

# **Básico de Python**

Sergio Pedro Rodrigues Oliveira

# SUMÁRIO

<b>1</b>	<b>Diagrama de estudo</b>	<b>1</b>
<b>2</b>	<b>Variáveis e tipos de dados simples</b>	<b>1</b>
2.1	<code>print()</code>	1
2.2	<code>print()</code> com variáveis	1
2.3	Regras de nomes de variáveis	2
2.4	Concatenando strings	3
2.5	Métodos auxiliares da função <code>print()</code>	3
2.6	Caracteres de escape	5
2.7	Removendo espaços em branco <code>print()</code>	6
2.8	Números	7
2.9	Funções de conversão de tipo	7
2.10	Descobrimo o tipo da variável usando a função <code>type()</code>	8
2.11	Operações básicas	9
2.12	Biblioteca <code>math</code> para ampliar operações matematicas	10
2.13	Operações lógicas básicas	11
2.14	Operadores de identidade	12
2.15	Operações de associação	13
2.16	Comentários	14
2.17	Zen Python	15
<b>3</b>	<b>Listas</b>	<b>16</b>
3.1	Lista	16
3.2	Acessando elementos de uma lista	16
3.3	Alterando, acrescentando e removendo elementos	17
3.3.1	Modificando elementos de uma lista	17
3.3.2	Acrescentando elementos em uma lista	18
3.3.2.1	Concatenando elementos no final de uma lista, método <code>.append()</code>	18
3.3.2.2	Inserindo elementos em uma lista, método <code>.insert()</code>	18
3.3.3	Removendo elementos de uma lista	19
3.3.3.1	Instrução <code>del</code>	19
3.3.3.2	Método <code>.pop()</code>	19
3.3.3.3	Método <code>.remove()</code>	20
3.4	Organizando uma lista	22
3.4.1	Método <code>.sort()</code>	22
3.4.2	A função <code>sorted()</code>	23
3.4.3	Método <code>.reverse()</code>	24
3.5	Descobrimo o tamanho de uma lista - <code>len()</code>	25

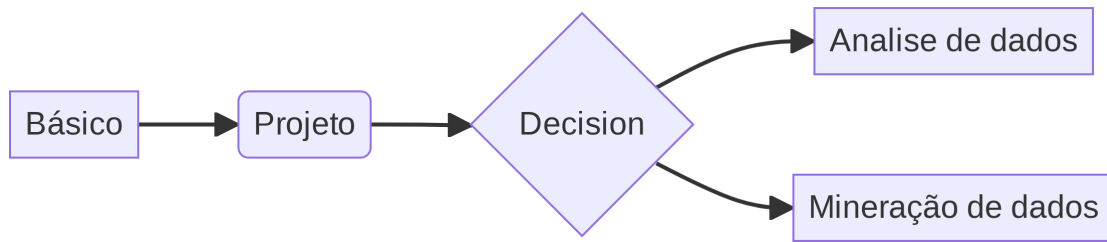
<b>4</b>	<b>Trabalhando com listas</b>	<b>26</b>
4.1	Percorrendo uma lista inteira com um laço . . . . .	26
4.2	Erros comuns de indentação . . . . .	26
4.3	Listas numéricas . . . . .	27

## LISTA DE FIGURAS

## LISTA DE TABELAS

1	Caracteres de escape . . . . .	5
2	Principais tipos de dados . . . . .	7
3	Funções de conversão de tipo . . . . .	7
4	Operações básicas . . . . .	9
5	Algumas operações da biblioteca <code>math</code> . . . . .	10
6	Operações Lógicas . . . . .	11
7	Operadores identidade . . . . .	12
8	Operadores de associação . . . . .	13

## 1 Diagrama de estudo



## 2 Variáveis e tipos de dados simples

### 2.1 print()

Print é uma função que exibe uma string na tela.

Exemplo:

```
print("string")
```

string

### 2.2 print() com variáveis

Podemos usar a função `print()` para imprimir uma variável string.

