

# MINI E-BOOK PYTHON

Um pequeno guia sobre  
os conceitos básicos da  
linguagem mais amigável  
do mercado.

FERNANDA CARLA DE FREITAS NASCIMENTO

<https://github.com/nascimentofernanda>



---

# MINI E-BOOK PYTHON

Um pequeno guia sobre os conceitos básicos da linguagem mais amigável do mercado.

Seja bem-vindo ao mundo da programação com Python! Este mini e-book foi criado para todos aqueles que desejam embarcar em uma jornada de aprendizado na linguagem de programação mais amigável e versátil do mercado.

***“Uma máquina consegue fazer o trabalho de 50 homens ordinários. Nenhuma máquina consegue fazer o trabalho de um homem extraordinário”.***  
***Elbert Hubbard***

Nas próximas páginas iremos explorar os principais conceitos de Python de forma clara e concisa, com exemplos práticos e alguns exercícios para consolidar o seu conhecimento.

Neste mini e-book, você irá aprender os fundamentos essenciais, como variáveis, estruturas de controle de fluxo, funções, classes e muito mais.

Lembre-se: a chave para o sucesso na programação é a prática consistente. Portanto, não tenha medo de enfrentar desafios e mergulhar de cabeça nos exemplos fornecidos. Quanto mais você praticar, mais confiante se tornará em suas habilidades de programação.

Sem mais delongas, vamos explorar o incrível mundo da programação com Python!

---

---

## 1. O que é o Python?

Python foi criada por Guido van Rossum e teve sua origem no final dos anos 1980. O nome "Python" foi inspirado no programa humorístico britânico chamado "Monty Python's Flying Circus", do qual Guido era fã. Ele desejava um nome único e fácil de lembrar para a sua nova linguagem de programação, e assim o nome "Python" foi escolhido.

A linguagem Python foi concebida com o objetivo de ser fácil de aprender, legível e expressiva. O primeiro lançamento público da linguagem foi a versão 0.9.0 em fevereiro de 1991. Desde então, Python tem passado por várias evoluções e melhorias, chegando à versão 1.0 em janeiro de 1994.

Guido van Rossum liderou o desenvolvimento da linguagem Python até julho de 2018, quando ele decidiu se afastar do cargo de líder de projeto, mas continuou a participar da comunidade Python como um "Benevolente Dictador Vitalício" (BDFL) emérito. Depois de sua saída, a comunidade Python continuou a contribuir e melhorar a linguagem, seguindo os princípios estabelecidos por Guido.

Ao longo dos anos, Python cresceu em popularidade e se tornou uma das linguagens de programação mais utilizadas no mundo, especialmente em áreas como desenvolvimento web, ciência de dados, inteligência artificial, automação, entre outras. Sua simplicidade e flexibilidade têm sido os principais fatores que atraem desenvolvedores e empresas para escolherem Python como sua linguagem de preferência.

---

## 1.1 Principais características:

Python é uma linguagem de programação conhecida por suas características distintas e poderosas, que a tornam amplamente utilizada em diferentes domínios. Algumas das principais características do Python incluem:

- **Sintaxe simples e legível:** A sintaxe de Python é projetada para ser clara e fácil de ler, o que facilita a compreensão do código mesmo por iniciantes. Isso permite que os programadores se concentrem mais na lógica do programa do que na estruturação complexa.
- **Tipagem dinâmica:** Python é uma linguagem de tipagem dinâmica, o que significa que as variáveis não precisam ser declaradas com um tipo específico. O tipo é determinado em tempo de execução, tornando o código mais flexível e permitindo uma maior reutilização.
- **Multiparadigma:** Python suporta múltiplos paradigmas de programação, incluindo programação orientada a objetos, programação imperativa e programação funcional. Isso dá aos desenvolvedores a liberdade de escolher o estilo de programação que melhor se adequa ao problema que estão resolvendo.
- **Biblioteca padrão abrangente:** Python possui uma rica biblioteca padrão que abrange várias áreas, desde manipulação de strings e operações matemáticas até acesso à rede e criação de interfaces gráficas. Isso permite que os desenvolvedores realizem tarefas complexas sem ter que escrever código do zero.
- **Grande comunidade e suporte:** Python possui uma comunidade ativa e engajada de desenvolvedores em todo o mundo. Isso significa que há uma abundância de recursos, documentação e bibliotecas de terceiros disponíveis, facilitando a resolução de problemas e o aprendizado contínuo.
- **Portabilidade:** Python é uma linguagem portátil, o que significa que os programas escritos em Python podem ser executados em diferentes plataformas sem a necessidade de grandes alterações.

