Introdução ao Python

A linguagem **Python** foi criada no final da década de 1980 e lançada oficialmente em **1991** por **Guido van Rossum**, um programador holandês. Ele trabalhava no CWI (Centrum Wiskunde & Informatica) na Holanda e queria desenvolver uma linguagem de programação mais acessível, fácil de ler e escrever, mas ainda poderosa o suficiente para resolver problemas reais.

Origem e Motivação

Guido van Rossum era um fã da linguagem **ABC**, que era fácil de aprender e tinha algumas ideias interessantes, como manipulação de strings e listas. No entanto, a ABC tinha algumas limitações, como a falta de extensibilidade. Ele decidiu então criar uma nova linguagem que resolvesse esses problemas e fosse mais prática para programadores profissionais.

O projeto começou no **Natal de 1989**, quando Guido queria um projeto divertido para ocupar seu tempo durante as férias. Ele se inspirou em C, Modula-3 e outras linguagens, mas manteve um foco na legibilidade e simplicidade.

Por que o nome "Python"?

O nome **Python** não tem nada a ver com cobras! 🐍 Na verdade, Guido van Rossum era fã do grupo de comédia britânico **Monty Python's Flying Circus**. Ele queria um nome curto, único e um pouco misterioso, então escolheu "Python".

Primeiros Anos (1991 - 2000)

- **1991**: Guido lançou a **versão 0.9.0** do Python no grupo de discussão Usenet alt.sources.
  - Já incluía **classes, exceções, módulos e tipos de dados como listas e dicionários**.
- **1994**: Criada a **Python Software Foundation (PSF)**, que passou a cuidar do desenvolvimento da linguagem.
- **2000**: Lançamento do **Python 2.0**, trazendo melhorias como **coleta de lixo automática** e **listas por compreensão (list comprehensions)**.

  ⚠ **Problema do Python 2**: Ele não era totalmente compatível com o Python 3, o que gerou dificuldades para os desenvolvedores por muitos anos.

Evolução com Python 3 (2008 - Presente)

- **2008**: Lançamento do **Python 3.0**, uma reformulação para tornar a linguagem mais consistente e moderna.
  - Melhorias na manipulação de strings (Unicode por padrão).
  - Alterações na sintaxe (exemplo: print virou função → print("Olá, mundo!")).
- **2020**: O suporte ao **Python 2 foi oficialmente encerrado**.
- **Atualmente**: Python é uma das linguagens mais populares do mundo, sendo amplamente usado em:
  - **Ciência de Dados e Machine Learning** (com pandas, NumPy, scikit-learn, TensorFlow).
  - **Desenvolvimento Web** (com frameworks como Django e Flask).
  - **Automação e Scripts**.
  - **Cibersegurança e Hacking Ético**.
  - **Jogos e Aplicações Desktop**.

Python Hoje e no Futuro

- Python se tornou a **linguagem mais popular** segundo rankings como **TIOBE e Stack Overflow Developer Survey**.
- Grandes empresas como **Google, Facebook, NASA e Netflix** usam Python intensivamente.
- A tendência é que o Python continue crescendo, especialmente em **inteligência artificial, automação e desenvolvimento web**.

Python surgiu como um projeto pessoal de **Guido van Rossum**, mas hoje é uma das linguagens mais usadas no mundo devido à sua simplicidade e versatilidade.

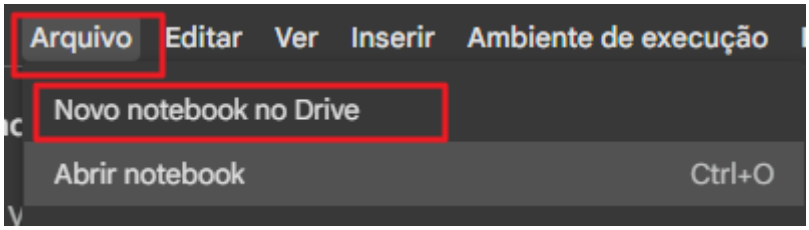
Python é uma linguagem de programação de alto nível, fácil de aprender e amplamente utilizada em diversos campos, como desenvolvimento web, análise de dados, inteligência artificial, automação, entre outros.

