

VERSAO PROFESSOR

EDUARDO MARQUES



SOBREVIVÊNCIA AO PYTHON

TUDO O QUE VOCÊ PRECISA
SABER PARA NÃO "TRAVAR"
NO CÓDIGO!



UnB | IE | CIC



Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Licenciatura em Computação

APRESENTAÇÃO

Este guia foi desenvolvido com o objetivo de fornecer uma abordagem prática e acessível para o aprendizado de Python. Ele visa atender estudantes, especialmente em cursos técnicos e de introdução à programação, que buscam uma fonte confiável e direta para solucionar problemas comuns, compreender conceitos fundamentais e desenvolver confiança na linguagem Python.

USO E COMPARTILHAMENTO

Esse material é livre para uso e compartilhamento, também pode ser adaptado para diversos usos (Exceto aquilo que não for de completa autoria). Desde que, não tenha propósitos financeiros, assim como, sendo necessário a menção ao material. Para mais informações, clique no link fixado na imagem da licença.



SUMÁRIO

Primeiros	3
Passos	
Estruturas	9
Básicas	
Erros	15
Comuns	
Depurando	27
Código	
Guias	35
Rápidos	
Boas	43
Práticas	
Fluxograma	51
Prático	
Consulta	60
Rápida	
Atenção	68
Final	

SEÇÃO 1

PRIMEIROS PASSOS





PRIMEIROS PASSOS

O que é Python?

Python é uma linguagem de programação versátil, conhecida por sua simplicidade e facilidade de aprendizado. É usada em diversas áreas, como desenvolvimento web, automação, ciência de dados e inteligência artificial. A frase “Python é para todos” resume bem sua acessibilidade!

Por que aprender Python?

- Simplicidade na sintaxe.
- Comunidade ativa e suporte abundante.
- Aplicações práticas em quase todas as áreas de tecnologia.

Como Instalar o Python e Bibliotecas

Aqui, você aprende a configurar seu ambiente de trabalho com Python, independente do sistema operacional.

Instalação no Windows

Baixe o instalador no site oficial:

<https://www.python.org>.

Execute e marque a opção “Add Python to PATH”

Siga as etapas do instalador. Teste no terminal digitando:

```
python --version
```



PRIMEIROS PASSOS

Instalação no macOS

Instale o Python com o comando:

```
brew install python
```

Verificação a instalação com:

```
python3 --version
```

Instalação no Linux

Na maioria das distribuições Linux, Python já vem instalado. Para garantir a versão mais recente, use:

```
sudo apt update  
sudo apt install python3
```

Verificação a instalação com:

```
python3 --version
```

- **Material de Suporte:** Tenha capturas de tela do processo de instalação para mostrar em sala de aula ou em um material complementar.
- **Atividade Extra:** Peça aos alunos para verificar a versão do Python instalada em seus computadores e relatar quaisquer problemas encontrados.



- **Sugestão de Demonstração:** Faça a instalação de uma biblioteca comum, como numpy, ao vivo, mostrando os resultados.
- **Dica de Resolução:** Para problemas no PATH, ensine os alunos a configurar variáveis de ambiente manualmente. É uma habilidade valiosa e frequentemente necessária.
- **Exemplo Real:** Mostre como o erro aparece no terminal para que os alunos possam identificá-lo facilmente.

PRIMEIROS PASSOS

Gerenciador de Bibliotecas

O pip é a ferramenta oficial para instalar bibliotecas no Python. Com o Python instalado, você já tem o pip.

Teste com:

```
pip --version
```

Para instalar uma biblioteca, use:

```
pip install nome_da_biblioteca
```

Macetes e Ferramentas Úteis

Use ambientes virtuais para gerenciar bibliotecas isoladamente:

```
python -m venv meu_ambiente
source meu_ambiente/bin/activate # Linux/macOS
meu_ambiente\Scripts\activate # Windows
```

Principais Problemas e Como Resolver

Problema 1: “Python não é reconhecido como comando”

- **Causa:** O Python não foi adicionado ao PATH.
- **Solução:** Reinstale o Python e marque a opção "Add Python to PATH".



PRIMEIROS PASSOS

Problema 2: Conflito entre versões (Python 2 e Python 3)

- **Causa:** Duas versões instaladas no sistema.
- **Solução:** Especifique a versão ao executar comandos:

```
python3 script.py
```

Problema 3: Permissões no Linux

- **Causa:** O usuário não tem permissão para instalar pacotes.
- **Solução:** Use o comando com sudo:

```
sudo apt install python3
```

Entendeu mesmo ou Só copiou?

Ao final desta seção, revise o que foi feito:

Você instalou o Python corretamente? Teste rodando:

```
print("Parabéns! Você configurou Python com sucesso!")
```

Consegue instalar uma biblioteca e usá-la? Experimente:

```
pip install numpy
```

```
import numpy as np
print(np.array([1, 2, 3]))
```

PRIMEIROS PASSOS

Desafios Rápidos

Escreva um programa que peça ao usuário para digitar seu nome e mostre a mensagem:

```
Olá, [nome]! Bem-vindo ao Python.
```

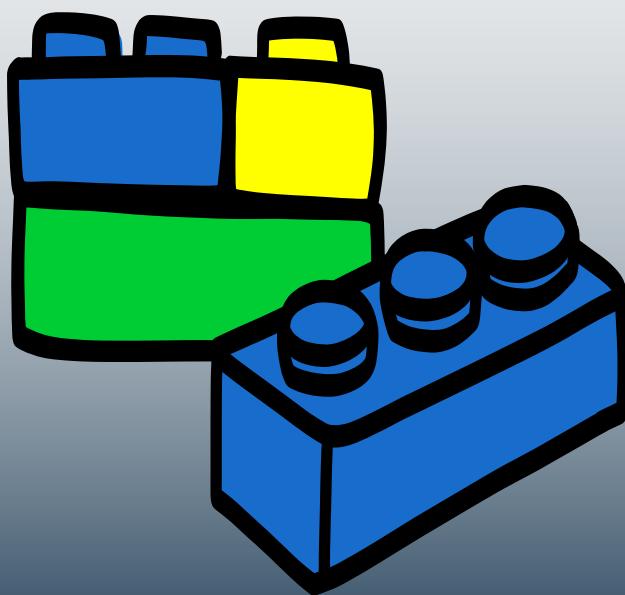
Instale a biblioteca requests e use-a para fazer uma requisição a uma página web:

```
import requests
response = requests.get("https://www.python.org")
print(response.status_code)
```

- **Sugestão de Avaliação:** Peça aos alunos para criar um programa simples que utilize uma entrada do usuário e gere uma saída.
- Exemplo:
`input("Qual é seu nome?")`.

SEÇÃO 2

ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON





ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON

O que são Estruturas Básicas?

Estruturas básicas são os blocos fundamentais de qualquer programa em Python. Elas incluem a manipulação de dados, controle de fluxo e organização do código, permitindo criar programas funcionais e eficientes.

Nesta seção, você aprenderá as estruturas fundamentais que são a base da programação em Python. Esses comandos serão seus aliados no dia a dia, desde operações matemáticas simples até manipulações avançadas de dados.