- 
- **Interpretada:** Python é uma linguagem interpretada, o que significa que o código-fonte é executado diretamente por um interpretador, em vez de ser compilado em código de máquina. Isso torna o desenvolvimento mais rápido e flexível, pois não é necessário um processo de compilação antes de executar o código.
  - **Open-source:** Python é uma linguagem de código aberto, o que significa que seu código-fonte é livremente disponível para a comunidade. Isso permite que os desenvolvedores contribuam para o seu desenvolvimento e que a linguagem permaneça constantemente atualizada e aprimorada.

Essas características tornam Python uma linguagem versátil e poderosa, adequada para uma ampla variedade de aplicações, desde pequenos scripts até projetos complexos e de escala empresarial. Seu crescimento constante em popularidade é um reflexo de sua simplicidade, eficiência e flexibilidade.

## 1.2 Onde utilizar?

Python é uma linguagem de programação extremamente versátil, o que significa que você pode utilizá-la em uma ampla variedade de contextos e projetos. Algumas das principais áreas em que Python se destaca incluem:

- **Desenvolvimento web:** Python é amplamente utilizado para criar aplicativos web. Frameworks populares como Django e Flask permitem criar sites e aplicações web robustas e escaláveis.
- **Análise de dados:** Python é muito popular entre cientistas de dados e analistas de dados devido às bibliotecas poderosas, como Pandas e NumPy, que facilitam a manipulação e análise de grandes conjuntos de dados.
- **Inteligência Artificial e Aprendizado de Máquina:** Com bibliotecas como TensorFlow, Keras e Scikit-learn, Python se tornou uma escolha de destaque para o desenvolvimento de modelos de aprendizado de máquina e implementação de projetos de inteligência artificial.

- 
- **Automação de tarefas:** Python é frequentemente usado para automatizar tarefas repetitivas e tediosas, tornando o trabalho do dia a dia mais eficiente.
  - **Desenvolvimento de jogos:** Embora não seja tão comum quanto outras linguagens específicas para jogos, Python pode ser usado no desenvolvimento de jogos 2D e prototipagem de jogos mais complexos.
  - **Aplicações desktop:** Com bibliotecas gráficas como Tkinter e PyQt, Python permite criar aplicações desktop multiplataforma com interfaces gráficas atraentes.
  - **Redes e segurança:** Python é usado para criar ferramentas de rede e segurança, como scanners de rede, aplicativos de monitoramento e muito mais.
  - **Automação de processos:** Python é útil para automatizar tarefas em sistemas, como realizar backups, manipulação de arquivos, entre outros.

### 1.3 Instalação do Python

Caso você não tenha o Python instalado no seu computador, basta acessar o link da página oficial: <https://www.python.org/downloads/>