Exemplo:

```
message = "Hello world!"  
print(message)
```

Hello world!

## 2.3 Regras de nomes de variáveis

Regras ou diretrizes para usar variáveis em Python.

- Nomes de variáveis deve conter apenas letras, números e underscores. Podemos começar a variável com letra ou underscore, mas nunca com um número.
- Espaços não são permitidos em nomes de variáveis, mas underscores podem ser usados para separar palavras.
- Evite usar palavras reservadas e nome de funções em Python como nome de variáveis.
- Nomes de variáveis devem ser concisos, porém descritivos.
- Tome cuidado ao usar a letra l e a letra maiúscula O, pois podem ser confundidas com os números 1 e 0.

## 2.4 Concatenando strings

Podemos usar o simbolo de (+) para combinar strings (concatenar).

Exemplo:

```
first_name = "ada"
last_name = "lovelace"
full_name = first_name + " " + last_name
print("Hello, " + full_name.title() + "!")
```

Hello, Ada Lovelace!

Os espaços em branco entre aspas servem para criar espaços na string.

## 2.5 Métodos auxiliares da função print()

### 1. .title()

Coloca apenas as primeiras letras em maiúsculas de cada palavra e o resto em minúscula.

Exemplo:

```
full_name = "ada lovelace"
print(full_name.title())
```

Ada Lovelace

### 2. .upper()

Coloca todas as letras em maiúsculas.

Exemplo:

```
full_name = "ada lovelace"
print(full_name.upper())
```

ADA LOVELACE

### 3. .lower()



Coloca todas as letras em minúsculas. O método `.lower()` é particularmente útil para armazenar dados. Converter os dados em minúscula antes de armazenar.

Exemplo:

```
full_name = "ada lovelace"  
print(full_name.lower())
```

```
ada lovelace
```

## 2.6 Caracteres de escape

Podemos inserir alguns caracteres de escape no texto para executar alguma ação, como pular linha, gerar tabulação e etc. Alguns caracteres podem ser vistos na [Table 1](#).

Todos os caracteres de escape começam com barra(\) + complemento.

Table 1: Caracteres de escape

Caracteres de escape	Descrição
\t	Gera tabulação (tab).
\n	Gera quebra de linha.

Exemplo:

```
print("Language:\nPython\nJava\nC\nJavaScript")
```

```
Language:
Python
Java
C
JavaScript
```

## 2.7 Removendo espaços em branco print()

### 1. .rstrip()

Remove espaço em branco do lado direito.

Exemplo:

```
favorite_language = 'python '  
favorite_language.rstrip()
```

'python'

### 2. .lstrip()

Remove espaço em branco do lado esquerdo.

Exemplo:

```
favorite_language = ' python'  
favorite_language.lstrip()
```

'python'

### 3. .strip()

Remove os espaços em branco dos dois lados ao mesmo tempo.

Exemplo:

```
favorite_language = ' python '  
favorite_language.strip()
```

'python'

- Os metodos usados não removem os espaços em branco em definitivo, para remover em definitivo é necessario armazenar o valor novo na variável.

```
favorite_language = ' python '  
favorite_language = favorite_language.strip()  
favorite_language
```

'python'

## 2.8 Números

A linguagem Python faz tipagem automática (dinâmica), tipa a variável de acordo com o uso. E o Python contém uma tipagem forte, não faz converção automática do tipo de uma variável para executar uma ação (operação).

Em resumo, python tem é uma linguagem de tipagem dinâmica e forte.

Os principais tipos de dados no Python são estão presentes na Table 2.

Table 2: Principais tipos de dados

Nome	Abreviação	Descrição
Inteiro	int	Números inteiros
Ponto flutuante	float	Números com ponto decimal

## 2.9 Funções de conversão de tipo

Podemos converter variáveis para determinado tipo especificado usando funções de conversão de tipo, como pode ser observado na Table 3.

Converter uma variável não é permanente, a não ser que a ação seja armazenada na variável explicitamente.