Sua sintaxe simples e a vasta biblioteca padrão tornam-na uma excelente escolha tanto para iniciantes quanto para programadores experientes.

A seguir, abordaremos uma série de exemplos didáticos para que você compreenda os conceitos fundamentais da linguagem Python e aprenda a programar de maneira prática e eficiente.

Vamos praticar

Acesse - https://colab.research.google.com/#scrollTo=Wf5KrEb6vrkR

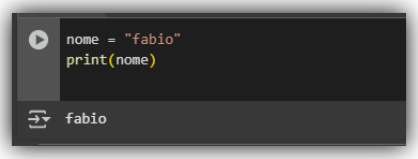


Passo a Passo

1. Imprimindo uma mensagem simples

Python

```
nome = "fabio"
print(nome)
```



Este código mostra como usar a função print() para exibir informações no console. O valor da variável nome será impresso.

---

2. Operações matemáticas básicas

Python

```
a = 5
b = 3
print(a + b)  # Soma
print(a - b)  # Subtração
print(a * b)  # Multiplicação
print(a / b)  # Divisão
```

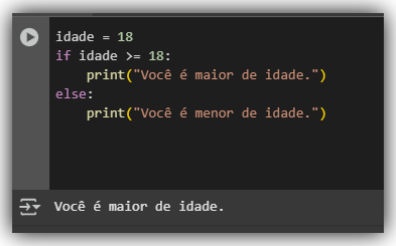
Aqui, você aprende a realizar operações matemáticas básicas: soma, subtração, multiplicação e divisão.

---

3. Usando estruturas condicionais (if-else)

Python

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



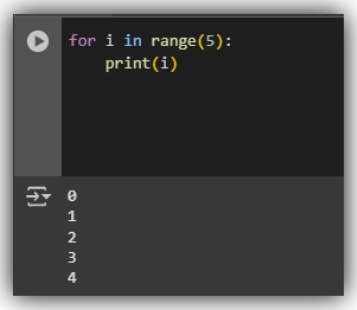
As estruturas condicionais (if-else) permitem que você execute diferentes blocos de código dependendo de uma condição.

---

4. Loops (for loop)

Python

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



O for loop é usado para iterar sobre uma sequência (como uma lista, tupla ou intervalo de números).

---

5. Loops (while loop)

Python

```
contador = 0
while contador < 5:
    print(contador)
    contador += 1
```

O while loop executa o código enquanto uma condição for verdadeira. No exemplo, ele continua até que o contador chegue a 5.

---

6. Definindo funções

Python

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

saudacao("Fabio")
```

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

saudacao("Fabio")
```

```
Olá, Fabio!
```

Funções em Python permitem que você agrupe código reutilizável.
A função saudacao recebe um parâmetro e exibe uma mensagem personalizada.

---

7. Listas

Python

```
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)
```

```python
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)
```

```
maçã
banana
laranja
```

Listas são coleções ordenadas e mutáveis.
Você pode usar um loop para acessar seus elementos.

---

8. Dicionários

Python

```
pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"])
```

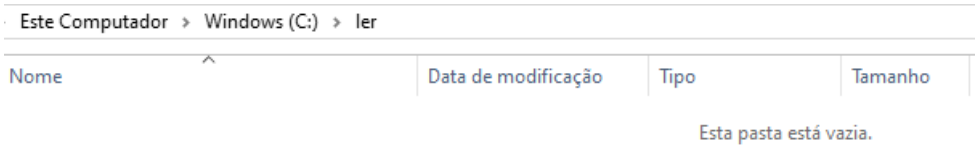
```python
pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"])
```

```
João
```

Dicionários são coleções não ordenadas de pares chave-valor.

Eles são úteis quando você precisa associar uma chave a um valor.

9. Manipulação de arquivos



Python

```
import os
# Obtendo o caminho para a pasta Downloads

downloads_path = os.path.join(os.path.expanduser("~"), "Downloads", "arquivo.txt")

# Abrindo e escrevendo no arquivo
with open(downloads_path, "w") as arquivo:arquivo.write("Olá, arquivo!")
```

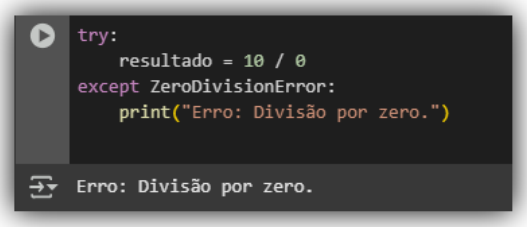
A manipulação de arquivos permite que você leia e escreva em arquivos de texto. No exemplo, estamos criando e escrevendo em um arquivo chamado arquivo.txt.

