeto e armazenam valores associados a ele.
- São acessados com `self`.
- Exemplo:

python

CopyEdit

```
peessoa = Pessoa("Maria", 25)
print(peessoa.nome)  # Maria
print(peessoa.idade)  # 25
```

4. Encapsulamento

✓ Definição

- Encapsulamento é o conceito de **esconder os detalhes internos** de um objeto e expor apenas o necessário.
- Permite proteger os dados de acessos e modificações não autorizadas.

✓ Atributos Privados

- Para definir um atributo como privado, use `__` (dois underscores).
- Para acessá-lo, crie **métodos específicos** (getters e setters).

- Exemplo:

python

CopyEdit

```
class ContaBancaria:
    def __init__(self, saldo):
        self.__saldo = saldo

    def depositar(self, valor):
        self.__saldo += valor

    def exibir_saldo(self):
        return self.__saldo
```

✓ Uso dos Getters e Setters

python

CopyEdit

```
conta = ContaBancaria(1000)
conta.depositar(500)
print(conta.exibir_saldo()) # 1500
```

5. Herança

✓ Definição

- A **herança** permite criar uma nova classe baseada em outra classe.
- A classe que é herdada é chamada de **superclasse** e a que herda é chamada de **subclasse**.
- Exemplo:

python

CopyEdit

```
class Animal:
    def __init__(self, nome):
```

```
        self.nome = nome

    def fazer_som(self):
        return "Som genérico"

class Cachorro(Animal):
    def fazer_som(self):
        return "Au Au!"
```

✓ Uso de Herança

python

CopyEdit

```
cachorro = Cachorro("Rex")
print(cachorro.fazer_som()) # Au Au!
```

6. Polimorfismo

✓ Definição

- **Polimorfismo** permite que diferentes classes tenham métodos com o mesmo nome, mas comportamentos diferentes.
- Exemplo:

python

CopyEdit

```
class Gato(Animal):
    def fazer_som(self):
        return "Miau!"

animais = [Cachorro("Rex"), Gato("Mimi")]

for animal in animais:
```

```
print(animal.fazer_som())
```

✓ Saída:

nginx

CopyEdit

Au Au!

Miau!

7. Métodos Especiais

✓ Métodos Mágicos

- Métodos que começam e terminam com **duplo underscore** (`__`).
- Os mais comuns são:
 - `__init__` → Construtor.
 - `__str__` → Representação em string.
 - `__len__` → Retorna o tamanho do objeto.

✓ Exemplo

python

CopyEdit

```
class Livro:

    def __init__(self, titulo, autor):

        self.titulo = titulo

        self.autor = autor

    def __str__(self):

        return f"{self.titulo} por {self.autor}"
```

```
livro = Livro("1984", "George Orwell")

print(livro)  # 1984 por George Orwell
```



Dicas para o Sucesso

- ✓ Crie classes e objetos para modelar problemas reais.
- ✓ Use **encapsulamento** para proteger dados sensíveis.
- ✓ Pratique **herança** para reutilizar código e estender funcionalidades.
- ✓ Use **polimorfismo** para implementar comportamento genérico em objetos de diferentes classes.
- ✓ Explore métodos mágicos para personalizar o comportamento dos objetos.

Módulo 9: Tratamento de Erros



Objetivo

Ensinar como identificar, capturar e tratar erros (exceções) em Python. Ao final deste módulo, você será capaz de escrever código mais seguro e confiável, capaz de lidar com situações inesperadas de forma elegante e eficiente.



Conteúdo

1. O que são Exceções?

✓ Definição

- **Exceções** são erros que ocorrem durante a execução do programa, interrompendo o fluxo normal do código.
- Elas são geradas automaticamente quando ocorre um problema, mas também podem ser criadas manualmente.

✓ Exemplos Comuns de Exceções

- **ZeroDivisionError** → Tentativa de dividir por zero.
- **FileNotFoundError** → Arquivo não encontrado.
- **TypeError** → Operação realizada com tipos incompatíveis.
- **ValueError** → Valor inválido para uma operação.

✓ Exemplo de erro sem tratamento:

python

CopyEdit

```
numero = int(input("Digite um número: "))  
  
resultado = 10 / numero # Se o usuário digitar 0, ocorrerá um  
erro.  
  
print(f"Resultado: {resultado}")
```

✓ Saída (se o usuário digitar 0):

vbnet

CopyEdit

```
ZeroDivisionError: division by zero
```

2. Blocos **try**, **except**, **else** e **finally**

✓ Bloco **try**

- Contém o código que pode gerar uma exceção.
- Se ocorrer uma exceção dentro do bloco **try**, o Python interrompe a execução e pula para o bloco **except**.

✓ Exemplo:

python

CopyEdit

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: Divisão por zero!")
```

✓ Bloco **except**

- Captura e trata a exceção.
- Pode especificar o tipo de erro ou capturar **todas as exceções** usando **except Exception**.

✓ Exemplo:

python

CopyEdit

```
try:
    numero = int("abc") # Erro: não é um número
```



```
except ValueError:

    print("Erro: Valor inválido!")
```

✓ Bloco **else**

- Executado se **nenhuma exceção** ocorrer.
- Útil para código que só deve rodar se tudo der certo.

✓ Exemplo:

```
python
CopyEdit
try:

    resultado = 10 / 2
except ZeroDivisionError:

    print("Erro: Divisão por zero!")
else:

    print(f"Resultado: {resultado}")
```

✓ Bloco **finally**

- Executado **sempre**, independentemente de ocorrer ou não uma exceção.
- Usado para liberar recursos, como fechar arquivos ou conexões com bancos de dados.

✓ Exemplo:

```
python
CopyEdit
try:

    arquivo = open("dados.txt", "r")

    conteudo = arquivo.read()
except FileNotFoundError:

    print("Erro: Arquivo não encontrado!")
```

```
finally:  
    arquivo.close()
```

3. Capturando Múltiplas Exceções

- Você pode capturar diferentes tipos de erros separadamente, para tratá-los de forma específica.

✓ Exemplo:

python

CopyEdit

```
try:  
    numero = int(input("Digite um número: "))  
    resultado = 10 / numero  
except ValueError:  
    print("Erro: Valor inválido!")  
except ZeroDivisionError:  
    print("Erro: Divisão por zero!")
```

4. Levantando Exceções com **raise**

✓ Definição

- O comando **raise** é usado para **levantar exceções manualmente**.
- É útil para verificar condições específicas e forçar um erro quando necessário.

✓ Exemplo:

python

CopyEdit