No mesmo site você encontra os tutoriais para instalação

no Linux <https://python.org.br/instalacao-linux/>

no Mac <https://python.org.br/instalacao-mac/> e

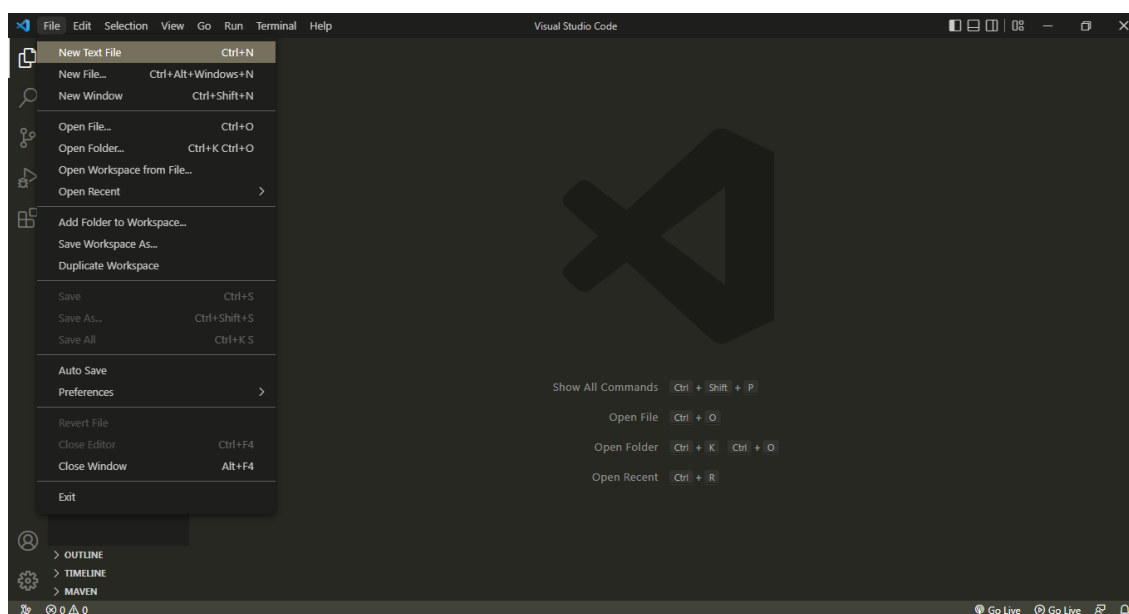
no Windows <https://python.org.br/instalacao-windows/>

Além do Python, você precisará de uma IDE (Ambiente de desenvolvimento) para escrever os seus códigos, para comparar algumas IDEs e escolher a sua favorita indico esse vídeo [aqui](#).

## 1.4 Primeiros passos

Vou utilizar a IDE VS Code, mas fique à vontade para utilizar a que preferir. Crie uma pasta para salvar os arquivos que iremos criar nos exemplos e exercícios do nosso mini e-book.

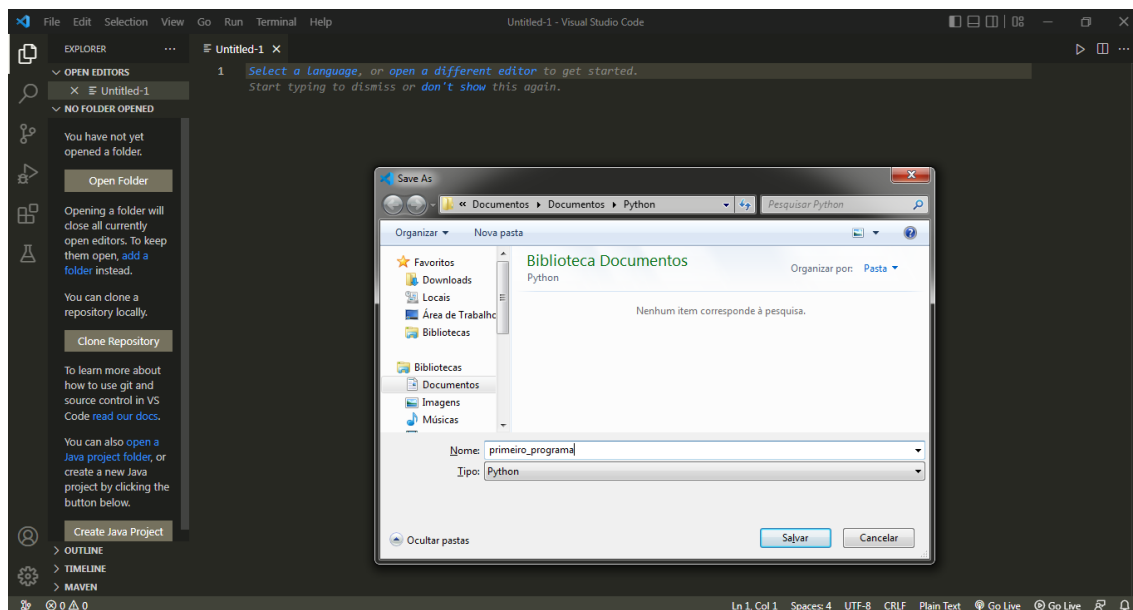
Vamos criar um novo arquivo de texto clicando em File > New Text File ou utilizando as teclas de atalho Ctrl+N



Para salvar, no nome será utilizada a mesma convenção da nomeação de variáveis e funções, o padrão snake\_case, onde é colocado um underline(\_) no lugar do espaço em branco. Será explicado melhor mais para frente.