Dica: Entender bem esses conceitos ajudará a construir programas mais eficientes e organizados.

Manipulação de Dados

Tipos de Dados

Python possui diferentes tipos de dados, que podem ser usados dependendo da necessidade:

- **Inteiros (int):** Números inteiros (ex.: 5, -3).
- **Números de ponto flutuante (float):** Números decimais (ex.: 3.14, -0.01).
- **Texto (str):** Sequência de caracteres (ex.: "Python").
- **Booleanos (bool):** Verdadeiro ou Falso (ex.: *True*, *False*).



ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON

Operações com Dados

```
# Soma, subtração, multiplicação e divisão
a = 10
b = 3
print(a + b) # Soma
print(a - b) # Subtração
print(a * b) # Multiplicação
print(a / b) # Divisão (resultado em float)
```

Conversão de Tipos

```
# Convertendo tipos
idade = "25"
idade = int(idade) # Converte string para inteiro
preco = 19.99
preco_inteiro = int(preco) # Converte float para inteiro
```

Manipulação de Strings

As strings são um dos tipos de dados mais usados em Python. Aqui estão alguns comandos essenciais para trabalhar com elas:

ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON

Operações básicas

```
# Concatenar strings
nome = "João"
sobrenome = "Silva"
print(nome + " " + sobrenome) # Resultado: João Silva

# Repetir uma string
print("Python! " * 3) # Resultado: Python! Python! Python!
```

Métodos Úteis

```
frase = "Olá, mundo!"
print(frase.upper()) # Resultado: OLÁ, MUNDO!
print(frase.lower()) # Resultado: olá, mundo!
print(frase.replace("mundo", "Python")) # Resultado: Olá, Python!
```

Fatiamento de Strings

```
# Acessando partes da string
texto = "Aprendendo Python"
print(texto[0:10]) # Resultado: Aprendendo
print(texto[-6:]) # Resultado: Python
```

ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON

Operadores Essenciais

Operadores Aritméticos

```
x = 15  
y = 4  
print(x + y) # Adição  
print(x - y) # Subtração  
print(x * y) # Multiplicação  
print(x / y) # Divisão  
print(x % y) # Módulo (resto da divisão)
```

Operadores de Comparação

```
# Retornam True ou False  
a = 10  
b = 20  
print(a == b) # Igual (False)  
print(a != b) # Diferente (True)  
print(a > b) # Maior que (False)
```



ESTRUTURAS BÁSICAS DA LINGUAGEM PYTHON

Operadores Lógicos

```
# Combinam condições  
idade = 25  
salario = 3000  
print(idade > 18 and salario > 2000) # True  
print(idade < 18 or salario > 2000) # True  
print(not (idade > 18)) # False
```

Desafios Rápidos

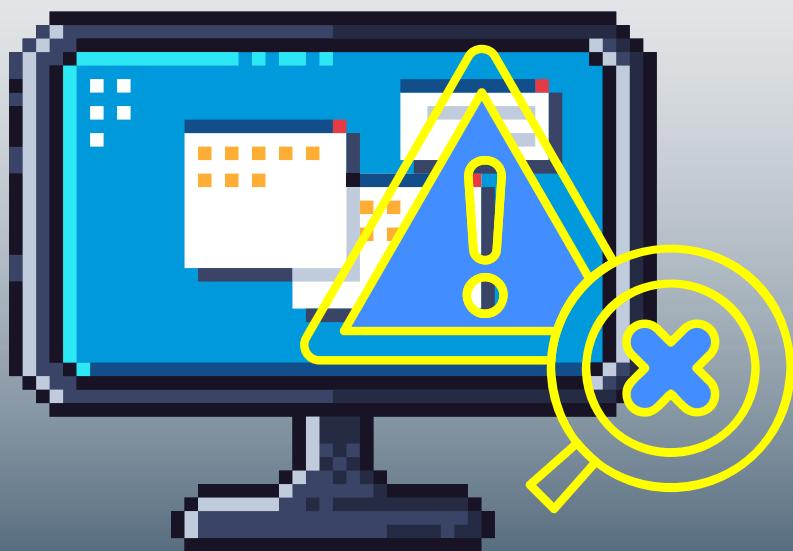
Crie um programa que receba o nome de uma pessoa e diga se ele contém mais de 5 caracteres.

Use operadores de comparação para verificar se um número é par ou ímpar.

Manipule uma frase para transformá-la em letras maiúsculas e, depois, inverta seus caracteres.

SEÇÃO 3

ERROS COMUNS E COMO VOCÊ VAI RESOLVER



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

- **Introdução à Aula:** Explique que erros são uma parte natural do aprendizado de programação e que entendê-los é uma habilidade fundamental.
- **Dica Pedagógica:** Realize um exercício inicial onde os alunos executem um código com erros intencionais para que possam observar as mensagens de erro e discuti-las.
- **Exemplo Prático:** Mostre uma mensagem de erro gerada pelo Python no terminal e explique cada parte dela.

Introdução

Erros acontecem, e isso faz parte do aprendizado. O Python é conhecido por oferecer mensagens de erro claras, mas entendê-las pode ser desafiador no início. Nesta seção, vamos explorar os erros mais comuns, suas causas e como solucioná-los.

SyntaxError: O Básico Está Errado

As strings são um dos tipos de dados mais usados em Python. Aqui estão alguns comandos essenciais para trabalhar com elas:

O que é?

Ocorre quando há algo de errado na estrutura do código, como falta de pontuação ou comandos mal escritos.

Exemplo:

```
print("Olá, Mundo! # Esqueci de fechar as aspas
```

Mensagem de erro:

```
SyntaxError: EOL while scanning string literal
```



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Como resolver:

- Verifique a linha mencionada no erro.
 - Certifique-se de que todas as aspas, parênteses e colchetes estão fechados corretamente.
-

NameError: Variável Não Definida

O que é?

Acontece quando tentamos usar uma variável que não foi declarada ou está escrita de forma incorreta.

Exemplo:

```
print(nome)
```

Mensagem de erro:

```
NameError: name 'nome' is not defined
```



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Como resolver:

- Confirme se a variável foi definida antes de usá-la.
- Verifique se o nome da variável foi escrito corretamente.

• **Dica Prática:**
Explique a diferença entre "tab" e "espaços". Reforce que o uso de 4 espaços é o padrão recomendado.

• **Atividade Extra:**
Peça aos alunos para corrigirem trechos de código com erros de indentação.

IndentationError: Problemas com Recuo

O que é?

Python usa indentação para definir blocos de código. Um erro de indentação ocorre quando o recuo está inconsistente.

Exemplo:

```
if True:  
    print("Indentação errada")
```

Mensagem de erro:

```
IndentationError: expected an indented block
```

ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Como resolver:

- Use sempre o mesmo tipo de recuo (espaços ou tabs).
 - No caso de espaços, use 4 espaços por nível de indentação.
-

TypeError: Tipos de Dados Incompatíveis

O que é?

Aparece quando você tenta usar um operador ou função em tipos de dados incompatíveis.