```
def dividir(a, b):  
    if b == 0:  
        raise ValueError("Divisor não pode ser zero!")  
    return a / b
```

```
try:
    resultado = dividir(10, 0)
except ValueError as erro:
    print(f"Erro: {erro}")
```

5. Criando Exceções Personalizadas

✓ Definição

- Exceções personalizadas permitem criar mensagens de erro mais claras e específicas.
- Para isso, você deve herdar a classe `Exception`.

✓ Exemplo:

python

CopyEdit

```
class SaldoInsuficienteError(Exception):
    pass

def sacar(saldo, valor):
    if valor > saldo:
        raise SaldoInsuficienteError("Saldo insuficiente!")
    return saldo - valor

try:
    novo_saldo = sacar(100, 200)
except SaldoInsuficienteError as erro:
    print(f"Erro: {erro}")
```

6. Boas Práticas no Tratamento de Erros

- ✓ **Seja Específico:** Capture apenas as exceções que você pode tratar.
- ✓ **Evite Capturar Todas as Exceções:** Capturar `Exception` sem especificar pode ocultar erros inesperados.
- ✓ **Documente as Exceções:** Use docstrings para explicar quais exceções uma função pode gerar.

✓ **Exemplo de Boas Práticas:**

python

CopyEdit

```
def calcular_media(notas):  
    """  
    Calcula a média de uma lista de notas.  
    :param notas: Lista de notas.  
    :return: Média das notas.  
    :raises ValueError: Se a lista estiver vazia.  
    """  
  
    if not notas:  
        raise ValueError("A lista de notas está vazia.")  
  
    return sum(notas) / len(notas)  
  
try:  
    media = calcular_media([])  
except ValueError as erro:  
    print(f"Erro: {erro}")
```



Dicas para o Sucesso

- ✓ Pratique o tratamento de erros em situações comuns, como leitura de arquivos e entrada de dados.
 - ✓ Use **exceções personalizadas** para melhorar a clareza do código.
 - ✓ Teste seu código com entradas inválidas para garantir que ele lida bem com erros.
-

Módulo 10: Introdução às APIs

Objetivo

Ensinar o que são APIs, como elas funcionam e como consumi-las usando Python. Ao final deste módulo, você será capaz de fazer requisições HTTP, processar respostas em JSON e integrar APIs em seus projetos para obter ou enviar dados de forma eficiente.

Conteúdo

1. O que é uma API?

Definição

- **API (Application Programming Interface)** é um conjunto de regras que permite que diferentes sistemas se comuniquem entre si.
- As APIs definem como um sistema pode fazer solicitações (requisições) e como ele deve responder (respostas).
- Elas funcionam como uma "ponte" que conecta dois sistemas distintos.

Tipos de APIs

- **APIs Web** → Usam protocolos HTTP/HTTPS para comunicação entre sistemas.
 - **Exemplo:** APIs RESTful, GraphQL.
- **APIs Locais** → Oferecidas por sistemas operacionais ou bibliotecas internas.
 - **Exemplo:** API do sistema de arquivos (OS) ou API de manipulação de gráficos (OpenGL).

Exemplos de APIs Públicas

- **OpenWeatherMap** → Retorna dados meteorológicos.
 - **GitHub API** → Fornece informações sobre repositórios e usuários.
 - **JSONPlaceholder** → API de teste que retorna dados fictícios.
-

2. Como Funcionam as APIs Web?

Requisições HTTP

As APIs Web utilizam o protocolo HTTP (ou HTTPS) para permitir a troca de dados entre sistemas.

Principais Métodos HTTP:

Método	Descrição	Exemplo
GET	Solicita dados de um servidor	Buscar um usuário
POST	Envia dados para o servidor	Criar um novo usuário
PUT	Atualiza dados existentes	Atualizar os dados de um usuário
DELETE	Remove dados	Deletar um usuário

✓ Exemplo de URL de API:

arduino

CopyEdit

<https://api.exemplo.com/recursos?parametro=valor>

✓ Respostas HTTP

O servidor responde com um código de status que informa o resultado da requisição.

Códigos de Status Comuns:

Código	Significado	Descrição
200	OK	Requisição bem-sucedida
201	Created	Recurso criado com sucesso
400	Bad Request	Requisição inválida
401	Unauthorized	Acesso não autorizado
404	Not Found	Recurso não encontrado
500	Internal Server Error	Erro no servidor

✓ Formato Comum de Resposta:

- **JSON** → JavaScript Object Notation
- **Exemplo de Resposta JSON:**

json

CopyEdit

```
{  
    "nome": "Maria",  
    "idade": 25,  
    "cidade": "São Paulo"  
}
```

3. Consumindo APIs com Python

✓ Biblioteca **requests**

O Python possui a biblioteca **requests**, que facilita a realização de requisições HTTP.

Instalação:

bash

CopyEdit

```
pip install requests
```

✓ Fazendo uma Requisição **GET**

✓ Exemplo:

python

CopyEdit

```
import requests  
  
resposta =  
requests.get("https://jsonplaceholder.typicode.com/posts/1")  
  
if resposta.status_code == 200:
```

```
        dados = resposta.json()
        print(dados)
else:
    print(f"Erro: {resposta.status_code}")
```

✓ Saída:

json

CopyEdit

```
{
    "userId": 1,
    "id": 1,
    "title": "sunt aut facere",
    "body": "quia et suscipit"
}
```

✓ Passando Parâmetros na URL

Você pode enviar parâmetros usando o argumento `params`.

✓ Exemplo:

python

CopyEdit

```
parametros = {"userId": 1}

resposta =
requests.get("https://jsonplaceholder.typicode.com/posts",
params=parametros)

print(resposta.json())
```

✓ Enviando Dados com POST

Você pode enviar dados JSON usando o argumento `json`.

✓ Exemplo:

python

CopyEdit

```
dados = {"title": "Novo Post", "body": "Conteúdo do post",
"userId": 1}

resposta =
requests.post("https://jsonplaceholder.typicode.com/posts",
json=dados)

print(resposta.status_code) # 201 (Created)
print(resposta.json())
```

4. Trabalhando com Respostas JSON

✓ Convertendo JSON para Dicionário

Use o método `.json()` para converter a resposta para um dicionário.

✓ Exemplo:

python

CopyEdit

```
resposta =
requests.get("https://jsonplaceholder.typicode.com/posts/1")
dados = resposta.json()

print(dados["title"]) # sunt aut facere
```

✓ Exemplo de Estrutura JSON

json

CopyEdit