Table 3: Funções de conversão de tipo

Tipo para converter	Função	Descrição
int	<code>int()</code>	Converte variável para o tipo inteiro(int)
float	<code>float()</code>	Converte variável para o tipo float
string	<code>str()</code>	Converte variável para o tipo string

A função `str()` é deveras importante, pois pode auxiliar na função `print()`. A função `print()` só imprime na tela variáveis string, sendo assim, precisamos converter as variáveis de outros tipos para string (pelo menos, momentaneamente), para cumprir essa condição.

Exemplo:

```
age = 23
print("Happy " + str(age) + "rd Birthday!")
```

Happy 23rd Birthday!

## 2.10 Descobrindo o tipo da variável usando a função `type()`

Podemos usar a função `type()` para descobrir o tipo de determinada variável.

```
age = 23  
print (type(age))
```

```
<class 'int'>
```

É uma **boa prática** usar a função `type()`, para conferir o tipo da variável, antes de manipular alguma variável. Assim o programador terá o controle sobre as variáveis que esta trabalhando. Essa boa prática evita erros.

Também é uma **boa prática**, ao identificar/observar um erro, conferir os tipos das variáveis envolvidas. É um dos erros mais comuns: erro de tipagem.

## 2.11 Operações básicas

A Table 4 apresenta as principais operações básicas do python.

Table 4: Operações básicas

Operação	Símbolo	Exemplo
Soma	+	$2+2=4$
Subtração	-	$3-2=1$
Multiplicação	*	$2*3=6$
Divisão	/	$5/4=1.25$
Divisão inteira	//	$5//4=1$
Resto da divisão (módulo)	%	$10\%8=2$
Potência	**	$3**2=9$
Raiz	**	$4**0.5=2$

## 2.12 Biblioteca `math` para ampliar operações matemáticas

Podemos usar o pacote `math` para ampliar as funções matemáticas do Python (básicas, trigonométricas e estatísticas). A Table 5 apresenta as principais funções básicas da biblioteca `math`.

Table 5: Algumas operações da biblioteca `math`

Operação	Símbolo	Exemplo
Soma	<code>math.add(x,y)</code>	<code>math.add(2,2) = (2+2)=4</code>
Subtração	<code>math.subtract(x,y)</code>	<code>math.subtract(2,2) = (2-2)=0</code>
Raiz quadrada	<code>math.sqrt()</code>	<code>math.sqrt(4)=2</code>
Potência	<code>math.pow(x,y)</code>	<code>math.pow(2,3) = (2**3)=8</code>
Seno	<code>math.sin()</code>	<code>math.sin()</code> , retorna um ângulo em radianos.
Cosseno	<code>math.cos()</code>	<code>math.cos()</code> , retorna um ângulo em radianos.
Tangente	<code>math.tan()</code>	<code>math.tan()</code> , retorna um ângulo em radianos.
potencia de Euler	<code>math.exp(x)</code>	<code>math.exp(x) = math.pow(math.e**x)</code>
Logaritmo natural, ou log neperiano	<code>math.log(x)</code>	<code>math.log(2)=0.69</code>
Logaritmo	<code>math.log(x[,base])</code>	<code>math.log(2,10)=0.3</code>

Para converter o ângulo para radianos podemos usar a função `math.radians()`.

```
import math
#Seno do ângulo de 45°
#Resultado em Radianos
print(str(math.sin(math.radians(45))))
```

0.7071067811865475

Para converter de radiano para grau podemos usar a função `math.degrees()`.

```
import math
#Seno do ângulo de 45°
#Resultado em ângulo
print(str(math.degrees(math.sin(math.radians(45)))))
```

40.51423422706977

## 2.13 Operações lógicas básicas

A Table 6 apresenta as principais operações lógica básica do python. As operações lógicas retornam `True` ou `False`.