Exemplo:

```
print("O resultado é: " + 10)
```

Mensagem de erro:

```
TypeError: can only concatenate str (not "int") to str
```

Como resolver:

- Converta os tipos para serem compatíveis:

```
print("O resultado é: " + str(10))
```

ERROS COMUNS E COMO VOCÊ VAI RESOLVER

ValueError: Valor Inesperado

O que é?

Surge quando uma função recebe um valor correto, mas que não é válido para o contexto.

Exemplo:

```
int("Python")
```

Mensagem de erro:

```
ValueError: invalid literal for int() with base 10: 'Python'
```

Como resolver:

- Certifique-se de que o valor está no formato correto para a função.

ZeroDivisionError: Divisão por Zero

O que é?

Ocorre ao tentar dividir um número por zero.



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Exemplo:

```
resultado = 10 / 0
```

Mensagem de erro:

```
ZeroDivisionError: division by zero
```

Como resolver:

- Verifique se o divisor é diferente de zero antes de realizar a divisão:

```
if divisor != 0:  
    resultado = 10 / divisor
```

IndexError: Índice Fora do Alcance

O que é?

Aparece quando tentamos acessar uma posição inexistente em uma lista.

ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Exemplo:

```
lista = [1, 2, 3]
print(lista[5])
```

Mensagem de erro:

```
IndexError: list index out of range
```

Como resolver:

- Use a função `len()` para verificar o tamanho da lista antes de acessar um índice:

```
if indice < len(lista):
    print(lista[indice])
```

KeyError: Chave Inexistente

O que é?

Ocorre quando tentamos acessar uma chave inexistente em um dicionário.

ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Exemplo:

```
dicionario = {"nome": "João"}  
print(dicionario["idade"])
```

Mensagem de erro:

```
KeyError: 'idade'
```

Como resolver:

- Verifique se a chave existe antes de acessá-la:

```
if "idade" in dicionario:  
    print(dicionario["idade"])
```

AttributeError: Atributo Não Encontrado

O que é?

Acontece quando tentamos acessar um atributo ou método que não existe para um determinado objeto.



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Exemplo:

```
lista = [1, 2, 3]
lista.append(4)
lista.sorter()
```

Mensagem de erro:

```
AttributeError: 'list' object has no attribute 'sorter'
```

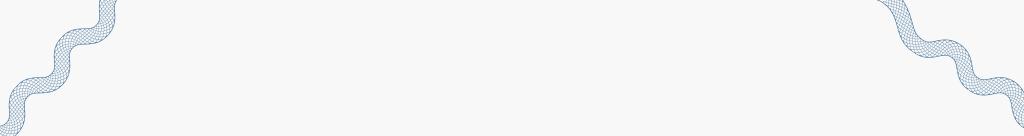
Como resolver:

- Verifique se o método existe para aquele objeto.

ImportError: Módulo Não Encontrado

O que é?

Aparece ao tentar importar um módulo que não está instalado ou não existe.



ERROS COMUNS E COMO VOCÊ VAI RESOLVER

Exemplo:

```
import numpyy
```

Mensagem de erro:

```
ImportError: No module named 'numpyy'
```

Como resolver:

- Certifique-se de que o módulo foi digitado corretamente.
- Instale o módulo antes de importá-lo:

```
pip install numpy
```

Dicas Finais

- Leia a mensagem de erro com calma. Ela geralmente indica o problema e onde ele ocorreu.

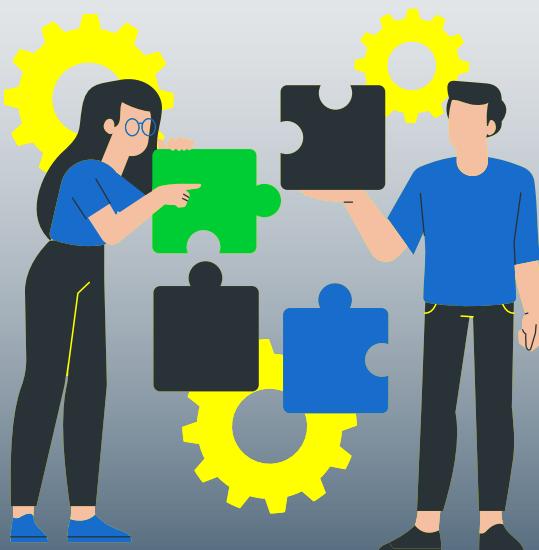
ERROS COMUNS E COMO VOCÊ VAI RESOLVER

- Use ferramentas como **try** e **except** para tratar erros e evitar que o programa quebre completamente:

```
try:  
    resultado = 10 / 0  
except ZeroDivisionError:  
    print("Divisão por zero não é permitida.")
```

SEÇÃO 4

SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS





SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

- **Introdução em Aula:** Explique que a depuração é uma combinação de técnica e paciência. Mostre que até programadores experientes enfrentam problemas, mas o importante é saber onde e como procurar soluções.

- **Dica Adicional:** Peça aos alunos para compartilhar como normalmente lidam com erros antes de aprenderem técnicas de depuração. Isso pode abrir espaço para discussões produtivas.

Introdução

Depurar um código é uma das habilidades mais importantes para qualquer programador. Isso envolve identificar, entender e corrigir erros no programa. Aqui, você aprenderá ferramentas e técnicas para solucionar problemas de forma eficiente e evitar frustrações.

Dica Geral: Nunca tenha medo de errar! Cada erro é uma oportunidade para aprender.

Como Interpretar Mensagens de Erro

As mensagens de erro em Python são detalhadas e indicam:

- O tipo de erro: Como **SyntaxError**, **NameError**, etc.
- A linha do erro: Onde ocorreu o problema.
- Uma explicação básica: Informações sobre o que deu errado.

Estrutura de uma Mensagem de Erro:

```
Traceback (most recent call last):
  File "meu_programa.py", line 10, in <module>
    print(nome)
NameError: name 'nome' is not defined
```

- **Dica Prática:**
Mostre exemplos reais de mensagens de erro e peça aos alunos para interpretarem as informações.
Explique que ler as mensagens "de baixo para cima" pode ajudar a localizar o erro principal.
- **Sugestão de Demonstração:**
Crie um programa simples e insira vários **print()** ao longo do código para demonstrar como monitorar o fluxo e os valores de variáveis.
- **Dica de Boas Práticas:**
Ensine os alunos a removerem os **print()** de depuração após resolverem os problemas, para manter o código limpo.

SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

Como interpretar:

1. Leia de baixo para cima: O último item da mensagem geralmente é o erro principal (**NameError**).
2. Identifique a linha: Procure o problema na linha mencionada (**line 10**).
3. Pesquise a solução: Muitas vezes, um simples Google do erro (com a mensagem exata) traz respostas rápidas.

Exercício Prático: Simule erros no código propositalmente para entender melhor as mensagens de erro.

print() é Seu Melhor Amigo

Usar **print()** é a técnica de depuração mais simples e eficaz. Ele ajuda a verificar o valor de variáveis e o fluxo do programa.