```
{  
    "nome": "Maria",  
    "idade": 25,  
    "cidade": "São Paulo"  
}
```

✓ Acessando os Valores:

python

CopyEdit

```
print(dados["nome"]) # Maria  
print(dados["idade"]) # 25  
print(dados["cidade"]) # São Paulo
```

5. Exemplo Prático: Consumindo uma API de Previsão do Tempo

✓ Passos:

1. Obtenha uma chave de API no [OpenWeatherMap](#).
2. Faça uma requisição para obter a previsão do tempo de uma cidade.
3. Exiba os dados relevantes, como temperatura e clima.

✓ Código:

python

CopyEdit

```
import requests  
  
chave_api = "SUA_CHAVE_AQUI"  
cidade = "São Paulo"  
  
url =  
f"http://api.openweathermap.org/data/2.5/weather?q={cidade}&appi  
d={chave_api}&units=metric"  
  
resposta = requests.get(url)
```

```
if resposta.status_code == 200:
    dados = resposta.json()
    temperatura = dados["main"]["temp"]
    clima = dados["weather"][0]["description"]
    print(f"Temperatura em {cidade}: {temperatura}°C")
    print(f"Condição: {clima}")
else:
    print(f"Erro: {resposta.status_code}")
```

6. Boas Práticas no Uso de APIs

✓ ✓ **Leia a Documentação:**

Cada API tem suas próprias regras e endpoints. Consulte a documentação oficial para detalhes.

✓ ✓ **Trate Erros:**

Verifique o código de status e trate os possíveis erros.

✓ ✓ **Use Caching:**

Armazene dados temporariamente para evitar requisições repetidas.

✓ ✓ **Respeite os Limites:**

Muitas APIs têm limites de requisições por dia ou hora.

✓ ✓ **Proteja as Chaves de API:**

Evite expor suas chaves de API em repositórios públicos.



Dicas para o Sucesso

✓ Pratique consumindo diferentes APIs públicas.

✓ Use bibliotecas como `requests` para simplificar o processo de requisição.

✓ Experimente APIs que retornam dados em formatos variados, como JSON e XML.

Módulo 11: Banco de Dados com PostgreSQL



Objetivo

Ensinar como conectar, manipular e gerenciar bancos de dados PostgreSQL usando Python. Ao final deste módulo, você será capaz de:

- ✓ Criar tabelas e definir relacionamentos.
 - ✓ Inserir, atualizar e excluir dados.
 - ✓ Realizar consultas SQL complexas.
 - ✓ Gerenciar transações para garantir a consistência dos dados.
-

Conteúdo

1. Introdução ao PostgreSQL

✓ O que é PostgreSQL?

- **PostgreSQL** é um sistema de gerenciamento de banco de dados relacional (SGBD) de código aberto.
 - Ele é conhecido por sua robustez, escalabilidade e conformidade com os padrões SQL.
 - Oferece suporte a operações transacionais complexas e é amplamente utilizado em aplicações empresariais e de alta escala.
-

✓ Por que usar PostgreSQL?

✓ **Suporte a transações ACID** → Atomicidade, Consistência, Isolamento e Durabilidade.

✓ **Extensível** → Criação de funções e tipos de dados personalizados.

✓ **Confiável** → Suporte a replicação, backups e controle de acesso robusto.

✓ **Código Aberto** → Comunidade ativa e melhorias constantes.

2. Instalação e Configuração

✓ Instalando o PostgreSQL

Windows:

- Baixe o instalador em [postgresql.org](https://www.postgresql.org).
- Siga o assistente de instalação.

Linux:

bash

CopyEdit

```
sudo apt update
```

```
sudo apt install postgresql
```

MacOS:

bash

CopyEdit

```
brew install postgresql
```

✓ Configurando o PostgreSQL

1. Inicie o servidor:

bash

CopyEdit

```
sudo service postgresql start
```

2. Acesse o terminal do PostgreSQL:

bash

CopyEdit

```
sudo -u postgres psql
```

3. Crie um banco de dados e um usuário:

sql

CopyEdit

```
CREATE DATABASE meu_banco;
```

```
CREATE USER meu_usuario WITH PASSWORD 'minha_senha';
```

```
GRANT ALL PRIVILEGES ON DATABASE meu_banco TO meu_usuario;
```

4. Saia do terminal:

bash

CopyEdit

```
\q
```

3. Conectando ao PostgreSQL com Python

✓ Instalando a Biblioteca **psycopg2**

psycopg2 é a biblioteca mais popular para conectar Python ao PostgreSQL.

bash

CopyEdit

```
pip install psycopg2
```

✓ Estabelecendo uma Conexão

✓ Exemplo:

python

CopyEdit

```
import psycopg2
```

```
# Conectar ao banco de dados
```

```
conn = psycopg2.connect(  
    dbname="meu_banco",  
    user="meu_usuario",  
    password="minha_senha",  
    host="localhost",  
    port="5432"  
)
```

```
# Criar um cursor
```

```
cursor = conn.cursor()
```

4. Executando Consultas SQL

✓ Criando Tabelas

Você pode criar tabelas usando comandos SQL diretamente via Python.

✓ Exemplo:

python

CopyEdit

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS clientes (
        id SERIAL PRIMARY KEY,
        nome VARCHAR(100) NOT NULL,
        email VARCHAR(100) UNIQUE NOT NULL
    )
""")
conn.commit()
```

✓ Inserindo Dados

✓ Exemplo:

python

CopyEdit

```
cursor.execute("""
    INSERT INTO clientes (nome, email)
    VALUES (%s, %s)
""", ("Maria", "maria@exemplo.com"))
conn.commit()
```

✓ Consultando Dados

Use `fetchall()` para buscar todos os registros ou `fetchone()` para buscar um registro.

✓ Exemplo:

python

CopyEdit

```
cursor.execute("SELECT * FROM clientes")
registros = cursor.fetchall()
```

```
for registro in registros:
    print(registro)
```

✓ Saída:

bash

CopyEdit

```
(1, 'Maria', 'maria@exemplo.com')
```

✓ Atualizando Dados

✓ Exemplo:

python

CopyEdit

```
cursor.execute("""
    UPDATE clientes
    SET email = %s
    WHERE id = %s
""", ("maria.nova@exemplo.com", 1))
conn.commit()
```

✓ Excluindo Dados

✓ Exemplo:

python

CopyEdit


```
cursor.execute("DELETE FROM clientes WHERE id = %s", (1,))
conn.commit()
```

5. Gerenciando Transações

✓ O que são Transações?