Pode salvar utilizando File > Save ou Ctrl+S

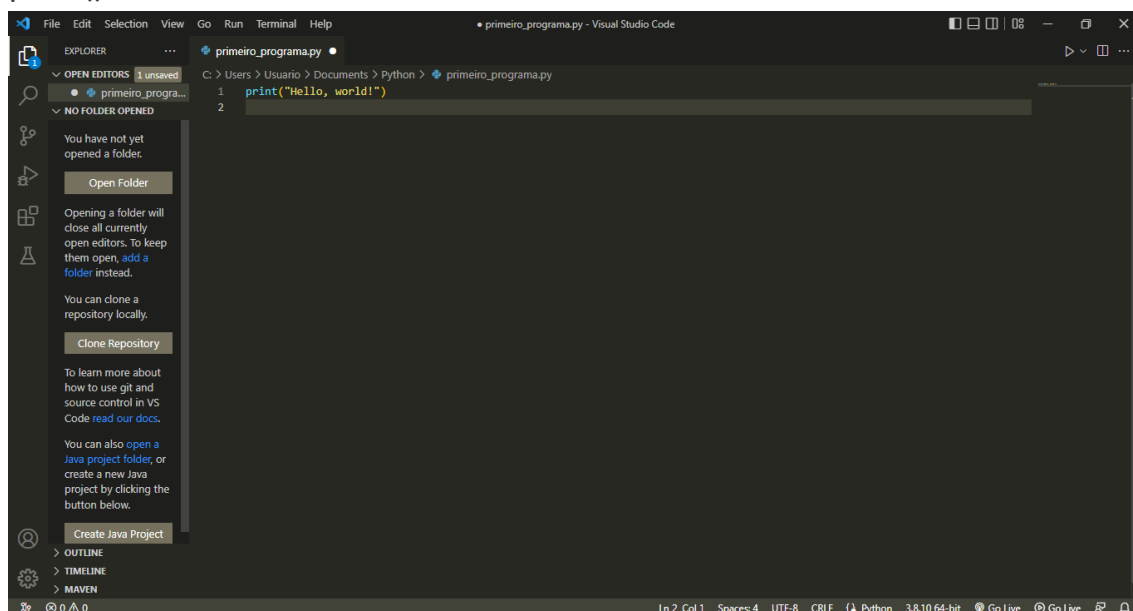




\*Não esqueça de salvar com a extensão Python(.py)!