---

10. Manipulando exceções

Python

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



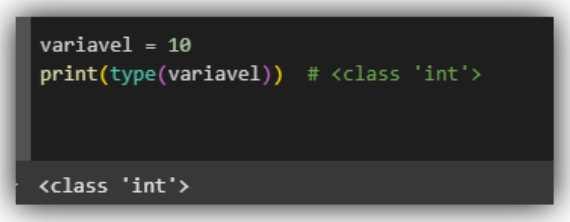
O tratamento de exceções ajuda a lidar com erros durante a execução do código, evitando que o programa quebre

---

11. Verificando tipo de variável

Python

```
variavel = 10
print(type(variavel))  # <class 'int'>
```



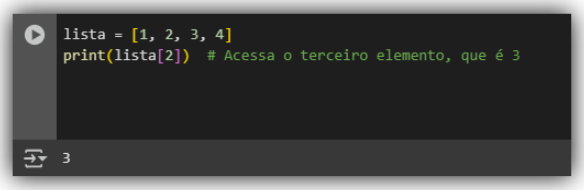
A função type() retorna o tipo de dado de uma variável. No caso, int significa inteiro.

---

12. Acessando elementos de uma lista

Python

```
lista = [1, 2, 3, 4]
print(lista[2])  # Acessa o terceiro elemento, que é 3
```



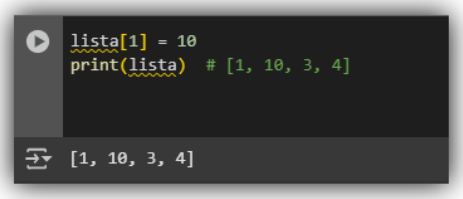
As listas em Python são indexadas, e você pode acessar seus elementos usando um índice

---

13. Modificando elementos de uma lista

Python

```
lista[1] = 10
print(lista)  # [1, 10, 3, 4]
```



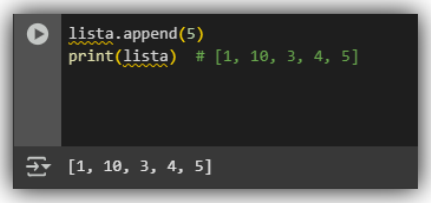
Você pode modificar os elementos de uma lista diretamente, usando o índice do elemento.

---

14. Adicionando elementos a uma lista

Python

```
lista.append(5)
print(lista)  # [1, 10, 3, 4, 5]
```



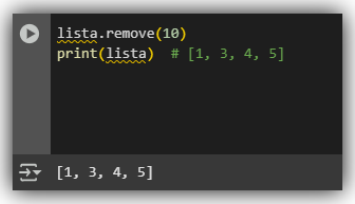
O método append() adiciona um novo elemento ao final de uma lista.

---

15. Removendo elementos de uma lista

Python

```
lista.remove(10)
print(lista)  # [1, 3, 4, 5]
```



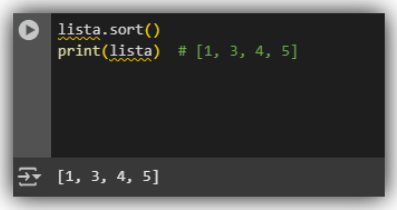
O método remove() remove a primeira ocorrência de um valor específico em uma lista.

---

16. Ordenando uma lista

Python

```
lista.sort()
print(lista)  # [1, 3, 4, 5]
```



O método sort() ordena os elementos de uma lista em ordem crescente.

---

17. Desempacotamento de listas

Python

```
a, b, c = [1, 2, 3]
```

print(a, b, c) # 1 2 3

```
a, b, c = [1, 2, 3]
print(a, b, c)  # 1 2 3
```

```
1 2 3
```

O desempacotamento de listas permite atribuir os elementos de uma lista a variáveis.