- Uma transação é um conjunto de operações que devem ser executadas como uma única unidade de trabalho.
 - Garantem que os dados permaneçam consistentes, mesmo em caso de falhas.
-

✓ Commit e Rollback

- **commit()** → Confirma as alterações no banco de dados.
- **rollback()** → Desfaz as alterações em caso de erro.

✓ Exemplo:

python

CopyEdit

try:

```
    cursor.execute("""
        INSERT INTO clientes (nome, email) VALUES (%s, %s)
    """, ("João", "joao@exemplo.com"))
    conn.commit()
```

except Exception as e:

```
    print(f"Erro: {e}")
    conn.rollback()
```

6. Boas Práticas com PostgreSQL

✓ Use Consultas Parametrizadas

Evita SQL injection e melhora a segurança.

✓ Exemplo:

python

CopyEdit

```
cursor.execute("SELECT * FROM clientes WHERE email = %s",  
("maria@exemplo.com",))
```

✓ Feche Conexões Após o Uso

Evita vazamento de memória e bloqueios.

✓ Exemplo:

python

CopyEdit

```
cursor.close()  
conn.close()
```

✓ Use Indexação para Melhorar o Desempenho

Crie índices em colunas frequentemente utilizadas em **WHERE** e **JOIN** para acelerar as consultas.

✓ Exemplo:

sql

CopyEdit

```
CREATE INDEX idx_email ON clientes (email);
```

✓ Evite Selecionar Todos os Dados

Selecione apenas os campos necessários para economizar memória e melhorar o desempenho.

✓ Exemplo:

python

CopyEdit

```
cursor.execute("SELECT nome, email FROM clientes WHERE id = %s",  
(1,))
```

Projeto Prático

Criar um Sistema de Gerenciamento de Clientes

✓ Crie um sistema que permita:

- Criar uma tabela `clientes` com colunas para `id`, `nome` e `email`.
- Adicionar novos clientes ao banco de dados.
- Listar todos os clientes cadastrados.
- Atualizar o e-mail de um cliente.
- Excluir um cliente com base no `id`.

✓ Exemplo de Código Completo:

python

CopyEdit

```
import psycopg2
```

```
conn = psycopg2.connect(  
    dbname="meu_banco",  
    user="meu_usuario",  
    password="minha_senha",  
    host="localhost"  
)
```

```
cursor = conn.cursor()
```

```
# Criar tabela
```

```
cursor.execute("""
```

```
    CREATE TABLE IF NOT EXISTS clientes (  
        id SERIAL PRIMARY KEY,  
        nome VARCHAR(100),  
        email VARCHAR(100) UNIQUE
```

```

        )
    """)
conn.commit()

# Inserir cliente
cursor.execute("""
    INSERT INTO clientes (nome, email)
    VALUES (%s, %s)
""", ("João", "joao@exemplo.com"))
conn.commit()

# Listar clientes
cursor.execute("SELECT * FROM clientes")
clientes = cursor.fetchall()
print(clientes)

# Fechar conexão
cursor.close()
conn.close()

```

Dicas para o Sucesso

- ✓ Pratique a criação e manipulação de tabelas no PostgreSQL.
- ✓ Use **JOIN**, **GROUP BY** e **ORDER BY** para criar consultas mais complexas.
- ✓ Certifique-se de fechar conexões após o uso.
- ✓ Trate erros com **try-except** para garantir a integridade dos dados.

Módulo 12: Testes Automatizados

Objetivo

Ensinar como escrever testes automatizados em Python para validar o funcionamento do código. Ao final deste módulo, você será capaz de:

- ✓ Criar testes unitários para validar funções e métodos.
 - ✓ Executar testes e interpretar resultados.
 - ✓ Capturar e tratar erros nos testes.
 - ✓ Implementar boas práticas para garantir a qualidade do código.
-

Conteúdo

1. O que são Testes Automatizados?

✓ Definição

- Testes automatizados são scripts que verificam se o código funciona conforme o esperado.
 - Eles ajudam a detectar **erros** (bugs) antes que o código seja implantado em produção.
 - São essenciais para garantir que novas alterações não quebrem funcionalidades existentes.
-

✓ Benefícios dos Testes Automatizados

- ✓ Reduzem o tempo de depuração.
 - ✓ Melhoram a qualidade e a confiabilidade do código.
 - ✓ Ajudam a identificar erros rapidamente.
 - ✓ Facilitam a refatoração e manutenção do código.
-

✓ Tipos de Testes

- **Testes Unitários** → Testam unidades individuais de código (ex: funções ou métodos).
 - **Testes de Integração** → Verificam a interação entre diferentes partes do sistema.
 - **Testes de Sistema** → Avaliam o comportamento do sistema como um todo.
 - **Testes de Regressão** → Garantem que novas mudanças não quebrem funcionalidades anteriores.
-

2. Introdução ao **unittest**

✓ O que é `unittest`?

- `unittest` é uma biblioteca padrão do Python para criação e execução de testes.
 - Inspirada em frameworks como **JUnit** (Java).
 - Oferece um conjunto robusto de ferramentas para testar funções, métodos e classes.
-

✓ Instalação (se necessário)

Se o `unittest` não estiver disponível, instale usando o pip:

bash

CopyEdit

```
pip install unittest2
```

✓ Estrutura de um Teste

- Testes são organizados em classes que herdam de `unittest.TestCase`.
- Os métodos de teste devem começar com `test_`.

✓ Exemplo:

python

CopyEdit

```
import unittest
```

```
class TestCalculadora(unittest.TestCase):
```

```
    def test_soma(self):
```

```
        self.assertEqual(2 + 2, 4)
```

```
if __name__ == '__main__':
```

```
    unittest.main()
```

✓ Saída:

markdown

CopyEdit

.

Ran 1 test in 0.001s

OK

3. Escrevendo Testes Unitários

✓ Métodos de Asserção

- **assertEqual(a, b)** → Verifica se **a == b**.
- **assertNotEqual(a, b)** → Verifica se **a != b**.
- **assertTrue(x)** → Verifica se **x** é **True**.
- **assertFalse(x)** → Verifica se **x** é **False**.
- **assertIsNone(x)** → Verifica se **x** é **None**.
- ***assertRaises(Exceção, função, args)** → Verifica se uma exceção foi levantada.

✓ Exemplo:

python

CopyEdit

```
def dividir(a, b):  
    if b == 0:  
        raise ValueError("Divisor não pode ser zero!")  
    return a / b
```

```
class TestOperacoes(unittest.TestCase):  
    def test_dividir(self):  
        self.assertEqual(dividir(10, 2), 5)  
        self.assertRaises(ValueError, dividir, 10, 0)
```

✓ Saída:

markdown

CopyEdit

.