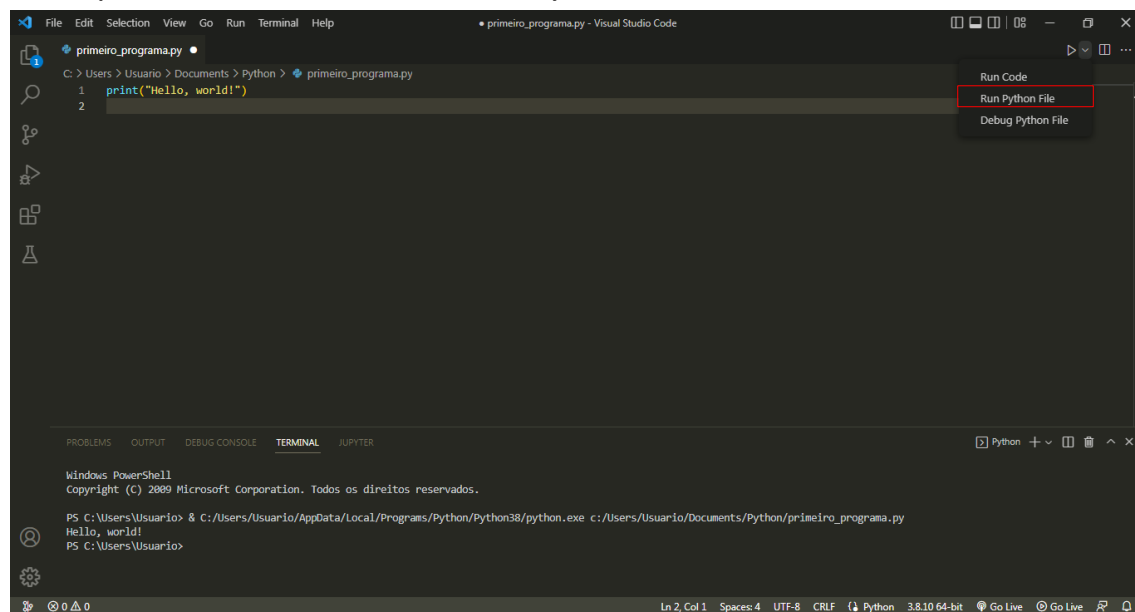
Dizem que quando estamos aprendendo uma nova linguagem, o primeiro programa deve ser exibir a frase: Hello, world! Caso contrário dá azar e não conseguimos aprender. Melhor não tentar a sorte, não é mesmo?

Para exibir uma mensagem na tela o comando utilizado em Python é o comando `print()`



Como o que queremos exibir é um texto e não uma variável, ele deve estar entre aspas.

Tudo pronto vamos clicar em Run Python File



## 2. Conhecendo a linguagem

### 2.1 Tipos de dados

Python suporta diversos tipos de dados, permitindo que você armazene e manipule informações de várias formas. Os tipos de dados padrão mais utilizados da linguagem são:

<b>Texto</b>	str
<b>Numérico</b>	int, float, complex
<b>Sequência</b>	list, tuple, range
<b>Mapa</b>	dict
<b>Coleção</b>	set, frozenset
<b>Booleano</b>	bool
<b>Binário</b>	bytes, bytearray, memoryview

#### Texto:

- str: Representa sequências de caracteres, como "Hello, World!", "Python é incrível!", etc. As strings podem ser delimitadas por aspas simples ou duplas.

---

### **Numérico:**

- **int:** Representa números inteiros, como 1, -20, 1000, etc.
- **float:** Representa números de ponto flutuante, como 3.14, -0.5, 2.0, etc.
- **complex:** Representa números complexos no formato "a + bj", onde "a" e "b" são números reais e "j" é a unidade imaginária (por exemplo, 2 + 3j).

### **Sequência:**

- **list:** É uma coleção ordenada e mutável de elementos, que podem ser de tipos diferentes. Por exemplo, [1, 2, 3], ["maçã", "banana", "laranja"].
- **tuple:** Semelhante a uma lista, mas é imutável, ou seja, seus elementos não podem ser alterados após a criação. Por exemplo, (10, 20, 30).
- **range:** é uma função embutida que cria uma sequência imutável de números inteiros. Essa sequência é frequentemente usada para gerar loops (como em um loop for) ou para criar listas de números de forma concisa. Por exemplo, (1, 10, 2).

### **Mapa:**

- **dict:** É uma estrutura de dados que armazena pares de chave-valor, onde cada chave é única e associada a um valor. É útil para armazenar dados que podem ser recuperados rapidamente usando uma chave específica.

### **Coleção:**

- **set:** Representa uma coleção não ordenada e não duplicada de elementos. Os conjuntos são usados para operações matemáticas de conjunto, como união, interseção, diferença, etc.
- **frozenset:** É um tipo de dados imutável que representa um conjunto (set) de elementos. A principal diferença entre frozenset e set é que o frozenset não pode ser alterado após a sua criação, tornando-o adequado para situações onde você deseja criar um conjunto cujos elementos não podem ser modificados.