---

18. Concatenando listas

Python

lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa) # [1, 2, 3, 4, 5, 6]

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)  # [1, 2, 3, 4, 5, 6]
```

```
[1, 2, 3, 4, 5, 6]
```

A concatenação de listas é feita usando o operador +.

---

19. Tuplas

Python

tupla = (1, 2, 3)
print(tupla[0]) # 1

```
tupla = (1, 2, 3)
print(tupla[0])  # 1
```

```
1
```

As tuplas são semelhantes às listas, mas são imutáveis. Ou seja, você não pode modificar seus elementos depois de criá-los.

---

20. Dicionários com listas

Python

```
pessoa = {
    "nome": "Carlos",
    "idade": 25,
    "enderecos": ["Rua A", "Rua B"]
}
print(pessoa["enderecos"])
```

Os dicionários podem conter listas como valores. Você pode acessar as listas de maneira similar.

Neste passo a passo, exploramos os conceitos fundamentais de Python, incluindo operações matemáticas, estruturas condicionais, loops, funções, listas, dicionários e manipulação de arquivos.

Com esses conceitos, você já pode começar a escrever scripts mais complexos e resolver problemas de programação de forma mais eficiente.

Estes exemplos podem ser usados como base para entender e praticar a linguagem Python.

Experimente modificar os exemplos e criar suas próprias variações para se aprofundar no aprendizado!

Codigos para Praticar

```
nome = "fabio";
print(nome);

# Exemplo 1: Imprimir uma mensagem simples
print("Olá, mundo!")

# Exemplo 2: Operações matemáticas básicas
a = 5
b = 3
print(a + b)  # Soma
print(a - b)  # Subtração
print(a * b)  # Multiplicação
print(a / b)  # Divisão

# Exemplo 3: Usando estruturas condicionais (if-else)
idade = 18
if idade >= 18:
    print("Você é maior de idade.")
else:
    print("Você é menor de idade.")

# Exemplo 4: Loops (for loop)
for i in range(5):
    print(i)

# Exemplo 5: Loops (while loop)
contador = 0
while contador < 5:
    print(contador)
    contador += 1

# Exemplo 6: Definindo funções
def saudacao(nome):
    print(f"Olá, {nome}!")

saudacao("Maria")

# Exemplo 7: Listas
frutas = ["maçã", "banana", "laranja"]
for fruta in frutas:
    print(fruta)

# Exemplo 8: Dicionários
pessoa = {"nome": "João", "idade": 30, "cidade": "São Paulo"}
print(pessoa["nome"])


# Exemplo 9: Manipulação de arquivos
import os
# Obtendo o caminho para a pasta Downloads
downloads_path = os.path.join(os.path.expanduser("~"), "Downloads", "arquivo.txt")
# Abrindo e escrevendo no arquivo
with open(downloads_path, "w") as arquivo:arquivo.write("Olá, arquivo!")
```

```
# Exemplo 10: Manipulando exceções
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: Divisão por zero.")


# Exemplo 11: Verificando tipo de variável
variavel = 10
print(type(variavel))  # <class 'int'>

# Exemplo 12: Acessando elementos de uma lista
lista = [1, 2, 3, 4]
print(lista[2])  # Acessa o terceiro elemento, que é 3

# Exemplo 13: Modificando elementos de uma lista
lista[1] = 10
print(lista)  # [1, 10, 3, 4]

# Exemplo 14: Adicionando elementos a uma lista
lista.append(5)
print(lista)  # [1, 10, 3, 4, 5]

# Exemplo 15: Removendo elementos de uma lista
lista.remove(10)
print(lista)  # [1, 3, 4, 5]

# Exemplo 16: Ordenando uma lista
lista.sort()
print(lista)  # [1, 3, 4, 5]

# Exemplo 17: Desempacotamento de listas
a, b, c = [1, 2, 3]
print(a, b, c)  # 1 2 3

# Exemplo 18: Concatenando listas
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
lista_completa = lista1 + lista2
print(lista_completa)  # [1, 2, 3, 4, 5, 6]