Ran 1 test in 0.001s

OK

4. Executando Testes

✓ Executando Testes pelo Terminal

- Para executar um arquivo de teste específico:

bash

CopyEdit

```
python -m unittest nome_do_arquivo.py
```

- Para descobrir e executar todos os testes em um diretório:

bash

CopyEdit

```
python -m unittest discover
```

✓ Executando Testes no VS Code

- Instale a extensão **Python**.
- Ative o suporte a testes na configuração.
- Abra o arquivo e clique em "**Executar Teste**".

✓ Executando Testes no PyCharm

- Crie uma nova configuração de teste.
 - Selecione o arquivo ou diretório de teste.
 - Clique em "**Executar**".
-

5. Testes com **setUp** e **tearDown**

✓ **setUp**

- Método executado **antes** de cada teste.
 - Usado para inicializar configurações, criar objetos ou abrir conexões.
-

✓ **tearDown**

- Método executado **após** cada teste.
 - Usado para liberar recursos e fechar conexões.
-

✓ **Exemplo:**

python

CopyEdit

```
class TestBancoDeDados(unittest.TestCase):  
    def setUp(self):  
        self.conexao = criar_conexao()  
        self.cursor = self.conexao.cursor()  
  
    def tearDown(self):  
        self.cursor.close()  
        self.conexao.close()  
  
    def test_consulta(self):  
        self.cursor.execute("SELECT 1")  
        resultado = self.cursor.fetchone()[0]  
        self.assertEqual(resultado, 1)
```

6. Testando Exceções

✓ Exemplo:

python

CopyEdit

```
def dividir(a, b):  
    if b == 0:  
        raise ValueError("Divisão por zero!")  
  
class TestOperacoes(unittest.TestCase):  
    def test_divisao_por_zero(self):  
        with self.assertRaises(ValueError):  
            dividir(10, 0)
```

7. Boas Práticas em Testes Automatizados

✓ Teste Casos de Borda

- Verifique cenários extremos ou inesperados (ex: divisão por zero).
- Exemplo:

python

CopyEdit

```
self.assertEqual(dividir(0, 1), 0)
```

✓ Mantenha os Testes Pequenos e Simples

- Cada teste deve cobrir um único comportamento ou funcionalidade.

✓ Use Nomes Descritivos

- Nomes de métodos devem descrever claramente o que está sendo testado.

- Exemplo: `test_soma_valores_negativos`.
-

✓ Não Ignore Erros Silenciosos

- Verifique se exceções são tratadas corretamente.
-

✓ Automatize a Execução dos Testes

- Use ferramentas de integração contínua (CI/CD) como GitHub Actions ou Travis CI para rodar testes automaticamente em cada `push`.
-

Projeto Prático

Criar um Sistema de Testes para uma Calculadora

✓ Crie uma classe `Calculadora` com métodos para:

- **soma** → Retorna a soma de dois números.
- **subtracao** → Retorna a subtração de dois números.
- **multiplicacao** → Retorna a multiplicação de dois números.
- **divisao** → Retorna a divisão de dois números (tratar divisão por zero).

✓ Crie uma classe de teste para validar os métodos usando `unittest`:

- Teste a saída correta dos métodos.
 - Teste o comportamento em casos extremos (ex: divisão por zero).
-

✓ Exemplo de Código Completo:

python

CopyEdit

```
class Calculadora:
    def soma(self, a, b):
        return a + b

    def subtracao(self, a, b):
        return a - b
```

```
def multiplicacao(self, a, b):  
    return a * b  
  
def divisao(self, a, b):  
    if b == 0:  
        raise ValueError("Divisão por zero!")  
    return a / b
```

✓ Testes:

python

CopyEdit

```
class TestCalculadora(unittest.TestCase):  
    def test_soma(self):  
        self.assertEqual(Calculadora().soma(2, 3), 5)
```

Dicas para o Sucesso

- ✓ Escreva testes para cada função e método criado.
- ✓ Cubra todos os cenários possíveis, incluindo casos extremos.
- ✓ Execute os testes regularmente para garantir a estabilidade do código.

Módulo 13: Desenvolvimento de APIs com Flask

Objetivo

Ensinar como criar APIs RESTful usando o framework Flask. Ao final deste módulo, você será capaz de:

- ✓ Configurar e executar um projeto Flask.
 - ✓ Criar rotas e lidar com métodos HTTP (GET, POST, PUT, DELETE).
 - ✓ Trabalhar com JSON para envio e recebimento de dados.
 - ✓ Implementar operações CRUD em uma API.
 - ✓ Tratar erros e seguir boas práticas de segurança.
-



Conteúdo

1. O que é Flask?

✓ Definição

- Flask é um **microframework** para desenvolvimento web em Python.
 - É leve, flexível e fácil de usar.
 - Ideal para criar APIs RESTful e aplicações web de pequeno e médio porte.
-

✓ Por que usar Flask?

- ✓ Código simples e minimalista.
 - ✓ Fácil integração com bibliotecas populares.
 - ✓ Extensível com plugins e ferramentas externas.
 - ✓ Suporte a rotas dinâmicas e retorno em JSON.
-

✓ Flask vs Django

Flask	Django
Simples e leve	Mais robusto e complexo
Flexível para diferentes arquiteturas	Arquitetura baseada em convenções
Melhor para APIs e projetos pequenos	Melhor para aplicações grandes e complexas

2. Instalação e Configuração

✓ Instalando o Flask

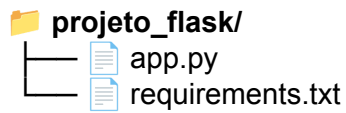
Use o `pip` para instalar o Flask:

bash

CopyEdit

```
pip install Flask
```

✓ Estrutura Básica de um Projeto Flask



✓ Exemplo de Código:

python

CopyEdit

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def home():
    return "Olá, mundo!"

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

✓ Executando o Projeto:

bash

CopyEdit

```
python app.py
```

✓ Saída:

csharp

CopyEdit

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

- Acesse o navegador e digite <http://127.0.0.1:5000/> para ver o resultado.

3. Rotas e Métodos HTTP

✓ Definindo Rotas

- Use o decorador `@app.route()` para definir rotas.
- Flask aceita diferentes métodos HTTP (GET, POST, PUT, DELETE).