### **Booleano:**

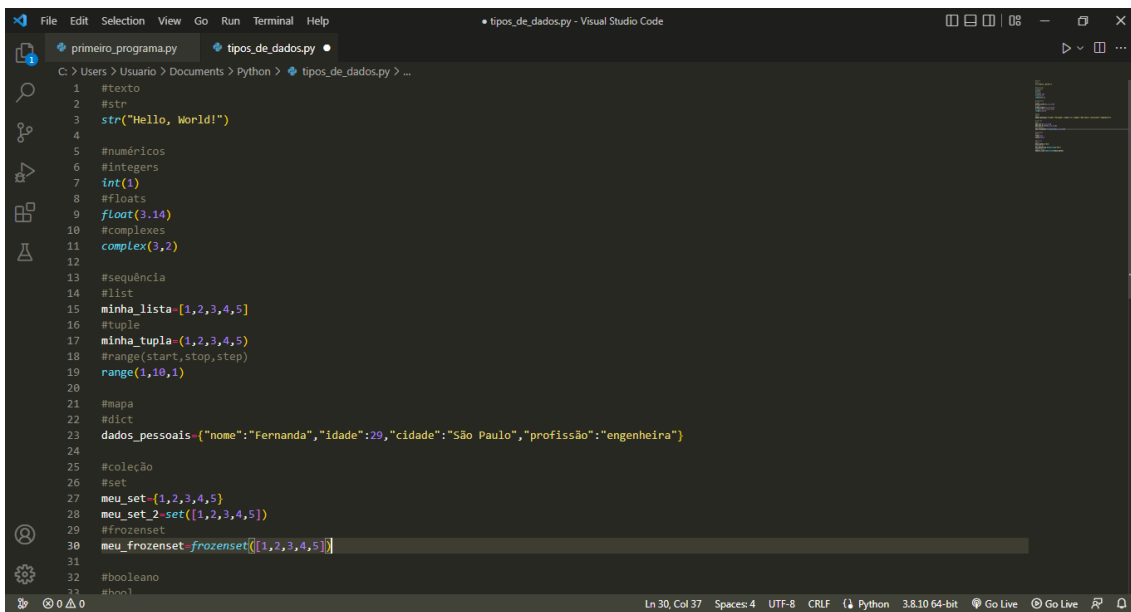
- **bool:** Representa valores lógicos verdadeiro (True) ou falso (False). É útil para expressar condições e tomar decisões em programas.

### **Binário:**

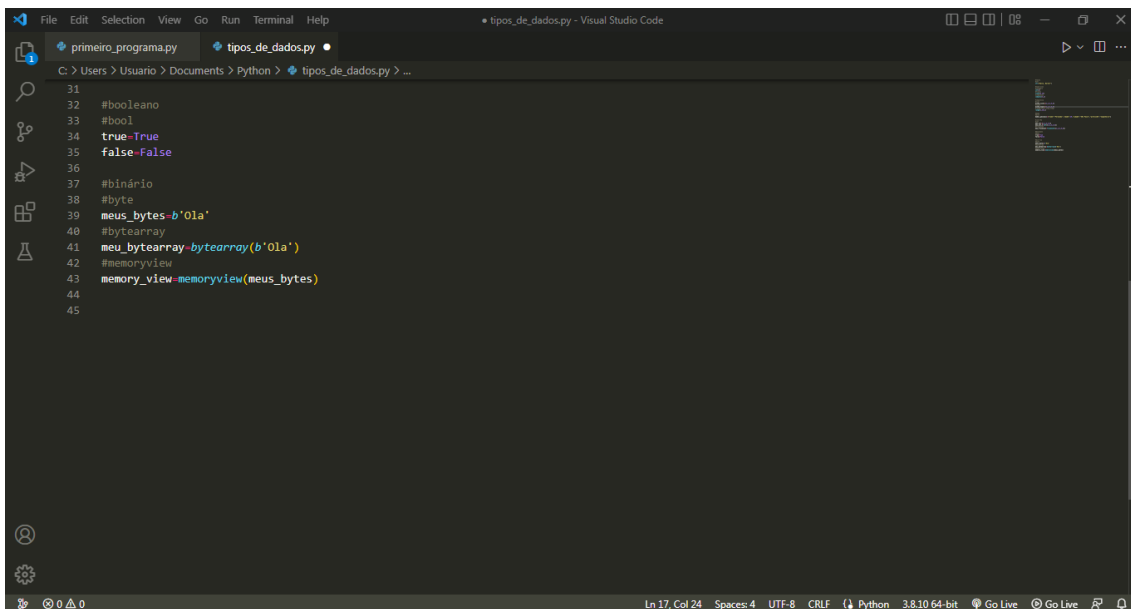
- **bytes:** é um tipo de dados imutável que representa uma sequência de bytes. É usado para armazenar dados binários, como arquivos, imagens, ou qualquer outro tipo de informação que não seja texto.
- **bytearray:** é um tipo de dados mutável que é semelhante ao bytes, mas com a diferença crucial de que os objetos bytearray podem ser modificados após a