# Exemplo 19: Tuplas
tupla = (1, 2, 3)
print(tupla[0])  # 1

# Exemplo 20: Dicionários com listas
pessoa = {
    "nome": "Carlos",
    "idade": 25,
    "enderecos": ["Rua A", "Rua B"]
}
print(pessoa["enderecos"])

# Exemplo 21: Definindo funções com parâmetros
def soma(a, b):
    return a + b
```

```
resultado = soma(5, 10)
print(resultado)  # 15

# Exemplo 22: Função com valor de retorno
def quadrado(x):
    return x ** 2

print(quadrado(4))  # 16

# Exemplo 23: Funções com argumentos padrões
def saudacao(nome="Visitante"):
    print(f"Olá, {nome}!")

saudacao("Ana")
saudacao()  # Olá, Visitante!

# Exemplo 24: Função recursiva
def fatorial(n):
    if n == 1:
        return 1
    else:
        return n * fatorial(n - 1)

print(fatorial(5))  # 120

# Exemplo 25: Lambdas (funções anônimas)
soma = lambda a, b: a + b
print(soma(3, 4))  # 7

# Exemplo 26: List comprehension
quadrados = [x ** 2 for x in range(5)]
print(quadrados)  # [0, 1, 4, 9, 16]

# Exemplo 27: Funções de map
numeros = [1, 2, 3, 4]
quadrados = list(map(lambda x: x ** 2, numeros))
print(quadrados)  # [1, 4, 9, 16]

# Exemplo 28: Função de filtro
pares = list(filter(lambda x: x % 2 == 0, numeros))
print(pares)  # [2, 4]

# Exemplo 29: Função reduce
from functools import reduce
soma_total = reduce(lambda x, y: x + y, numeros)
print(soma_total)  # 10

# Exemplo 30: Compreensão de dicionários
dicionario = {x: x ** 2 for x in range(5)}
print(dicionario)  # {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# Exemplo 31: Acessando caracteres de uma string
texto = "Python"
print(texto[0])  # 'P'
```

```python
# Exemplo 32: Fatiamento de strings
print(texto[1:4])  # 'yth'

# Exemplo 33: Concatenando strings
nome = "Ana"
sobrenome = "Silva"
nome_completo = nome + " " + sobrenome
print(nome_completo)  # 'Ana Silva'

# Exemplo 34: Métodos de string
frase = "olá mundo"
print(frase.upper())  # 'OLÁ MUNDO'

# Exemplo 35: Substituindo palavras em uma string
texto = "Eu gosto de Python"
novo_texto = texto.replace("Python", "Java")
print(novo_texto)  # 'Eu gosto de Java'

# Exemplo 36: Dividindo uma string
texto = "Python é ótimo"
palavras = texto.split()
print(palavras)  # ['Python', 'é', 'ótimo']

# Exemplo 37: Verificando se uma substring está presente
print("Python" in texto)  # True

# Exemplo 38: Função join para concatenar lista de strings
palavras = ["Python", "é", "legal"]
frase = " ".join(palavras)
print(frase)  # 'Python é legal'

# Exemplo 39: Usando o operador "in" com listas
numeros = [1, 2, 3, 4, 5]
print(3 in numeros)  # True

# Exemplo 40: Manipulando conjuntos
conjunto = {1, 2, 3}
conjunto.add(4)
print(conjunto)  # {1, 2, 3, 4}

# Exemplo 41: Diferença entre listas e conjuntos
lista = [1, 2, 3, 3, 4]
conjunto = set(lista)
print(conjunto)  # {1, 2, 3, 4} - Conjunto remove duplicatas

# Exemplo 42: Criando e manipulando arquivos
with open("meuarquivo.txt", "w") as arquivo:
    arquivo.write("Python é uma linguagem poderosa!")

with open("meuarquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)  # 'Python é uma linguagem poderosa!'

