

PROBLEMA: Faça a soma de dois números digitados pelo usuário.

a

PROBLEMA GERAL

a soma de dois
números
digitados pelo
usuário.

peça dois
números ao
usuário.

PASSO 01

some esses
números.

PASSO 02

mostre o
resultado.

PASSO 03

teste e
melhore o
código.

PASSO 04



```
numero1 = float(input())
numero2 = float(input())

soma = numero1 + numero2
print(soma)
```

CRIAR O CÓDIGO

Crie um arquivo e adicione o primeiro esboço do código.

SAÍDA

2
5
7.0



```
def somar(a, b):  
    return a + b  
  
numero1 = float(input("Digite o primeiro  
número: "))  
numero2 = float(input("Digite o segundo  
número: "))  
  
resultado = somar(numero1, numero2)  
print(f"A soma é: {resultado}")
```

SAÍDA

MELHORANDO O CÓDIGO

Organize o código em funções pequenas em que cada uma tem uma única responsabilidade.

Digite o primeiro número: 2
Digite o segundo número: 5
A soma é: 7.0



```
def somar(a, b):  
    return a + b  
  
try:  
    numero1 = float(input("Digite o  
primeiro número: "))  
    numero2 = float(input("Digite o  
segundo número: "))  
  
    resultado = somar(numero1, numero2)  
    print(f"A soma é: {resultado}")  
except ValueError:  
    print("Erro: Digite apenas números  
válidos!")
```

SAÍDA

TRATAR POSSÍVEIS ERROS

E se o usuário digitar algo que não seja um número? O código quebraria!

Digite o primeiro número: abc
Erro: Digite apenas números válidos!



```
def somar(a, b):  
    return a + b  
  
try:  
    numero1 = float(input("Digite o  
primeiro número: "))  
    numero2 = float(input("Digite o  
segundo número: "))  
  
    resultado = somar(numero1, numero2)  
    print(f"A soma é: {resultado}")  
except ValueError:  
    print("Erro: Digite apenas números  
válidos!")
```

SAÍDA

TESTANDO MANUALMENTE

Verifique como o código reage a diferentes cenários!

Digite o primeiro número: 5

Digite o segundo número:

Erro: Digite apenas números válidos!

Digite o primeiro número: 2

Digite o segundo número: 5

A soma é: 7.0

Erro	Mensagem	Motivo	Solução
Entrada Inválida	ValueError	Ocorre quando tentamos converter um valor que não pode ser transformado no tipo esperado. Ex: <code>float("abc")</code> .	Tratar erro com bloco <code>try/except</code> .
Tipo Inválido	TypeError	Ocorre quando realizamos operações incompatíveis entre tipos de dados. Ex: <code>10 + "5"</code> .	Converter valores antes de operações. Ex: <code>int("5")</code> .
Chave inexistente	KeyError	Ocorre ao tentar acessar uma chave que não existe em um dicionário.	Usar <code>.get("chave", valor_padrao)</code> para evitar erro.
Índice fora da lista	IndexError	Ocorre ao tentar acessar um índice que não existe em uma lista. Ex: acessar <code>lista[5]</code> em uma lista de 3 itens).	Conferir <code>len(lista)</code> antes de acessar um índice.

POSSÍVEIS ERROS EM PROJETOS

Erro	Mensagem	Motivo	Solução
Erro de importação	<code>ImportError</code>	Ocorre quando um módulo é encontrado, mas não pode ser carregado corretamente.	Verificar se o módulo está instalado e disponível.
Módulo não encontrado	<code>ModuleNotFoundError</code>	Ocorre quando tentamos importar um módulo que não está instalado ou não existe.	Garantir que o módulo está instalado com <code>`pip install modulo`</code> .
Atributo inexistente	<code>AttributeError</code>	Ocorre quando tentamos acessar um atributo ou método que não existe em um objeto. Ex: "abc".append(5).	Converter valores antes de operações. Ex: <code>int("5")</code> .
Sintaxe inválida	<code>SyntaxError</code>	Ocorre quando há um erro na escrita do código, como esquecer um : ou parênteses.	Revisar a sintaxe e corrigir erros antes de rodar o código.

POSSÍVEIS ERROS EM PROJETOS



Para conhecer mais sobre erros específicos,
consulte a [documentação oficial](#) do Python.





Python Enhancement Proposals | Python > PEP Index > PEP 8

Contents

- Introduction
- A Foolish Consistency is the Hobgoblin of Little Minds
- Code Lay-out
 - Indentation
 - Tabs or Spaces?
 - Maximum Line Length
 - Should a Line Break Before or After a Binary Operator?
 - Blank Lines
 - Source File Encoding
 - Imports
 - Module Level Dunder Names
 - String Quotes
 - Whitespace in Expressions and Statements
 - Pet Peeves
 - Other Recommendations
 - When to Use Trailing Commas
 - Comments
 - Block Comments
 - Inline Comments
 - Documentation Strings
 - Naming Conventions
 - Overriding Principle
 - Descriptive Naming Style

a

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>

Status: Active

Type: Process

Created: 05-Jul-2001

Post-History: 05-Jul-2001, 01-Aug-2013

▶ Table of Contents

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing [style guidelines for the C code in the C implementation of Python](#).