✓ Exemplo:

python

CopyEdit

```
@app.route('/saudacao/<nome>', methods=['GET'])
def saudacao(nome):
    return f"Olá, {nome}!"
```

✓ Aceitando Vários Métodos

✓ Exemplo:

python

CopyEdit

```
@app.route('/dados', methods=['POST', 'PUT'])
def receber_dados():
    return "Dados recebidos!"
```

- **GET** → Para leitura de dados.
 - **POST** → Para criar novos recursos.
 - **PUT** → Para atualizar dados existentes.
 - **DELETE** → Para excluir recursos.
-

4. Trabalhando com JSON

✓ Recebendo JSON

Use `request.json` para capturar dados enviados em formato JSON.

✓ Exemplo:

python

CopyEdit

```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/api', methods=['POST'])
def receber_dados():
    dados = request.json
    return jsonify({"mensagem": "Dados recebidos!", "dados":
dados})
```

✓ Enviando JSON

Use `jsonify()` para formatar a resposta como JSON.

✓ Exemplo:

python

CopyEdit

```
@app.route('/status', methods=['GET'])
def status():
    return jsonify({"status": "API funcionando!", "versao":
"1.0"})
```

✓ Saída:

json

CopyEdit

```
{
  "status": "API funcionando!",
  "versao": "1.0"
}
```

5. Criando uma API CRUD

✓ Definição de CRUD

CRUD → **Create** (Criar), **Read** (Ler), **Update** (Atualizar) e **Delete** (Excluir).

- Essas operações são fundamentais para manipular dados em APIs RESTful.

✓ Exemplo de API CRUD para Gerenciar Tarefas:

python

CopyEdit

```
tarefas = []

@app.route('/tarefas', methods=['GET'])
def listar_tarefas():
    return jsonify(tarefas)

@app.route('/tarefas', methods=['POST'])
def criar_tarefa():
    nova_tarefa = request.json
    tarefas.append(nova_tarefa)
    return jsonify(nova_tarefa), 201

@app.route('/tarefas/<int:id>', methods=['PUT'])
def atualizar_tarefa(id):
    if id < len(tarefas):
        tarefas[id] = request.json
        return jsonify(tarefas[id])
    return jsonify({"erro": "Tarefa não encontrada"}), 404

@app.route('/tarefas/<int:id>', methods=['DELETE'])
```

```
def excluir_tarefa(id):  
    if id < len(tarefas):  
        tarefa_removida = tarefas.pop(id)  
        return jsonify(tarefa_removida)  
    return jsonify({"erro": "Tarefa não encontrada"}), 404
```

✓ Testando com cURL:

bash

CopyEdit

```
curl -X POST -H "Content-Type: application/json" -d '{"nome":  
"Estudar Flask"}' http://127.0.0.1:5000/tarefas
```

6. Gerenciamento de Erros

✓ Tratamento de Exceções com `@app.errorhandler`

- 404 → Recurso não encontrado
- 500 → Erro interno no servidor

✓ Exemplo:

python

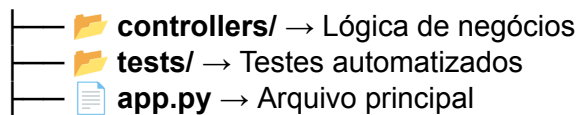
CopyEdit

```
@app.errorhandler(404)  
def recurso_nao_encontrado(erro):  
    return jsonify({"erro": "Recurso não encontrado"}), 404
```

7. Boas Práticas com Flask

✓ Organização do Projeto

- Crie uma estrutura clara e organizada:
 📁 projeto_flask/
 ├── 📁 routes/ → Arquivos de rotas
 ├── 📁 models/ → Arquivos de modelos



✓ Uso de Blueprints

- Permite organizar rotas em módulos.
- Exemplo:

python

CopyEdit

```
from flask import Blueprint

rotas = Blueprint('rotas', __name__)

@rotas.route('/home')
def home():
    return "Página inicial"
```

✓ Validação de Dados

- Use a biblioteca `marshmallow` para validar entradas.

bash

CopyEdit

```
pip install marshmallow
```

✓ Exemplo:

python

CopyEdit

```
from marshmallow import Schema, fields

class TarefaSchema(Schema):
    nome = fields.String(required=True)
```

✓ Segurança

- ✓ Sempre use HTTPS.
 - ✓ Valide os dados recebidos para evitar ataques de injeção.
 - ✓ Use tokens de autenticação (JWT).
-

Projeto Prático

Criar uma API de Gerenciamento de Usuários

✓ Crie uma API com operações CRUD para gerenciar usuários:

- **POST** → Criar um novo usuário
- **GET** → Retornar todos os usuários
- **PUT** → Atualizar um usuário
- **DELETE** → Excluir um usuário

✓ Adicione:

- ✓ Validação de dados com `marshmallow`
 - ✓ Tratamento de erros
 - ✓ Documentação com Swagger
-

Dicas para o Sucesso

- ✓ Teste sua API com o Postman ou cURL.
- ✓ Pratique a criação de diferentes rotas.
- ✓ Explore o uso de Blueprints para organizar melhor o projeto.

Módulo 14: Introdução ao Django

Objetivo

Ensinar os conceitos básicos do Django e como usá-lo para criar aplicações web robustas e escaláveis. Ao final deste módulo, você será capaz de:

- ✓ Configurar um projeto Django.
 - ✓ Criar modelos, views e templates.
 - ✓ Criar um painel de administração para gerenciar dados.
 - ✓ Compreender o padrão MVT (Model-View-Template).
-



Conteúdo

1. O que é Django?

✓ Definição

- Django é um **framework web Python de alto nível** que permite o desenvolvimento rápido de aplicações web.
 - Segue o padrão **MVC (Model-View-Controller)**, adaptado para **MVT (Model-View-Template)** no Django.
-

✓ Por que usar Django?

- ✓ Pronto para uso (banco de dados, autenticação, painel de administração).
 - ✓ Código seguro (proteção contra ataques comuns como CSRF e SQL Injection).
 - ✓ Escalável (usado por grandes plataformas como Instagram e Pinterest).
 - ✓ Suporta desenvolvimento modular (facilidade para adicionar novas funcionalidades).
-

✓ Django vs Flask

Django	Flask
Framework completo com muitas funcionalidades integradas	Microframework simples e flexível
Arquitetura baseada em convenção	Mais liberdade de escolha para configuração
Melhor para projetos complexos	Melhor para APIs e projetos pequenos

2. Instalação e Configuração

✓ Instalando o Django

Use o **pip** para instalar o Django:

bash

CopyEdit

```
pip install django
```

✓ Criando um Novo Projeto Django

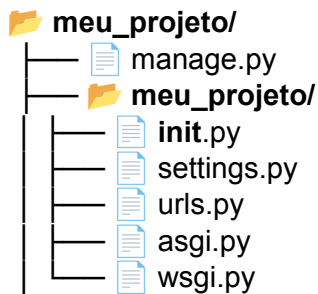
Use o comando `django-admin` para criar um novo projeto:

bash

CopyEdit

```
django-admin startproject meu_projeto
```

✓ Estrutura do Projeto



✓ Executando o Servidor de Desenvolvimento

No diretório do projeto, execute:

bash

CopyEdit

```
python manage.py runserver
```

✓ Saída:

nginx

CopyEdit

```
Starting development server at http://127.0.0.1:8000/
```

- Acesse o navegador e digite `http://127.0.0.1:8000/`.
 - Você verá a tela inicial do Django.
-

3. Criando uma Aplicação Django

✓ O que é uma Aplicação?