# Exemplo 43: Manipulando diretórios
import os
os.mkdir("meu_diretorio")
print("Diretório criado!")
```

```
# Exemplo 44: Verificando se um arquivo ou diretório existe
existe = os.path.exists("meuarquivo.txt")
print(existe)  # True

# Exemplo 45: Listando arquivos em um diretório
arquivos = os.listdir(".")
print(arquivos)  # Listagem de arquivos no diretório atual

# Exemplo 46: Utilizando a biblioteca math
import math
print(math.sqrt(16))  # 4.0

# Exemplo 47: Gerando números aleatórios
import random
print(random.randint(1, 10))  # Número aleatório entre 1 e 10

# Exemplo 48: Trabalhando com data e hora
import datetime
hoje = datetime.datetime.now()
print(hoje)  # Exibe a data e hora atual

# Exemplo 49: Tratamento de erros com try-except
try:
    numero = int(input("Digite um número: "))
except ValueError:
    print("Erro: Entrada inválida!")

# Exemplo 50: Função de ajuda (help)
help(print)  # Exibe a documentação da função print
```

Explicando

1. Imprimir uma mensagem simples

A função print() é usada para exibir mensagens ou valores na tela. Este exemplo mostra como imprimir uma simples saudação.

---

2. Operações matemáticas básicas

Neste exemplo, mostramos como realizar operações matemáticas como soma, subtração, multiplicação e divisão entre dois números. O resultado de cada operação é impresso na tela.

---

3. Usando estruturas condicionais (if-else)

As estruturas condicionais (if-else) permitem executar diferentes blocos de código com base em uma condição. Este exemplo verifica se uma pessoa é maior ou menor de idade.

---

4. Loops (for loop)

O loop for é utilizado para repetir uma ação um número específico de vezes ou para iterar sobre uma sequência de valores. Neste caso, o loop imprime números de 0 a 4.

---

5. Loops (while loop)

O loop while repete um bloco de código enquanto uma condição for verdadeira. Este exemplo utiliza um contador para imprimir números de 0 a 4.

---

6. Definindo funções

Uma função é um bloco de código que pode ser reutilizado. Este exemplo define uma função saudacao() que recebe um nome como parâmetro e imprime uma saudação personalizada.

---

7. Listas

Listas são coleções de elementos ordenados. Este exemplo percorre uma lista de frutas e imprime cada uma delas.

---

8. Dicionários

Dicionários são coleções de pares chave-valor. Aqui, é mostrado como acessar um valor dentro de um dicionário, usando uma chave específica.

---

9. Manipulação de arquivos

Este exemplo mostra como criar e escrever em um arquivo de texto dentro da pasta **Downloads** do usuário. Ele usa o método open() para abrir ou criar um arquivo e escrever nele.

---

10. Manipulando exceções

O tratamento de exceções permite capturar erros durante a execução do código, evitando que o programa pare de funcionar. Este exemplo trata um erro de divisão por zero.

---

11. Verificando tipo de variável

O Python permite verificar o tipo de uma variável com a função type(). Este exemplo imprime o tipo de uma variável inteira.

---

12. Acessando elementos de uma lista

As listas são indexadas, o que significa que você pode acessar seus elementos usando índices. Este exemplo mostra como acessar o terceiro elemento de uma lista.

---

13. Modificando elementos de uma lista

Você pode alterar os valores dentro de uma lista acessando um índice específico. Este exemplo modifica o segundo item de uma lista.

---

14. Adicionando elementos a uma lista

Você pode adicionar elementos ao final de uma lista usando o método append(). Aqui, é mostrado como adicionar um número ao final da lista.

---

15. Removendo elementos de uma lista

A função remove() permite remover o primeiro elemento que corresponde ao valor fornecido. Este exemplo remove um item específico de uma lista.

---

16. Ordenando uma lista

Listas podem ser ordenadas em ordem crescente com o método sort(). Este exemplo mostra como ordenar os números de uma lista.

17. Desempacotamento de listas

O desempacotamento de listas é uma maneira de atribuir múltiplos valores de uma lista a variáveis separadas. Este exemplo mostra como atribuir três valores de uma lista a três variáveis.

---

18. Concatenando listas

Listas podem ser combinadas usando o operador +, o que cria uma nova lista contendo os elementos de ambas. Este exemplo concatena duas listas.

---

19. Tuplas

As tuplas são coleções imutáveis. Este exemplo mostra como acessar elementos de uma tupla. Diferente das listas, as tuplas não podem ser alteradas após a criação.

---

20. Dicionários com listas

Os dicionários podem armazenar listas como valores. Este exemplo mostra como acessar uma lista dentro de um dicionário.

---

21. Definindo funções com parâmetros

Funções podem receber parâmetros para operar de maneira dinâmica. Este exemplo define uma função que recebe dois números e retorna sua soma.

---

22. Função com valor de retorno

As funções podem retornar valores, que podem ser usados em outras partes do código. Este exemplo define uma função que retorna o quadrado de um número.

---

23. Funções com argumentos padrões

Funções podem ter valores padrão para seus parâmetros. Este exemplo mostra como a função pode ser chamada com ou sem um argumento, usando um valor padrão se o argumento não for fornecido.

---

24. Função recursiva

Funções recursivas são aquelas que chamam a si mesmas. Este exemplo define uma função que calcula o fatorial de um número, utilizando recursão.

25. Lambdas (funções anônimas)

Funções lambda são funções pequenas e anônimas que podem ser definidas em uma única linha. Este exemplo mostra uma função lambda que soma dois números.

---

26. List comprehension