Como usar **print()** para depuração:

Verificar valores de variáveis:

```
x = 5  
y = x + 10  
print(f"x: {x}, y: {y}")
```

SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

Acompanhar o fluxo do programa:

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

Detectar pontos de falha:

```
if total < 0:
    print("Erro: Total negativo")
```

Boas práticas com *print()*:

Use mensagens descritivas:

```
print("Início do processamento...")
```

Remova os *print()* de depuração após corrigir o problema.

Ferramentas Úteis para Depuração

Além do *print()*, existem ferramentas poderosas que tornam a depuração mais eficiente.



SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

Debugger Integrado

A maioria das IDEs, como PyCharm, VSCode e Thonny, possui ferramentas de depuração integradas.

Como usar:

- **Debugger Integrado:**
Explique como usar breakpoints em IDEs como VSCode ou PyCharm.
Demonstre ao vivo como pausas no código permitem verificar variáveis em tempo real
- Insira um ponto de interrupção (breakpoint) no código.
- Execute o programa em modo debug.
- Analise valores e o fluxo do programa passo a passo.

Biblioteca *pdb* (Python Debugger)

O Python possui um depurador embutido chamado *pdb*.

Como usar:

```
import pdb  
pdb.set_trace()
```

Comandos básicos do *pdb*:

- *n*: Executa a próxima linha.
- *c*: Continua a execução até o próximo breakpoint.
- *q*: Sai do depurador.

SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

Logs com *logging*

Para depuração mais estruturada, use a biblioteca *logging*:

Vantagens:

- Permite salvar mensagens de depuração em arquivos.
- É mais profissional do que usar apenas *print()*.

Exemplo:

```
import logging
logging.basicConfig(level=logging.DEBUG)
logging.debug("Mensagem de depuração")
```

Ferramentas Externas

- **PyCharm Debugger:** Interface gráfica para depuração passo a passo.
- **Jupyter Notebook:** Permite executar células separadamente e inspecionar valores.
- **Visual Studio Code:** Oferece ferramentas de análise em tempo real.

SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

Estratégias Avançadas

Divida e Conquiste

Ao depurar um código longo, isole partes menores para identificar onde o erro ocorre:

```
def calcular_total(valores):
    print(f"Valores recebidos: {valores}")
    soma = sum(valores)
    print(f"Soma calculada: {soma}")
    return soma
```

Teste com Dados Simples

Use entradas menores e simples para verificar o funcionamento básico do programa.

```
valores = [1, 2, 3] # Teste simples
```

Adicione Testes Automatizados

Use bibliotecas como unittest ou pytest para criar testes que garantam o funcionamento correto do código:



SE TORNE MESTRE EM RESOLVER SEUS PROBLEMAS

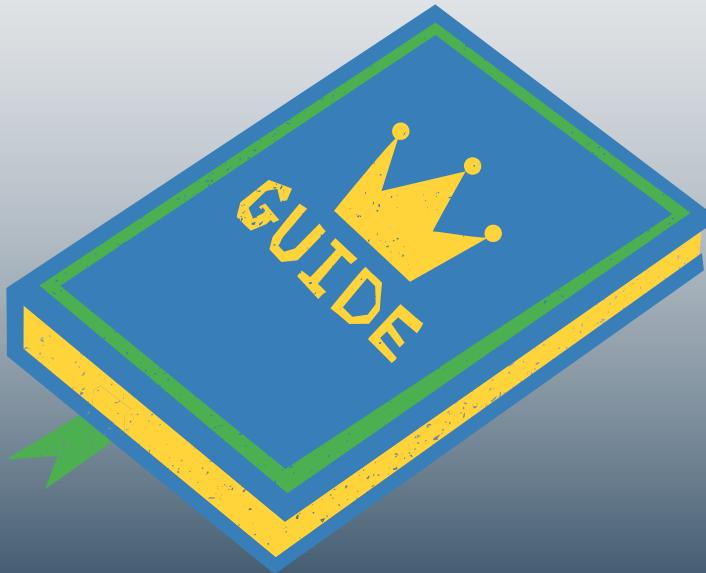
```
def soma(a, b):  
    return a + b  
  
def test_soma():  
    assert soma(2, 3) == 5
```

Dicas Finais

- Leia o código com atenção antes de executar.
- Sempre execute o código em partes menores para identificar erros rapidamente.
- Use a comunidade Python (como Stack Overflow) para encontrar soluções para problemas mais complexos.

SEÇÃO 5

GUIAS RÁPIDOS



GUIAS RÁPIDOS

- **Contextualização:**

Destaque a importância de os alunos terem acesso a uma fonte de referência

rápida durante a codificação.

Explique que os guias rápidos servem como "atalhos" para soluções cotidianas.

- **Dica Pedagógica:**

Sugira que os alunos imprimam ou mantenham esta seção aberta enquanto trabalham em projetos, para consulta rápida.

Introdução

Esta seção traz instruções práticas e diretas para você trabalhar com conceitos fundamentais do Python. O objetivo é acabar com as dúvidas e simplificar a aplicação desses conceitos no dia a dia. Cada guia aborda um tópico específico com explicações claras e exemplos aplicáveis.

Como Trabalhar com Listas e Dicionários

Listas e dicionários são fundamentais em Python para armazenar e manipular dados de forma eficiente.

Listas

- O que são?

Uma lista é uma coleção ordenada de itens que podem ser de diferentes tipos.

- Operações Comuns com Listas:

GUIAS RÁPIDOS



- **Introdução**

Didática:

Explique as diferenças entre listas (coleções ordenadas) e dicionários (pares de chave-valor). Utilize exemplos cotidianos, como listas de compras ou um catálogo de produtos, para ilustrar.

- **Atividade**

Sugerida:

Peça aos alunos para criarem uma lista e um dicionário baseados em informações pessoais (por exemplo, hobbies ou características).

```
frutas = ["maçã", "banana", "laranja"]

# Acessar elementos
print(frutas[0]) # Resultado: maçã

# Adicionar itens
frutas.append("uva")

# Remover itens
frutas.remove("banana")

# Ordenar
frutas.sort()
```

- Iterar sobre uma Lista:

```
for fruta in frutas:
    print(f"Eu gosto de {fruta}")
```

Dicionários

- O que são?

Um dicionário armazena pares de chave-valor, ideal para buscar informações rapidamente.

- Criar e Manipular Dicionários:



GUIAS RÁPIDOS

```
aluno = {"nome": "João", "idade": 20, "curso": "Python"}  
  
# Acessar valores  
print(aluno["nome"]) # Resultado: João  
  
# Adicionar ou Atualizar  
aluno["nota"] = 9.5  
  
# Remover chave-valor  
del aluno["curso"]
```

- Iterar sobre um Dicionário:

```
for chave, valor in aluno.items():  
    print(f'{chave}: {valor}')
```

Estruturas de Controle: Decisões e Repetições

Estruturas Condicionais (Decisões)

- *if, elif e else:*

```
idade = 18  
if idade < 18:  
    print("Menor de idade")  
elif idade == 18:  
    print("Exatamente 18!")  
else:  
    print("Maior de idade")
```



GUIAS RÁPIDOS

- **Dica para Laços:**
Explique que o `for` é usado para iterações conhecidas, enquanto o `while` é mais flexível e depende de uma condição para continuar.