- Uma aplicação é um módulo que realiza uma função específica dentro do projeto.
 - Projetos Django podem conter várias aplicações.
-

✓ Criando uma Aplicação

bash

CopyEdit

```
python manage.py startapp minha_app
```

✓ Estrutura da Aplicação

```
minha_app/  
├── migrations/  
├── init.py  
├── admin.py  
├── apps.py  
├── models.py  
├── tests.py  
└── views.py
```

✓ Registrando a Aplicação

Adicione o nome da aplicação em `settings.py`:

python

CopyEdit

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',
```

```
'minha_app',  
]
```

4. Modelos (Models)

✓ O que são Modelos?

- Modelos são classes que representam tabelas no banco de dados.
 - Django usa um ORM (Object-Relational Mapping) para mapear modelos para tabelas SQL.
-

✓ Criando um Modelo

 **models.py**

python

CopyEdit

```
from django.db import models
```

```
class Produto(models.Model):
```

```
    nome = models.CharField(max_length=100)
```

```
    preco = models.DecimalField(max_digits=10, decimal_places=2)
```

```
    descricao = models.TextField()
```

```
    def __str__(self):
```

```
        return self.nome
```

✓ Criando e Aplicando Migrações

Crie as migrações:

bash

CopyEdit

```
python manage.py makemigrations
```


Aplique as migrações:

bash

CopyEdit

```
python manage.py migrate
```

5. Views e URLs

✓ O que são Views?

- Views definem como o Django processa uma requisição e retorna uma resposta.
 - Podem ser baseadas em funções ou classes.
-

✓ Criando uma View

 **views.py**

python

CopyEdit

```
from django.http import HttpResponse
```

```
def home(request):
```

```
    return HttpResponse("Bem-vindo ao meu site!")
```

✓ Mapeando URLs

 **urls.py**

python

CopyEdit

```
from django.urls import path
```

```
from .views import home
```

```
urlpatterns = [
    path('', home, name='home'),
]
```

6. Templates

✓ O que são Templates?

- Templates são arquivos HTML que permitem incluir dados dinamicamente.
-

✓ Configurando Templates

 **settings.py**

python

CopyEdit

```
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        ...
    },
]
```

✓ Criando um Template

 **templates/home.html**

html

CopyEdit

```
<!DOCTYPE html>
<html>
```

```
<head>
    <title>Minha Aplicação</title>
</head>
<body>
    <h1>Bem-vindo, {{ nome }}!</h1>
</body>
</html>
```

✓ Renderizando um Template

 **views.py**

python

CopyEdit

```
from django.shortcuts import render
```

```
def home(request):
    return render(request, 'home.html', {'nome': 'João'})
```

7. Painel de Administração

✓ Configurando o Painel de Administração

Registre o modelo no painel de administração:

 **admin.py**

python

CopyEdit

```
from django.contrib import admin
```

```
from .models import Produto
```

```
admin.site.register(Produto)
```

✓ Criando um Superusuário

bash

CopyEdit

```
python manage.py createsuperuser
```

- Acesse o painel de administração em <http://127.0.0.1:8000/admin/>
- Insira o nome de usuário e senha criados.

8. Testes Automatizados

✓ Criando um Teste

 tests.py

python

CopyEdit

```
from django.test import TestCase

class TesteSimples(TestCase):
    def test_soma(self):
        self.assertEqual(1 + 1, 2)
```

✓ Executando os Testes:

bash

CopyEdit

```
python manage.py test
```



Projeto Prático



Criar um Sistema de Cadastro de Produtos

- ✓ Crie um sistema com as funcionalidades:
 - ✓ Criar um produto.
 - ✓ Listar todos os produtos.
 - ✓ Atualizar um produto existente.
 - ✓ Excluir um produto.
 - ✓ Gerenciar os dados pelo painel de administração.

- ✓ **Dicas:**
 - ✓ Use o ORM para manipulação de dados.
 - ✓ Crie um template para listagem de produtos.
 - ✓ Crie testes automatizados para validar o sistema.
-

Dicas para o Sucesso

- ✓ Pratique a criação de modelos e views.
- ✓ Explore o painel de administração para gerenciar dados.
- ✓ Consulte a documentação oficial do Django para detalhes avançados.

Módulo 15: Projeto Final

Objetivo

Consolidar todo o aprendizado do curso desenvolvendo uma aplicação completa.

Ao final deste módulo, você será capaz de:

- ✓ Estruturar um projeto completo com Python.
 - ✓ Implementar operações CRUD usando Flask ou Django.
 - ✓ Integrar o sistema com um banco de dados PostgreSQL.
 - ✓ Testar e fazer o deploy da aplicação.
-

Conteúdo

1. Escolha do Projeto

Aqui estão algumas ideias de projetos que você pode desenvolver. Escolha um de acordo com o seu nível de confiança e interesse:

1.1. Sistema de Gerenciamento de Tarefas (To-Do List)

Funcionalidades:

- ✓ Criar, editar e excluir tarefas.
- ✓ Marcar tarefas como concluídas.
- ✓ Filtrar tarefas por status (pendentes, concluídas).

Tecnologias:

- ✓ Flask ou Django para o backend.

- ✓ PostgreSQL para o banco de dados.
 - ✓ Frontend com HTML/CSS ou um framework como React (opcional).
-

1.2. Blog Pessoal

Funcionalidades:

- ✓ Criar, editar e excluir posts.
- ✓ Sistema de comentários.
- ✓ Autenticação de usuários (login e logout).

Tecnologias:

- ✓ Django (com painel de administração).
 - ✓ PostgreSQL ou SQLite para o banco de dados.
 - ✓ Frontend com templates Django ou um framework moderno (React ou Vue).
-

1.3. API de Previsão do Tempo

Funcionalidades:

- ✓ Consumir uma API externa de previsão do tempo (ex: OpenWeatherMap).
- ✓ Retornar dados em formato JSON.
- ✓ Permitir busca por cidade.

Tecnologias:

- ✓ Flask para criar a API.
 - ✓ Biblioteca `requests` para consumir a API externa.
-

1.4. Sistema de Gerenciamento de Estoque

Funcionalidades:

- ✓ Cadastrar, editar e excluir produtos.
- ✓ Registrar entradas e saídas de estoque.
- ✓ Gerar relatórios de estoque.

Tecnologias:

- ✓ Django (com painel de administração).
 - ✓ PostgreSQL para o banco de dados.
 - ✓ Frontend com templates Django.
-

2. Estrutura do Projeto

Independentemente do projeto escolhido, siga esta estrutura básica para garantir um código bem organizado e funcional:

✓ 2.1. Planejamento

- ✓ Defina as funcionalidades principais.
 - ✓ Crie um diagrama do banco de dados (se aplicável).
 - ✓ Planeje as rotas da API ou as páginas da aplicação.
 - ✓ Crie um roadmap de desenvolvimento.
-

✓ 2.2. Configuração do Ambiente

- ✓ Crie um ambiente virtual:

bash

CopyEdit