A list comprehension é uma maneira compacta de criar listas. Este exemplo cria uma lista dos quadrados dos números de 0 a 4.

---

27. Funções de map

A função map() aplica uma função a todos os itens de uma sequência, retornando uma nova sequência com os resultados. Este exemplo calcula o quadrado de todos os números de uma lista.

---

28. Função de filtro

A função filter() filtra elementos de uma sequência com base em uma condição fornecida. Este exemplo seleciona os números pares de uma lista.

---

29. Função reduce

A função reduce() aplica uma função acumuladora a uma sequência, reduzindo-a a um único valor. Este exemplo soma todos os números de uma lista.

---

30. Compreensão de dicionários

Assim como a list comprehension, você também pode criar dicionários de forma concisa. Este exemplo cria um dicionário com números e seus quadrados.

---

31. Acessando caracteres de uma string

As strings são sequências de caracteres e você pode acessar um caractere específico usando índices. Este exemplo acessa o primeiro caractere de uma string.

---

32. Fatiamento de strings

O fatiamento de strings permite que você acesse um intervalo de caracteres. Este exemplo mostra como obter uma parte de uma string.

33. Concatenando strings

Strings podem ser concatenadas (unidas) com o operador +. Este exemplo concatena o nome e o sobrenome para formar um nome completo.

---

34. Métodos de string

As strings possuem muitos métodos úteis, como upper(), que converte todos os caracteres em maiúsculas. Este exemplo converte uma frase para maiúsculas.

---

35. Substituindo palavras em uma string

O método replace() permite substituir uma palavra ou parte de uma string por outra. Este exemplo substitui a palavra "Python" por "Java".

---

36. Dividindo uma string

O método split() divide uma string em uma lista de substrings, usando um separador. Este exemplo divide uma frase em palavras.

---

37. Verificando se uma substring está presente

Você pode verificar se uma substring está presente em uma string com o operador in. Este exemplo verifica se "Python" está presente em uma string.

---

38. Função join para concatenar lista de strings

O método join() permite concatenar todos os elementos de uma lista de strings, inserindo um separador entre eles. Este exemplo junta uma lista de palavras em uma frase.

---

39. Usando o operador "in" com listas

O operador in também pode ser usado com listas para verificar se um item está presente. Este exemplo verifica se o número 3 está na lista.

---

40. Manipulando conjuntos

Conjuntos são coleções de elementos únicos. O método add() permite adicionar um novo elemento a um conjunto. Este exemplo adiciona o número 4 a um conjunto.

41. Diferença entre listas e conjuntos

A principal diferença entre listas e conjuntos é que os conjuntos não permitem elementos duplicados. Este exemplo converte uma lista em um conjunto, removendo duplicatas.

---

42. Criando e manipulando arquivos

Este exemplo mostra como criar e escrever em um arquivo, e depois ler o conteúdo do arquivo. A manipulação de arquivos é útil para armazenar dados.

---

43. Manipulando diretórios

A função os.mkdir() permite criar novos diretórios. Este exemplo cria um diretório chamado meu_diretorio.

---

44. Verificando se um arquivo ou diretório existe

O método os.path.exists() verifica se um arquivo ou diretório existe no caminho especificado. Este exemplo verifica se o arquivo meuarquivo.txt existe.

---

45. Listando arquivos em um diretório

O método os.listdir() lista todos os arquivos e pastas presentes em um diretório. Este exemplo lista os arquivos no diretório atual.

---

46. Utilizando a biblioteca math

A biblioteca math oferece várias funções matemáticas, como sqrt(), que calcula a raiz quadrada. Este exemplo calcula a raiz quadrada de 16.

---

47. Gerando números aleatórios

A função random.randint() gera números inteiros aleatórios dentro de um intervalo especificado. Este exemplo gera um número aleatório entre 1 e 10.

---

48. Trabalhando com data e hora

A biblioteca datetime permite trabalhar com datas e horas. Este exemplo obtém a data e hora atual e a imprime.

49. Tratamento de erros com try-except

O bloco try-except captura e lida com erros durante a execução do código. Este exemplo trata o erro de entrada inválida ao tentar converter uma entrada para um número.

---

50. Função de ajuda (help)

A função help() exibe a documentação de qualquer função, módulo ou objeto. Este exemplo exibe a documentação da função print().

Projeto

Esse projeto representa um **Gerenciador de Tarefas** onde é possível adicionar, listar e remover tarefas. O código está bem comentado para facilitar a explicação na faculdade.

Funcionalidades do projeto:

- Adicionar uma nova tarefa
- Listar todas as tarefas
- Remover uma tarefa pelo índice

Explicação:

1. **Criação da classe GerenciadorDeTarefas**:
   - Possui um atributo tarefas que armazena as tarefas em uma lista.
2. **Método adicionar_tarefa**:
   - Adiciona uma nova tarefa à lista e exibe uma mensagem de confirmação.
3. **Método listar_tarefas**:
   - Percorre a lista e exibe as tarefas de maneira numerada.
   - Se não houver tarefas, exibe uma mensagem informando.
4. **Método remover_tarefa**:
   - Remove uma tarefa da lista com base no índice informado pelo usuário.
   - Garante que o índice fornecido seja válido antes de remover.
5. **Testando a classe**:
   - Criamos uma instância do GerenciadorDeTarefas.
   - Adicionamos três tarefas de exemplo.
   - Listamos as tarefas cadastradas.
   - Removemos uma tarefa específica.
   - Listamos novamente para verificar a remoção.

Codigo