- Aninhamento de Condições:

```
nota = 85
if nota >= 90:
    print("Aprovado com mérito")
else:
    if nota >= 70:
        print("Aprovado")
    else:
        print("Reprovado")
```

-
- **Dica Pedagógica:**
Explique que funções são como "receitas" – uma vez definidas, podem ser usadas repetidamente com diferentes ingredientes (parâmetros).

Estruturas de Repetição

- Laço `for`:

```
for numero in range(1, 6):
    print(numero)
```

- Laço `while`:

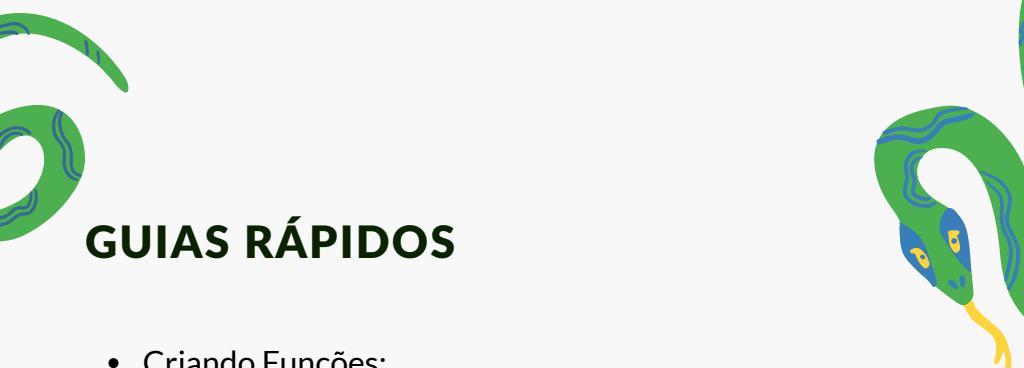
```
contador = 0
while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
```

-
- **Atividade Proposta:**
Peça aos alunos para criarem uma função que calcule o quadrado de um número.

Funções em 2 Minutos

O que são Funções?

Funções são blocos de código reutilizáveis que ajudam a organizar e simplificar programas.



GUIAS RÁPIDOS

- Criando Funções:

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

- Parâmetros Padrão:

```
def saudacao(nome="Visitante"):  
    return f"Bem-vindo, {nome}!"  
  
print(saudacao()) # Resultado: Bem-vindo, Visitante!
```

- Funções com Múltiplos Parâmetros:

```
def soma(a, b):  
    return a + b  
  
print(soma(3, 4))
```

Trabalhando com Arquivos

Leitura de Arquivos



GUIAS RÁPIDOS

- **Introdução**

Didática:

Mostre os modos de abrir arquivos: leitura (r), escrita (w) e adição (a).

- **Dica Adicional:**

Explique a importância de sempre fechar os arquivos ou usar o comando with para evitar erros.

- Abrindo e Lendo um Arquivo:

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

- Ler Linha por Linha:

```
with open("dados.txt", "r") as arquivo:  
    for linha in arquivo:  
        print(linha.strip())
```

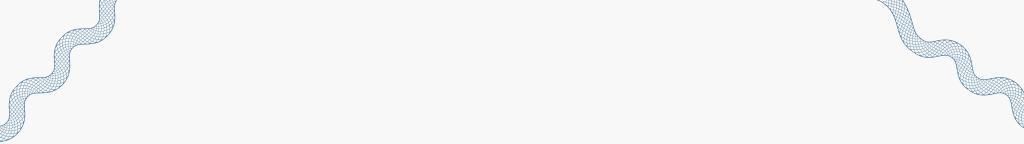
Escrevendo em Arquivos

- Criar ou Sobrescrever Arquivo:

```
with open("dados.txt", "w") as arquivo:  
    arquivo.write("Escrevendo em um arquivo!")
```

- Adicionar ao Final de um Arquivo:

```
with open("dados.txt", "a") as arquivo:  
    arquivo.write("\nNova linha adicionada!")
```



GUIAS RÁPIDOS

Dicas finais

- **Organize seu Código:** Use funções para separar operações com listas, dicionários ou arquivos.
- **Teste em Partes:** Trabalhe com exemplos pequenos antes de integrá-los a projetos maiores.
- **Pratique:** Crie programas que envolvam listas, dicionários e manipulação de arquivos para fixar o aprendizado.

SEÇÃO 6

UM POUCO DE ESTILO: FACILITE SUA VIDA





UM POUCO DE ESTILO: FACILITE SUA VIDA

Introdução

Escrever código limpo e organizado é tão importante quanto fazer o código funcionar.

Boas práticas não só facilitam a leitura e manutenção do código, mas também tornam seu trabalho mais eficiente e profissional. Nesta seção, você aprenderá a adotar um estilo de programação que simplifica sua vida e a dos outros que possam ler ou usar seu código.

• Dica Pedagógica:

Mostre exemplos de códigos desorganizados e como isso dificulta a manutenção. Depois, compare com versões organizadas e bem documentadas.

• Dica Adicional:

Sugira que variáveis podem ser substantivos (idade) e funções podem ser verbos (calcular_media).

Nomes Significativos

Por que usar nomes significativos?

Nomes claros ajudam você e outros desenvolvedores a entenderem o propósito de uma variável, função ou classe sem precisar ler o código inteiro.

Exemplo: Nomes Ruins

```
x = 100
y = 5
z = x / y
print(z) # O que significa z?
```



UM POUCO DE ESTILO: FACILITE SUA VIDA

Exemplo: Nomes Significativos

```
total_vendas = 100
numero_dias = 5
media_diaria = total_vendas / numero_dias
print(media_diaria) # Fica claro o que está sendo calculado
```

Dicas:

- Use substantivos para variáveis (*idade, nome_do_aluno*).
- Use verbos para funções (*calcular_media, imprimir_relatorio*).
- Evite abreviações confusas (*ctt* para " contato" pode ser difícil de entender).

Comentários Claros

Quando usar comentários?

Use comentários para explicar por que algo está sendo feito, não o que está sendo feito (o código já explica o que faz).

UM POUCO DE ESTILO: FACILITE SUA VIDA

Exemplo Ruim

```
x = 10 # Atribuindo 10 a x
```

Exemplo Bom

```
# Calcula a média de vendas diárias  
media_diarria = total_vendas / numero_dias
```

Tipos de Comentários:

Comentários de Explicação:

```
# Verifica se o usuário tem mais de 18 anos para acessar  
if idade >= 18:  
    print("Acesso permitido")
```

Dicas:

- Seja objetivo: mantenha os comentários curtos e diretos.
- Atualize os comentários se o código mudar.

UM POUCO DE ESTILO: FACILITE SUA VIDA

Docstrings (Para funções e classes):

```
def calcular_media(valores):
    """
    Calcula a média de uma lista de números.

    :param valores: Lista de números
    :return: Média dos números
    """

    return sum(valores) / len(valores)
```

Organização no Código

Por que organizar o código?

Um código organizado é mais fácil de ler, entender e manter. Isso é especialmente importante quando o programa cresce ou quando mais pessoas trabalham no mesmo projeto.