```
python -m venv venv
```

```
source venv/bin/activate # Linux/Mac
```

```
venv\Scripts\activate # Windows
```

- ✓ Instale as dependências necessárias (exemplo para Flask):

bash

CopyEdit

```
pip install flask flask_sqlalchemy requests
```

- ✓ Instale as dependências para Django:

bash

CopyEdit

```
pip install django psycopg2
```

✓ 2.3. Desenvolvimento

Siga uma abordagem modular:

- ✓ Crie modelos para representar os dados no banco de dados.
- ✓ Desenvolva as views ou endpoints da API.
- ✓ Implemente a lógica de negócio de forma organizada.

👉 Use Git para versionar o código:

bash

CopyEdit

```
git init
```

```
git add .
```

```
git commit -m "Primeiro commit"
```

👉 Crie um repositório no GitHub:

bash

CopyEdit

```
git remote add origin
```

```
https://github.com/seu-usuario/seu-projeto.git
```

```
git push -u origin main
```

✅ 2.4. Testes

✓ Escreva testes unitários para as funcionalidades principais.

✓ Use `unittest` ou `pytest` para executar os testes.

Exemplo de teste com Flask:

📄 **test_app.py**

python

CopyEdit

```
import unittest
```

```
from app import app, db, Tarefa
```

```
class TestTarefas(unittest.TestCase):
```

```
    def setUp(self):
```

```
        app.config['TESTING'] = True
```

```
        app.config['SQLALCHEMY_DATABASE_URI'] =  
'sqlite:///memory:'
```

```
        self.app = app.test_client()
```

```
        db.create_all()
```



```
def tearDown(self):  
    db.session.remove()  
    db.drop_all()  
  
def test_criar_tarefa(self):  
    resposta = self.app.post('/tarefas', json={"descricao":  
"Teste"})  
    self.assertEqual(resposta.status_code, 201)
```

✓ Rodando os testes:

bash

CopyEdit

```
python -m unittest discover
```

✓ 2.5. Documentação

✓ Crie um arquivo **README.md** com:

- Descrição do projeto.
- Tecnologias usadas.
- Como instalar e executar.
- Exemplos de requisições (se for uma API).

 **README.md**

markdown

CopyEdit

```
# Sistema de Gerenciamento de Tarefas
```

```
## Tecnologias
```

```
- Flask
```

```
- PostgreSQL
```

Como executar

1. Clone o repositório:

git clone <https://github.com/seu-usuario/seu-projeto.git>

csharp

CopyEdit

2. Instale as dependências:

pip install -r requirements.txt

markdown

CopyEdit

3. Execute o servidor:

python app.py

CopyEdit

✓ 2.6. Deploy (Opcional)

✓ Hospede o projeto em plataformas como:

- Heroku
- Vercel
- AWS

📌 Exemplo de Deploy no Heroku:

1. Instale o Heroku CLI:

bash

CopyEdit

curl <https://cli-assets.heroku.com/install.sh> | sh

2. Faça login no Heroku:

bash

CopyEdit

```
heroku login
```

3. Crie o app:

```
bash
```

```
CopyEdit
```

```
heroku create nome-do-app
```

4. Faça o deploy:

```
bash
```

```
CopyEdit
```

```
git push heroku main
```

3. Exemplo de Código Completo (Sistema de Tarefas com Flask)

 **app.py**

```
python
```

```
CopyEdit
```

```
from flask import Flask, request, jsonify
```

```
from flask_sqlalchemy import SQLAlchemy
```

```
app = Flask(__name__)
```

```
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///tarefas.db'
```

```
db = SQLAlchemy(app)
```

```
class Tarefa(db.Model):
```

```
    id = db.Column(db.Integer, primary_key=True)
```

```
    descricao = db.Column(db.String(100), nullable=False)
```

```
    concluida = db.Column(db.Boolean, default=False)
```

```
@app.route('/tarefas', methods=['GET'])
def listar_tarefas():
    tarefas = Tarefa.query.all()

    return jsonify([{"id": t.id, "descricao": t.descricao,
"concluida": t.concluida} for t in tarefas])

@app.route('/tarefas', methods=['POST'])
def criar_tarefa():
    dados = request.json

    nova_tarefa = Tarefa(descricao=dados['descricao'])

    db.session.add(nova_tarefa)

    db.session.commit()

    return jsonify({"id": nova_tarefa.id, "descricao":
nova_tarefa.descricao, "concluida": nova_tarefa.concluida}), 201

if __name__ == '__main__':
    db.create_all()

    app.run(debug=True)
```



Dicas para o Sucesso

- ✓ Comece pequeno e vá aumentando a complexidade.
- ✓ Use Git para versionar cada funcionalidade.
- ✓ Implemente boas práticas de código limpo e reutilizável.
- ✓ Teste cada funcionalidade para garantir que o sistema esteja funcionando corretamente.
- ✓ Documente o projeto para facilitar a manutenção e uso futuro.



Conclusão

Parabéns por chegar até aqui! 🎯

Você agora tem o conhecimento necessário para desenvolver uma aplicação completa com Python.

Este projeto será uma peça valiosa no seu portfólio e poderá abrir portas para oportunidades profissionais.

👉 **Agora é com você! Boa sorte!** 🚀🐍