Table 6: Operações Lógicas

Operação	Nome	Função	Exemplo
<code>==</code>	Igual a	Varifica se um valor é igual ao outro.	<code>1==1 = True</code>
<code>!=</code>	Diferente de	Varifica se um valor é diferente ao outro.	<code>1!=2 = True</code>
<code>&gt;</code>	Maior que	Varifica se um valor é maior que outro.	<code>5&gt;1 = True</code>
<code>&gt;=</code>	Maior ou igual	Varifica se um valor é maior ou igual a outro.	<code>5&gt;=5 = True</code>
<code>&lt;</code>	Menor que	Varifica se um valor é menor que outro.	<code>1&lt;5 = True</code>
<code>&lt;=</code>	Menor ou igual	Varifica se um valor é menor ou igual a outro.	<code>1&lt;=4 = True</code>
<code>and</code>	E	Retorna <code>True</code> se ambas as afirmações forem verdadeiras.	<code>(1==1) and (4&lt;5)</code>
<code>or</code>	Ou	Retorna <code>True</code> se uma das afirmações for verdadeiras.	<code>(1==1) or (2&lt;1)</code>
<code>not</code>	Negação	Retorna <code>Falso</code> se o resultado for verdadeiro, ou o contrario.	<code>not (1==1) = False</code>



## 2.14 Operadores de identidade

Os operadores de identidade, Table 7, são utilizados para comparar objetos, se os objetos testados referenciam o mesmo objeto.

Table 7: Operadores identidade

Operador	Definição
is	Retorna <b>True</b> se ambas as variáveis são o mesmo objeto.
is not	Retorna <b>True</b> se ambas as variáveis não são o mesmo objeto.

Exemplo de operações de identidade:

```
lista = [1,2,3]
outra_lista = [1,2,3]
recebe_lista = lista

print(f"São o mesmo objeto: {lista is outra_lista}")
```

São o mesmo objeto: False

```
lista = [1,2,3]
outra_lista = [1,2,3]
recebe_lista = lista

print(f"São o mesmo objeto: {lista is recebe_lista}")
```

São o mesmo objeto: True

## 2.15 Operações de associação

Os operadores de associação, Table 8, servem para verificar se determinado objeto esta **associado** ou **pertence** a determinada estrutura de dados.

Table 8: Operadores de associação

Operação	Função
in	Retorna <b>True</b> caso valor seja encontrado na sequência.
not in	Retorna <b>True</b> caso valor não seja encontrado na sequência.

Exemplos de operações de associação:

```
lista = ["Python", 'Academy', "Operadores", 'Condições']  
print('Python' in lista)
```

True

```
lista = ["Python", 'Academy', "Operadores", 'Condições']  
print('SQL' not in lista)
```

True

## 2.16 Comentários

Um comentário permite escrever notas em seus programas em linguagem natural. Em Python, o caractere sustenido (#) indica um comentário. Tudo que vier depois de um caractere sustenido em seu código será ignorado pelo interpretador Python.

**Boas práticas** em comentários:

1. Explicar o que o código deve fazer.
2. Como faz para funcionar.

## 2.17 Zen Python

É um guia de **boas práticas**.

```
import this
```

The Zen of Python, by Tim Peters

```
Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *right* now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!
```

Principais pontos:

1. Bonito é melhor do que feio.
2. Simples é melhor que complexo.
3. Complexo é melhor que complicado.
4. Legibilidade conta.
5. Deve haver uma - e, de preferência, apenas uma - maneira óbvia de fazer algo.
6. Agora é melhor que nunca.