```
class GerenciadorDeTarefas:

    """
    Classe para gerenciar uma lista de tarefas simples.
    Permite adicionar, listar e remover tarefas.
    """

    def __init__(self):
        """Inicializa a lista de tarefas vazia"""
        self.tarefas = []

    def adicionar_tarefa(self, descricao):
        """
        Adiciona uma nova tarefa à lista.

        Parâmetros:
        descricao (str): Descrição da tarefa.
        """
        self.tarefas.append(descricao)
        print(f"Tarefa adicionada: {descricao}")

    def listar_tarefas(self):
        """
        Lista todas as tarefas cadastradas.
        """
        if not self.tarefas:
            print("Nenhuma tarefa cadastrada.")
        else:
            print("\nLista de Tarefas:")
            for i, tarefa in enumerate(self.tarefas):
                print(f"{i + 1}. {tarefa}")

    def remover_tarefa(self, indice):
        """
        Remove uma tarefa da lista com base no índice informado.

        Parâmetros:
        indice (int): Índice da tarefa a ser removida (começa de 1 para o
usuário).
        """
        if 1 <= indice <= len(self.tarefas):
            tarefa_removida = self.tarefas.pop(indice - 1)
            print(f"Tarefa removida: {tarefa_removida}")
        else:
            print("Índice inválido! Tente novamente.")
```

```
# Criando uma instância da classe
gerenciador = GerenciadorDeTarefas()

# Adicionando algumas tarefas
gerenciador.adicionar_tarefa("Estudar para a prova de matemática")
gerenciador.adicionar_tarefa("Ler um artigo sobre Inteligência
Artificial")
gerenciador.adicionar_tarefa("Fazer exercícios físicos")

# Listando as tarefas
gerenciador.listar_tarefas()

# Removendo uma tarefa
gerenciador.remover_tarefa(2)

# Listando as tarefas novamente
gerenciador.listar_tarefas()
```

Esse código é um ótimo exemplo prático de POO (Programação Orientada a Objetos)