Estrutura Recomendada para um Script Python:

- Importações no início:

```
import os
import math
```



UM POUCO DE ESTILO: FACILITE SUA VIDA

- **Explicação Inicial:**
Explique que o PEP 8 é um guia oficial de estilo para Python e cobre desde a indentação até o comprimento máximo de linhas.
- **Atividade Proposta:**
Peça aos alunos para revisar um trecho de código bagunçado e organizá-lo conforme o PEP 8.

- Definições de Funções:

```
def calcular_media(valores):  
    return sum(valores) / len(valores)
```

- Código Principal:

```
if __name__ == "__main__":  
    valores = [10, 20, 30]  
    print(calcular_media(valores))
```

Dicas de Formatação:

PEP 8:

Siga as diretrizes de estilo do Python:

- Use 4 espaços para indentação (nunca misture tab e espaço).
- Deixe uma linha em branco entre funções.
- Limite o comprimento das linhas a 79 caracteres.

Exemplo Antes e Depois

- Antes (Desorganizado):

```
def calc(a):return sum(a)/len(a) #calcula media  
v=[10,20,30];print(calc(v))
```

UM POUCO DE ESTILO: FACILITE SUA VIDA

- Depois (Organizado):

```
def calcular_media(valores):  
    """  
        Calcula a média de uma lista de números.  
    """  
  
    return sum(valores) / len(valores)  
  
valores = [10, 20, 30]  
print(calcular_media(valores))
```

Uso de Módulos e Funções:

Divida o código em funções e arquivos para facilitar a leitura:

- Crie funções para operações específicas.
- Use módulos para separar funcionalidades.

Desafios Rápidos

- Reescreva o seguinte código usando nomes significativos e boas práticas:

```
x = 10  
y = 20  
print(x*y)
```

UM POUCO DE ESTILO: FACILITE SUA VIDA

- Adicione comentários claros a este código:

```
def soma(a, b):  
    return a + b  
resultado = soma(5, 7)  
print(resultado)
```

- Organize o código de acordo com a PEP 8:

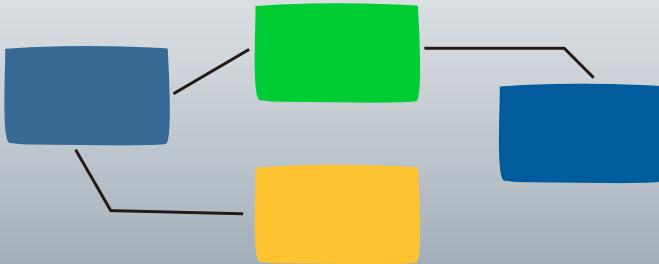
```
def calc(v):return sum(v)/len(v)  
v=[10,20,30]  
print(calc(v))
```

Dicas Finais

- Pratique a PEP 8: Instale ferramentas como flake8 para verificar o estilo do código.
- Comunique-se no Código: Imagine que alguém precisará ler seu código no futuro (incluindo você!).

SEÇÃO 7

FLUXOGRAMAS RÁPIDOS





- **Introdução ao Tema:**
Explique que fluxogramas são essenciais para o desenvolvimento lógico antes da implementação do código. Eles ajudam os alunos a visualizar o fluxo de controle, o que é especialmente útil para iniciantes.

- **Atividade Proposta:**
Peça aos alunos para desenharem fluxogramas para problemas simples, como "verificar se um número é par ou ímpar".

- **Dica Prática:**
Use softwares gratuitos, como Lucidchart ou Draw.io, para criar fluxogramas em aula.

FLUXOGRAMAS RÁPIDOS

Introdução

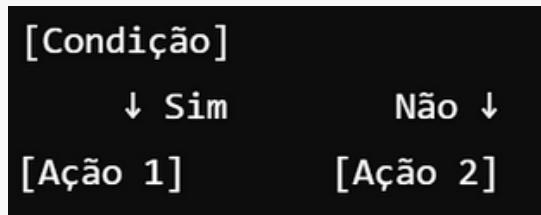
Fluxogramas são ferramentas visuais que ajudam a planejar e compreender o fluxo de um programa antes mesmo de começar a codificar. Eles são úteis para resolver problemas lógicos e identificar os comandos corretos para cada situação.

Diagramas para Lógica de Decisões

Decisão Simples: “Se/Se Não”

Se você precisa tomar uma decisão com base em uma condição, use a estrutura *if/else*.

Exemplo Visual:



FLUXOGRAMAS RÁPIDOS

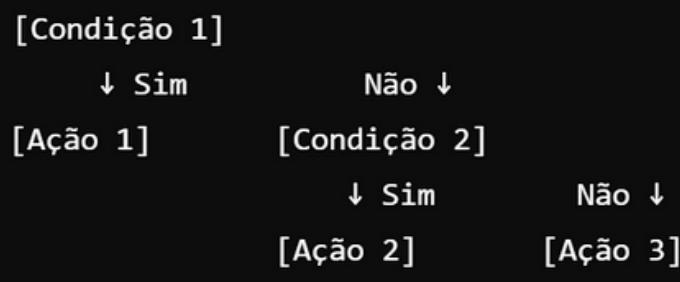
Exemplo Prático no Código:

```
idade = 20  
if idade >= 18:  
    print("Maior de idade")  
else:  
    print("Menor de idade")
```

Decisão Múltipla: “Se/Elif/Else”

Se há várias condições a serem verificadas, use *elif*.

Exemplo Visual:



FLUXOGRAMAS RÁPIDOS

Exemplo Prático no Código:

```
nota = 85  
if nota >= 90:  
    print("Aprovado com mérito")  
elif nota >= 70:  
    print("Aprovado")  
else:  
    print("Reprovado")
```

Diagramas para Repetições (Loops)

Laço “Enquanto” (**while**)

Use **while** quando você não sabe quantas vezes precisará repetir, mas sabe a condição.



FLUXOGRAMAS RÁPIDOS

Exemplo Visual:

```
[Condição Inicial]
    ↓ Sim
[Ação Repetida]
    ↓
[Atualiza Condição] → [Verifica de Novo]
    ↓ Não
[Finaliza]
```

Exemplo Prático no Código:

```
contador = 0
while contador < 5:
    print(f"Contador: {contador}")
    contador += 1
```

Laço “Para” (for)

Use **for** quando você sabe o número de iterações ou precisa percorrer elementos de uma lista.

FLUXOGRAMAS RÁPIDOS

Exemplo Visual:

[Lista ou Faixa de Valores]



[Executa Ação para Cada Item]



[Finaliza]

Exemplo Prático no Código:

```
for numero in range(1, 6):
    print(f"Número: {numero}")
```



FLUXOGRAMAS RÁPIDOS

Dicas de Comandos por Objetivo

Se você está tentando...

- Uso da Tabela:**
Incentive os alunos a usarem esta tabela como um guia rápido durante a codificação.
- Atividade Proposta:**
Dê objetivos práticos (ex.: "Armazene as notas de 5 alunos") e peça aos alunos para identificar os comandos corretos.