## 3 Listas

### 3.1 Lista

Uma lista é uma coleção de itens em uma ordem em particular. Os colchetes([]) indicam uma lista e os elementos individuais de uma lista são separados por vírgula. [ver [1](#), p. 71]

Exemplo:

```
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

### 3.2 Acessando elementos de uma lista

Podemos acessar a qualquer item de uma lista informando a posição, ou índice. As posições de uma lista começam no 0, e não no 1.

Para acessar um elemento de uma lista, informamos o nome da lista seguido do índice do item entre colchetes.

Exemplo:

```
#Acessando o primeiro item da lista
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles[0].title())
```

Trek

Para acessar a lista de trás pra frente podemos usar a posição invertida seguida do símbolo de menos na frente. Sendo assim, a posição do último item é -1, do penúltimo -2 e assim sucessivamente.

Exemplo:

```
#Acessando o último item da lista
bicycles = ['trek','cannondale','redline','specialized']
print(bicycles[-1].title())
```

Specialized

### 3.3 Alterando, acrescentando e removendo elementos

Dado que a lista é um elemento dinâmico (pode, e provavelmente ocorrerá, de sofrer modificações com o uso), este tópico comentará os principais formas de modificação de listas.

#### 3.3.1 Modificando elementos de uma lista

Para alterar um elemento que você quer modificar, use o nome da lista seguido do índice do elemento que quer modificar, e então forneça um novo valor.

```
#Alterando o item 1 da lista (índice 0)
motorcycles = ['honda','yamaha','suzuki']
motorcycles[0] = 'ducati'
print(motorcycles)
```

```
['ducati', 'yamaha', 'suzuki']
```

### 3.3.2 Acrescentando elementos em uma lista

Existem diversas formas de adicionar elementos a uma lista:

#### 3.3.2.1 Concatenando elementos no final de uma lista, método `.append()`

Adiciona um novo elemento no final da lista usando o método `.append()`.

Exemplo:

```
#Adicionando elemento ao final da lista
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.append('ducati')
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki', 'ducati']
```

#### 3.3.2.2 Inserindo elementos em uma lista, método `.insert()`

Este método insere um elemento em determinada posição da lista, usando o método `.insert(índice, elemento)`.

Exemplo:

```
#Adicionando um item na segunda posição da lista (índice 1)
motorcycles = ['honda', 'yamaha', 'suzuki']
motorcycles.insert(1, 'ducati')
print(motorcycles)
```

```
['honda', 'ducati', 'yamaha', 'suzuki']
```

### 3.3.3 Removendo elementos de uma lista

Os métodos para remover um item, ou um conjunto de itens, de uma lista.

#### 3.3.3.1 Instrução `del`

Se a posição do item que você quer remover de uma lista for conhecida, a instrução `del` remove (deleta) um item em qualquer determinada posição. Depois de removido (deletado) não podemos mais acessar o valor, quando usado a instrução `del`.

```
# Remover (deletar) primeiro item da lista, índice 0
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

del motorcycles[0]
print(motorcycles)
```

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

#### 3.3.3.2 Método `.pop()`

Existem duas formas de usar o método `.pop()`:

1. `.pop()`

As vezes há necessidade de usar o valor de um item depois de removê-lo de uma lista. O método `.pop()` remove o **último** item de uma lista, mas permite que você trabalhe com esse item depois da remoção.

Remove o primeiro item de uma pilha, ou seja, o último item de uma lista.

Para usarmos o item removido é necessário, salva-lo numa variável.

Exemplo:

```
# Uso do método .pop()
# Removendo último item da lista e
# Trabalhando com o item removido.
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

pop_motorcycle = motorcycles.pop()
```



```
print(motorcycles)
print(pop_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki
```

## 2. .pop(índice)

Podemos usar o `.pop()` para remover um item em qualquer posição em uma lista, se incluirmos o índice do item que você deseja remover entre parênteses.

Exemplo:

```
# Uso do método .pop()
# Removendo o segundo item da lista e
# Trabalhando com o item removido.
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)

pop_motorcycle = motorcycles.pop(1)
print(motorcycles)
print(pop_motorcycle)
```

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
yamaha
```

### 3.3.3.3 Método `.remove()`

Remove um item de acordo com o valor. É usado quando sabemos o valor do item, mas não a posição.

O método `.remove()` apaga apenas a primeira ocorrência do valor especificado. Para apagar mais de uma ocorrência será necessário o uso de um laço, para cada ocorrência.