sua criação. Isso significa que você pode alterar os elementos individuais do bytearray, adicionando ou removendo bytes conforme necessário.

- **memoryview:** É um tipo de objeto que permite visualizar os dados de um objeto de sequência (como bytes, bytearray ou um array) como uma sequência de bytes sem copiar os dados. Isso proporciona uma forma eficiente de acessar e manipular os dados da sequência original sem criar uma cópia dela.



```
1 #texto
2 #str
3 str("Hello, World!")
4
5 #numéricos
6 #integers
7 int(1)
8 #floats
9 float(3.14)
10 #complexes
11 complex(3,2)
12
13 #sequência
14 #list
15 minha_lista=[1,2,3,4,5]
16 #tuple
17 minha_tupla=(1,2,3,4,5)
18 #range(start,stop,step)
19 range(1,10,1)
20
21 #mapa
22 #dict
23 dados_pessoais={"nome":"Fernanda","idade":29,"cidade":"São Paulo","profissão":"engenheira"}
24
25 #coleção
26 #set
27 meu_set={1,2,3,4,5}
28 meu_set_2=set([1,2,3,4,5])
29 #frozenset
30 meu_frozenset=frozenset([1,2,3,4,5])
31
32 #booleano
33 #bool
```

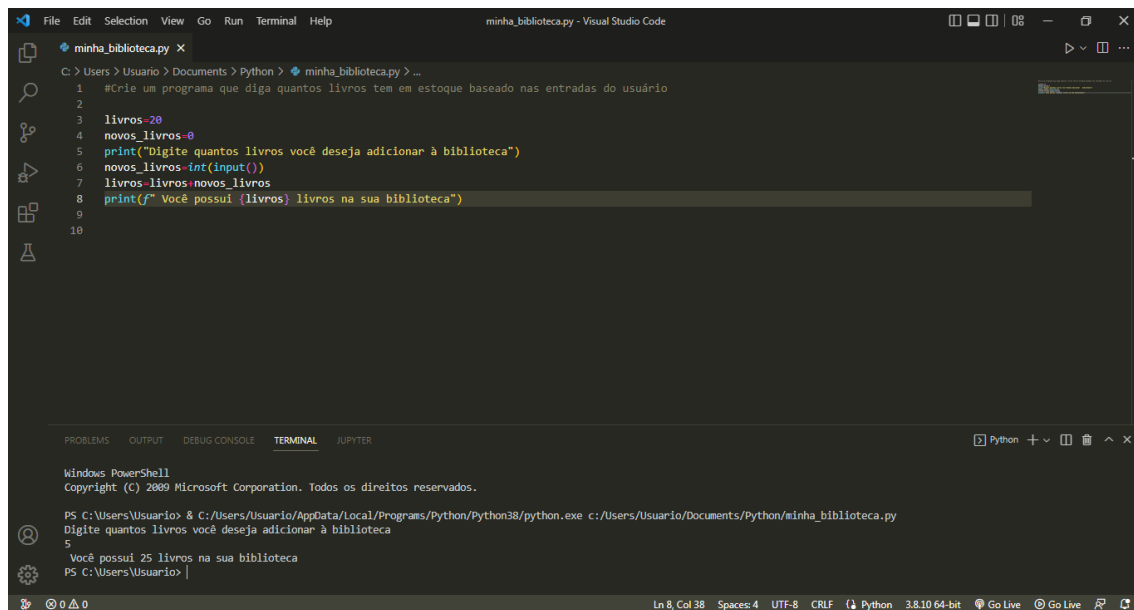


```
31
32 #booleano
33 #bool
34 true=True
35 false=False
36
37 #binário
38 #byte
39 meus_bytes=b'Ola'
40 #bytearray
41 meu_bytearray=bytearray(b'Ola')
42 #memoryview
43 memory_view=memoryview(meus_bytes)
44
45
```