Objetivo	Comando	Exemplo
Repetir algo N vezes	<code>for</code>	<code>for i in range(10):</code>
Continuar até uma condição mudar	<code>while</code>	<code>while saldo > 0:</code>
Verificar várias condições	<code>if/elif/else</code>	<code>if x > 10:</code>
Armazenar vários valores ordenados	<code>list</code>	<code>minha_lista = [1, 2, 3]</code>
Armazenar pares chave-valor	<code>dict</code>	<code>meu_dicionario = {"chave": "valor"}</code>
Manipular texto	<code>str</code>	<code>"texto".upper()</code>
Ler um arquivo	<code>open()</code>	<code>with open("arquivo.txt") as f:</code>
Adicionar itens a uma lista	<code>.append()</code>	<code>lista.append("novo item")</code>



FLUXOGRAMAS RÁPIDOS

Desafios Rápidos

Use o Fluxograma para Planejar um Programa

Crie um fluxograma que represente este problema:

- O usuário insere sua idade.
- Se for menor de 18 anos, exiba “Menor de idade”.
- Se for maior ou igual a 18 anos, exiba “Maior de idade”.

Transforme o Fluxograma em Código:

Baseie-se no fluxograma para escrever um programa funcional em Python.

Desafio com Repetição:

Planeje um fluxograma para este problema:

- O programa soma números inseridos pelo usuário até que o valor inserido seja 0.
- Exiba o total no final.

FLUXOGRAMAS RÁPIDOS

Código esperado:

```
total = 0
while True:
    numero = int(input("Digite um número (0 para sair): "))
    if numero == 0:
        break
    total += numero
print(f"Total: {total}")
```

Conclusão

Os fluxogramas são ferramentas poderosas para estruturar ideias antes de codificar. Eles ajudam a evitar erros e tornam o processo de programação mais organizado. Use fluxogramas sempre que enfrentar problemas complexos ou precisar explicar a lógica do código para outras pessoas.

SEÇÃO 8

S.O.S PYTHON





- **Introdução Pedagógica:** Explique que esta seção serve como um "kit de primeiros socorros" para programação em Python. Os alunos podem utilizá-la para resolver problemas recorrentes de forma independente.

S.O.S PYTHON

Introdução

Nesta seção, reunimos soluções rápidas para os problemas mais comuns enfrentados durante o aprendizado de Python. Seja para resolver um erro inesperado ou relembrar um comando essencial, você encontrará aqui um suporte imediato.

Problema + Solução Simples

Problema 1: "Meu loop não para!"

- **Causa Comum:** A condição do loop nunca é falsa.
- **Solução:** Verifique se a condição do loop está sendo alterada corretamente.

```
contador = 0
while contador < 5:
    print(f"Contador: {contador}")
    contador += 1 # Sem isso, o loop seria infinito
```

- **Explicação Adicional:** Discuta com os alunos como as condições de parada funcionam em `while`.



S.O.S PYTHON

Problema 2: "Recebo None como saída."

- **Causa Comum:** Sua função não está retornando nenhum valor.
- **Solução:** Adicione a palavra-chave **return** para retornar um resultado.

```
def saudacao(nome):  
    return f"Olá, {nome}!"  
  
print(saudacao("Ana")) # Agora retorna algo
```

Problema 3: "Erro de divisão por zero."

- **Causa Comum:** Tentativa de dividir por 0.
- **Solução:** Adicione uma verificação antes de realizar a divisão.

```
divisor = 0  
if divisor != 0:  
    resultado = 10 / divisor  
else:  
    print("Erro: divisor não pode ser zero")
```

S.O.S PYTHON

Problema 4: "Erro *KeyError* ao acessar um dicionário."

- **Causa Comum:** Tentativa de acessar uma chave inexistente.
- **Solução:** Verifique se a chave existe antes de acessá-la.

```
dicionario = {"nome": "João"}  
print(dicionario.get("idade", "Chave não encontrada"))
```

Problema 5: "Erro de indentação no código."

- **Causa Comum:** Uso inconsistente de espaços e tabulações.
- **Solução:** Use um editor que padronize o recuo. No Python, prefira 4 espaços por nível de indentação.

```
if True:  
    print("Indentação correta")
```

Problema 6: "Por que minha lista não atualiza?"

- **Causa Comum:** Atribuição incorreta ou falta de referência.
- **Solução:** Modifique a lista diretamente.

S.O.S PYTHON

```
lista = [1, 2, 3]
lista.append(4)
print(lista) # Resultado: [1, 2, 3, 4]
```

Cheatsheets Rápidos

Os cheatsheets são resumos rápidos e práticos para lembrar comandos e operações essenciais.

Manipulação de Strings

Operações	Exemplo	Resultado
Converter para maiúsculas	"python".upper()	"PYTHON"
Substituir texto	"Olá, mundo".replace("mundo", "Python")	"Olá, Python"
Dividir texto	"1,2,3".split(",")	['1', '2', '3']



- **Uso Prático:**
Mostre como esses comandos são aplicáveis em situações reais, como processamento de dados de texto.

- **Dica Extra:**
Explique a diferença entre **append** e **extend** para evitar confusões comuns.

- **Sugestão de Atividade:**
Divida a turma em pequenos grupos e peça que cada grupo resolva diferentes versões deste desafio, utilizando funções e loops.

S.O.S PYTHON

Operações com Listas

Operações	Exemplo	Resultado
Adicionar item	<code>lista.append(4)</code>	[1, 2, 3, 4]
Remover item	<code>lista.remove(2)</code>	[1, 3]
Tamanho da lista	<code>len(lista)</code>	3

Estruturas de Controle

Operações	Exemplo	Resultado
Condicional	<code>if x > 10:</code>	Executa se a condição for verdadeira
Laço for	<code>for i in range(5):</code>	Repete de 0 a 4
Laço while	<code>while x < 10:</code>	Repete enquanto a condição for verdadeira

S.O.S PYTHON

Funções

Operações	Exemplo	Resultado
Definir função	<code>def soma(a, b):</code>	Define uma função
Retornar valor	<code>return a + b</code>	Retorna o resultado
Chamar função	<code>soma(3, 4)</code>	Executa a função

Manipulação de Arquivos

Operações	Exemplo	Resultado
Ler arquivo	<code>with open("arquivo.txt") as f: print(f.read())</code>	Lê o conteúdo do arquivo
Escrever no arquivo	<code>with open("arquivo.txt", "w") as f: f.write("Olá!")</code>	Escreve no arquivo

S.O.S PYTHON

Desafios

Identifique e corrija os erros no seguinte código:

```
if True  
    print("Erro de sintaxe")
```

Use o cheatsheet para criar um programa que:

- Leia uma lista de números do usuário.
- Remova números negativos e exiba os números restantes.

CONSIDERAÇÕES FINAIS

EDUARDO MARQUES



CONSIDERAÇÕES FINAIS

Um Caminho Claro no Universo Python

Ao longo deste Guia de Sobrevivência ao Python, exploramos os conceitos fundamentais da linguagem Python de maneira prática e objetiva. Nosso objetivo foi oferecer não apenas respostas rápidas para problemas cotidianos, mas também as ferramentas e os conhecimentos necessários para que você se torne um programador confiante e autônomo.