Exemplo:

```
# Uso do método .remove()
# Removendo um item da lista pelo valor
motorcycles = ['honda', 'yamaha', 'ducati']
print(motorcycles)
```

```
too_expensive = 'ducati'  
motorcycles.remove(too_expensive)  
print(motorcycles)
```

```
['honda', 'yamaha', 'ducati']  
['honda', 'yamaha']
```

## 3.4 Organizando uma lista

Dado que com frequência, as listas são organizadas numa ordem imprevisível, se torna necessário organizar as informações em uma ordem particular. O Python tem mecanismos para organizar listas. São eles:

### 3.4.1 Método `.sort()`

Ordena uma lista em ordem alfabética, ou alfabética inversa.

Para ordenar uma lista em ordem alfabética inversa, basta passar o argumento `reverse = True` para o método `.sort()`.

Uma vez ordenada pelo método `.sort()` a lista não retorna a ordem original (ordenação permanente).

Exemplo:

```
# Ordenando a lista cars usando o método .sort()
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.sort()
print(cars)
cars.sort(reverse=True)
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
```

### 3.4.2 A função `sorted()`

A função `sorted()` ordena uma lista de forma temporaria, não altera a lista original, em ordem alfabética. Ou seja, a lista volta a forma original ao final do uso da função.

Assim como no método `.sort()`, podemos ordenar a lista em ordem alfabética inversa adicionando o argumento `reverse=True`.

Exemplo:

```
# Ordenando temporariamente a lista cars usando a função sorted()
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
print(sorted(cars))
print(sorted(cars, reverse=True))
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['audi', 'bmw', 'subaru', 'toyota']
['toyota', 'subaru', 'bmw', 'audi']
['bmw', 'audi', 'toyota', 'subaru']
```

### 3.4.3 Método `.reverse()`

Para inverter a ordem original de uma lista, podemos usar o método `.reverse()`.

O método `.reverse()` não organiza a lista em ordem alfabética inversa, o método inverte a lista original.

O método `.reverse()` ordena de forma permanente a lista, porém se usarmos o método novamente, teremos a lista original. Logo, é fácil reverter o uso do método `.reverse()`.

Exemplo:

```
# Método .reverse() para inverte, de modo permanente, a ordem da lista.
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
['bmw', 'audi', 'toyota', 'subaru']
```

### 3.5 Descobrindo o tamanho de uma lista - len()

Podemos descobrir o tamanho de uma lista usando a função `len()`.

Exemplo:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']  
len(cars)
```

4

## 4 Trabalhando com listas

### 4.1 Percorrendo uma lista inteira com um laço

Podemos usar um laço `for` para percorrer toda uma lista, podendo assim entre outras coisas, efetuar tarefas em cada item da lista.

A estrutura básica do `for` é:

```
for variável_nova in lista :  
    tarefas
```

O laço diz para a cada iteração pegar um elemento da lista e armazenar na nova variável, e executar uma tarefa a cada iteração. Toda tarefa indentada depois dos dois pontos é considerada dentro do laço.

No Python o `for`, usa indentação para determinar o que esta dentro do laço.

Qual quer linha após o laço que não for indentada é considerada fora do laço.

Exemplo:

```
#Executando um laço com base numa lista  
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
    print(magician)
```

```
alice  
david  
carolina
```

### 4.2 Erros comuns de indentação

- Esquecer de indentar.
- Esquecer de indentar linhas adicionais do laço.
- Indentação desnecessária.
- Indentando desnecessariamente após o laço.
- Esquecer os dois-pontos do laço `for`.

### 4.3 Listas numéricas



## Bibliografia

- [1] Eric Matthes. *Curso Intensivo de Python - 3ª Edição: Uma Introdução Prática e Baseada em Projetos à Programação*. Novatec Editora, 2023. ISBN: 9788575228432. URL: <https://books.google.com.br/books?id=mkW7EAAAQBAJ>.