This document and [PEP 257](#) (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered

BOAS PRÁTICAS

CONHECENDO A PEP 8

É um guia de estilo para escrever código Python de forma legível, organizada e padronizada. Seguir essas regras facilita a manutenção, colaboração e qualidade do código.

Acessar!



Organização e estrutura do código



Nomeação de variáveis, funções e classes



Comentários e documentação clara e objetiva



Para se aprofundar em boas práticas

FORMAÇÕES > PROGRAMAÇÃO > BOAS PRÁTICAS EM PYTHON

Formação Boas práticas em Python

Aprenda os fundamentos das boas práticas de programação com Python

* Esta formação faz parte dos nossos [cursos de Programação](#)



Acessar!



PASSO 1

Organizar a execução
e passo a passo do
projeto.

Definir o problema.

PASSO 2

PASSO 3

Escrever o código.

PASSO 5

Estruturar o projeto.

PASSO 4

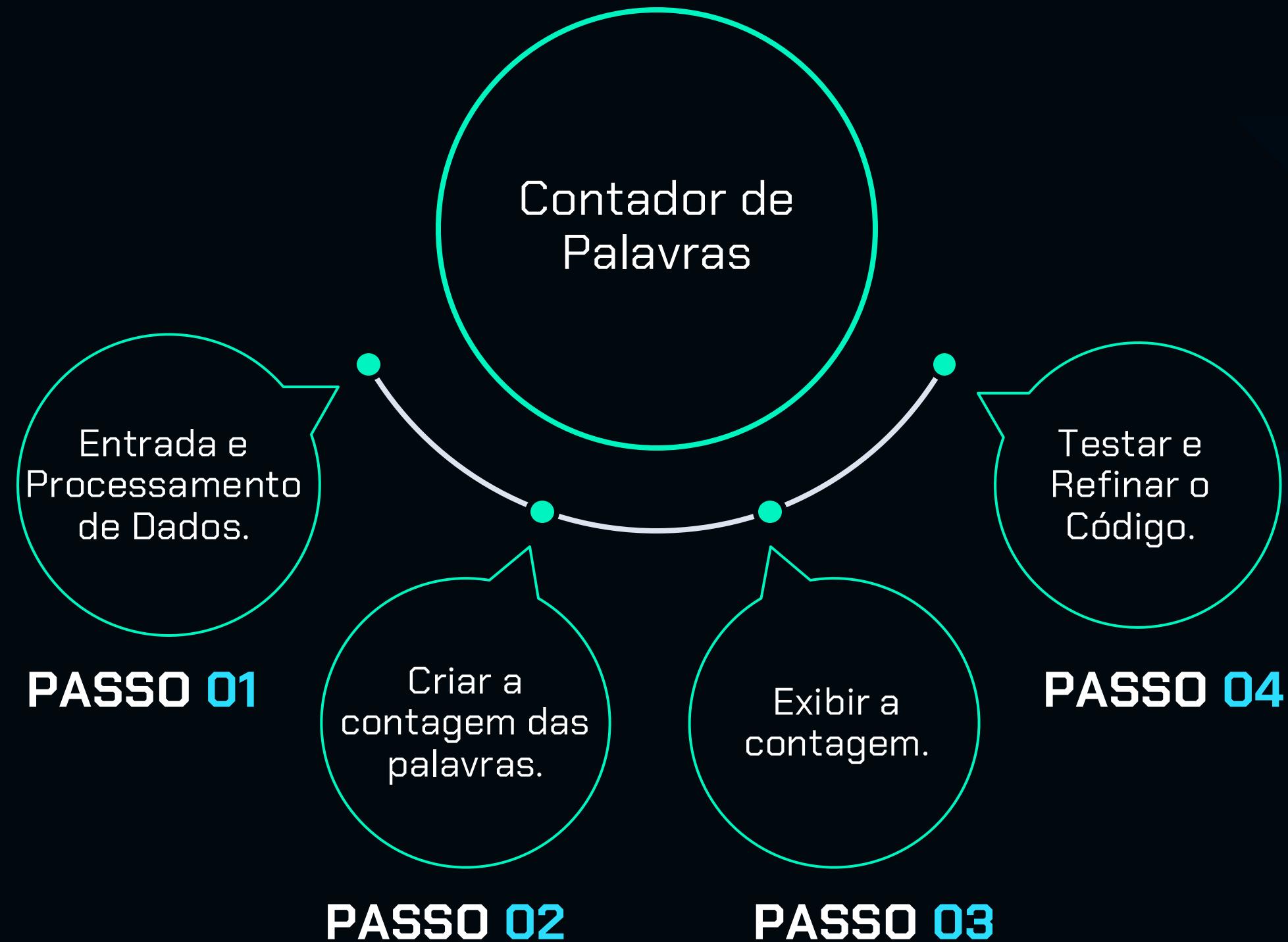
**Testar e refinar o
código.**

VÍDEO 02

a

PROBLEMA: Faça um **contador de palavras**. O projeto deve receber uma frase do usuário e contar quantas vezes cada palavra aparece.