Principais Características do Material:

- **Acessibilidade:** Linguagem clara e exemplos práticos para todos os níveis de conhecimento.
- **Flexibilidade:** Aplicável tanto para estudo individual quanto para uso em sala de aula.
- **Praticidade:** Cheatsheets, fluxogramas e guias rápidos para consultas imediatas.
- **Amparo pedagógico:** Reforço de boas práticas e estratégias de aprendizado técnico.

Seções Destaques:

- **Primeiros Passos:** Configurando seu ambiente Python com confiança.

CONSIDERAÇÕES FINAIS

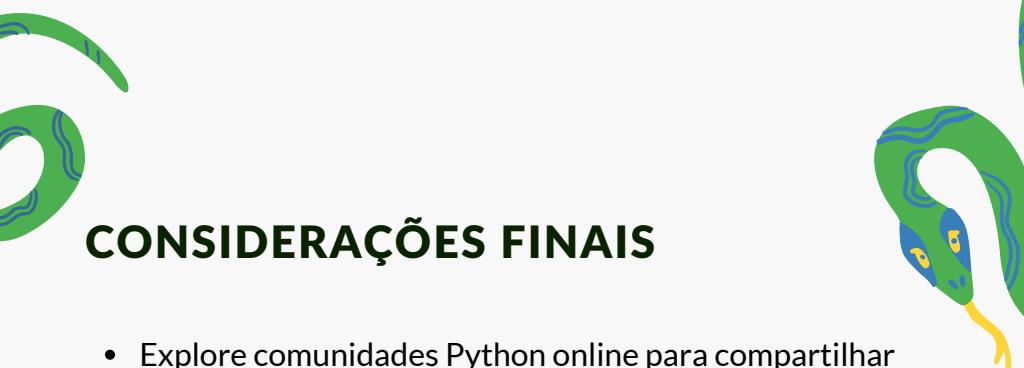
- **Estruturas Básicas:** Domine listas, dicionários e os operadores essenciais.
- **Depuração:** Aprenda a resolver problemas e interpretar mensagens de erro.
- **Fluxogramas e SOS Python:** Soluções visuais e práticas para lógica e erros comuns.

Reflexão Final:

A jornada no aprendizado de Python é contínua. Agora que você dominou os fundamentos, está preparado para enfrentar desafios maiores e mais complexos. Lembre-se: cada erro é uma oportunidade de aprendizado, e cada solução encontrada é um passo em direção à excelência na programação.

Próximos Passos:

- Aprofunde-se em bibliotecas populares como pandas para ciência de dados ou flask para desenvolvimento web.
- Participe de projetos reais para aplicar e consolidar seu conhecimento.



CONSIDERAÇÕES FINAIS

- Explore comunidades Python online para compartilhar dúvidas e soluções.

Mensagem de Encerramento:

Obrigado por confiar no Guia de Sobrevivência ao Python. Continue explorando, aprendendo e criando soluções incríveis. O Python é um universo, e você acaba de dar os primeiros passos!

REFERÊNCIAS

BRASIL. Ministério da Educação. **Base Nacional Comum Curricular (BNCC).** Brasília: MEC, 2018. Disponível em: <https://www.gov.br/mec/pt-br>. Acesso em: 10 dez. 2024.

CORACINI, Maria José; CAVALLARI, Juliana Santana (Orgs.). (Des)construindo Verdade(s) no/pelo Material Didático: Discurso, Identidade, Ensino. Campinas: Pontes Editores, 2016. Disponível em: https://ponteseditores.com.br/loja3/pontes-editores-home-2_trashed/ensino-e-a-prendizagem/produto-384/. Acesso em: 12 dez. 2024.

PYTHON SOFTWARE FOUNDATION. **The Python Language Reference.** Disponível em: <https://docs.python.org/>. Acesso em: 10 dez. 2024.

DA SILVA, Andrey H.; LOUREIRO, Álvaro A.; ARAÚJO, Pedro R. M. **Relating Voluntary Turnover with Job Characteristics, Satisfaction and Work Exhaustion - An Initial Study with Brazilian Developers.** Disponível em: <https://arxiv.org/abs/1901.11499>. Acesso em: 12 dez. 2024.

REFERÊNCIAS

DECISION REPORT. Síndrome de Burnout em profissionais de TI. 2023. Disponível em: <https://decisionreport.com.br/663-dos-profissionais-brasileiros-de-ti-relataram-estresse-no-trabalho/>. Acesso em: 12 dez. 2024.

DGRH UNICAMP. Frustração no trabalho: como combater?. Divisão de Recursos Humanos - Unicamp, 2022. Disponível em: <https://www.dgrh.unicamp.br/materia/frustracao-no-trabalho-como-combater/>. Acesso em: 12 dez. 2024.

FREITAS, Olga Cristina Rocha de. Equipamentos e Materiais Didáticos. Brasília: Ministério da Educação, 2007. Disponível em: http://portal.mec.gov.br/seb/arquivos/pdf/profunc/equip_mat_dit.pdf. Acesso em: 12 dez. 2024.

STACK OVERFLOW. Perguntas e Respostas sobre Python. Disponível em: <https://stackoverflow.com/>. Acesso em: 10 dez. 2024.

W3SCHOOLS. Python Tutorial. Disponível em: <https://www.w3schools.com/python/>. Acesso em: 10 dez. 2024.

VERSÃO PROFESSOR

A versão comentada para professores oferece margens ampliadas com orientações pedagógicas, dicas práticas e exemplos complementares. É uma ferramenta que facilita a explicação do conteúdo e o atendimento às dúvidas dos alunos. Perfeita para maximizar o aprendizado em sala de aula.

A versão comentada para professores foi cuidadosamente projetada para oferecer suporte didático em sala de aula. Este material inclui:

- Margens Comentadas:
- Cada página contém observações pedagógicas e técnicas, orientadas para facilitar a explicação do conteúdo.
- Sugestões de abordagem para tornar os conceitos mais acessíveis aos alunos.

PYTHON PODE SER SIMPLES, PRÁTICO E ATÉ DIVERTIDO!

ESTE GUIA FOI CRIADO PARA SER O SEU COMPANHEIRO EM CADA ETAPA DO APRENDIZADO DE PYTHON. ELE OFERECE DICAS RÁPIDAS, EXEMPLOS PRÁTICOS E SOLUÇÕES PARA OS PROBLEMAS MAIS COMUNS QUE SURGEM NO CAMINHO.

NÃO IMPORTA SE VOCÊ ESTÁ COMEÇANDO AGORA OU JÁ TEM ALGUMA EXPERIÊNCIA, O GUIA DE SOBREVIVÊNCIA AO PYTHON ESTÁ AQUI PARA AJUDAR VOCÊ A SUPERAR DESAFIOS, DOMINAR OS FUNDAMENTOS E AVANÇAR NA PROGRAMAÇÃO.

PORQUE APRENDER PYTHON NÃO PRECISA SER COMPLICADO.

**EDUARDO MARQUES
VERSÃO 1.0 - 2024.**