PROBLEMA GERAL





contador.py

```
frase = input("Digite uma frase: ")  
palavras = frase.split()  
print(len(palavras))  
print(palavras)
```

SAÍDA

CÓDIGO INICIAL

- ✓ Crie um arquivo `contador.py` e desenvolva o código do contador.

Digite uma frase: olá mundo mundo!!
3
['olá', 'mundo', 'mundo!!']



contador.py

```
def contar_palavras(frase):  
    palavras = frase.split()  
    print(palavras)  
    return len(palavras)
```

APRIMORANDO O CÓDIGO

- ✓ crie uma função para o contador.



main.py

```
from contador import contar_palavras

frase = input("Digite uma frase: ")
quantidade = contar_palavras(frase)
print(f"A frase tem {quantidade} palavras.")
```

SAÍDA

APRIMORANDO O CÓDIGO

- ✓ crie um arquivo `main.py` para executar os testes no mesmo diretório que `contador.py`.

Digite uma frase: livros são livros.
['livros', 'são', 'livros.']

A frase tem 3 palavras.

Tipo de Entrada	Exemplo	Problema
Entrada Vazia	Usuário pressiona Enter sem digitar nada ("").	O código pode falhar ou retornar valores incorretos.
Apenas espaços	(" ")	Deve ser tratado como vazio.
Apenas pontuação	(",.!?;:")	Não contém palavras válidas.
Texto com pontuação	("Olá, mundo!")	A pontuação interfere na contagem correta.
Caracteres especiais e acentos	("Python é incrível!")	Pode gerar palavras separadas caso haja tratamento incorreto.
Vários espaços entre palavras	("Essa frase é curta !")	O <code>split()</code> pode gerar listas incorretas.

Possíveis entradas do usuário

Tipo de Entrada	Exemplo	Problema
Palavras repetidas	("Python python é bom").	Deve contar corretamente a frequência.
Maiúsculas e minúsculas	("Python PYTHON python").	O código deve tratar palavras como iguais.
Números na frase	("Faço 26 anos em 2 meses.").	Dependendo do contexto, pode ser palavra válida ou não.
Somente números	("123 456 789")	Números podem ser tratados como palavras.
Mistura de letras e números	("Python3 é melhor que Python2?")	A pontuação pode separar indevidamente.

Possíveis entradas do usuário



contador.py

```
def limpar_texto(texto):  
    texto = texto.lower()  
    caracteres = ",.!|?;:\\'()[]{}"  
    for char in caracteres:  
        texto = texto.replace(char, "")  
    return texto  
  
def contar_palavras(frase):  
    frase = limpar_texto(frase)  
    palavras = frase.split()  
    return len(palavras)
```

APRIMORANDO O CÓDIGO

- ✓ faça o tratamento da frase para remover pontuação;
- ✓ converta as palavras em minúsculas.



contador.py

```
> def limpar_texto(texto): ...  
  
def contar_palavras(frase):  
    frase = limpar_texto(frase)  
    if not frase.strip():  
        return {}  
    palavras = frase.split()  
    contagem = {}  
    for palavra in palavras:  
        contagem[palavra] =  
contagem.get(palavra, 0) + 1  
    return contagem
```

APRIMORANDO O CÓDIGO

- ✓ use um dicionário para armazenar a frequência das palavras e evitar entradas inválidas.



main.py

```
from contador import contar_palavras

frase = input("Digite uma frase: ").strip()
if not frase:
    print("Erro: Nenhuma frase foi digitada.")
else:
    resultado = contar_palavras(frase)
    if resultado:
        print("Contagem de Palavras:")
        for palavra, quantidade in
resultado.items():
            print(f"{palavra}: {quantidade}")
    else:
        print(f"Nenhuma palavra válida foi
encontrada.")
```

APRIMORANDO O CÓDIGO

- ✓ verifique se a entrada está vazia;
- ✓ apresente a contagem de palavras de forma clara.



EXEMPLOS DE RESPOSTA:

SAÍDA

Digite uma frase: OLÁ, mUndo!

Contagem de palavras:

olá: 1

mundo: 1

SAÍDA

SAÍDA

Digite uma frase: python[] é incrível incrível.

Contagem de palavras:

python: 1

é: 1

incrível: 2

Digite uma frase: Python3 é melhor que Python2 em 2025?
Contagem de palavras:
python3: 1
é: 1
melhor: 1
que: 1
python2: 1
em: 1
2025: 1

Compartilhe um resumo de seus novos
conhecimentos em suas redes sociais.

#aprendizadoalura