## 2.1.2 Variáveis e constantes

O que são variáveis?

Variáveis são valores previamente armazenados que irão variar no decorrer do programa. Como assim? Por exemplo, vamos supor que você tem um programa para contar quantos livros você tem, supondo que hoje você tinha 20 livros e esse valor está armazenado em uma variável (exemplo: `livros = 20`), então você adiciona mais 5 livros para atualizar seu “estoque”, o valor de livros não será mais 20 e sim 25. Um exemplo desse programa pode ser visto abaixo



```
minha_biblioteca.py - Visual Studio Code
1 #Crie um programa que diga quantos livros tem em estoque baseado nas entradas do usuário
2
3 livros=20
4 novos_livros=0
5 print("Digite quantos livros você deseja adicionar à biblioteca")
6 novos_livros=int(input())
7 livros=livros+novos_livros
8 print(f"Você possui {livros} livros na sua biblioteca")
9
10
```

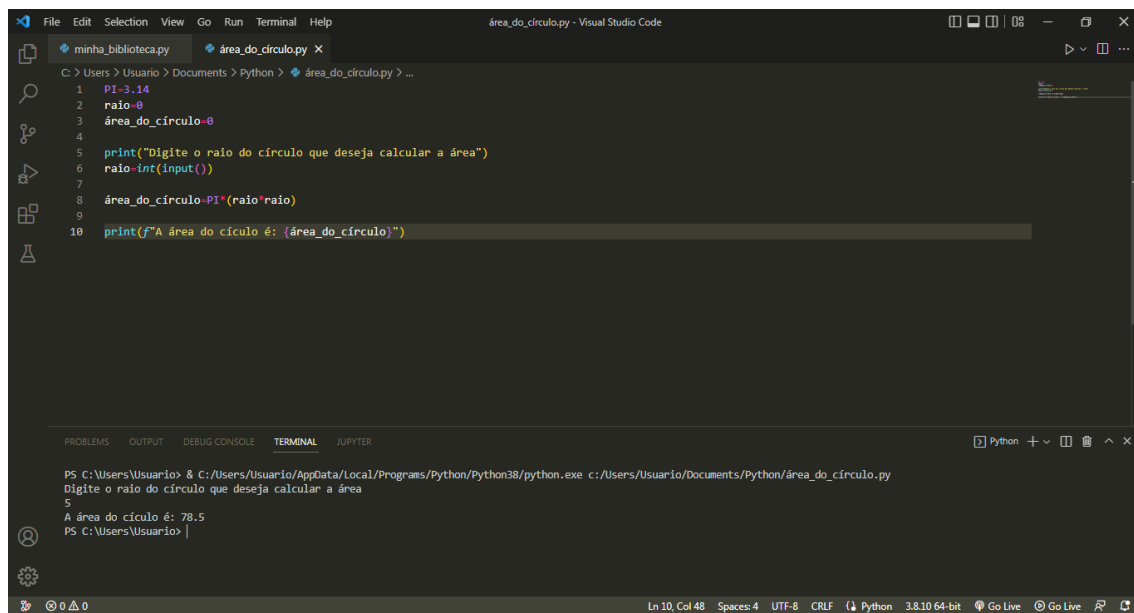
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

Windows PowerShell  
Copyright (C) 2009 Microsoft Corporation. Todos os direitos reservados.

PS C:\Users\Usuario> & C:/Users/Usuario/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Usuario/Documents/Python/minha\_biblioteca.py  
Digite quantos livros você deseja adicionar à biblioteca  
5  
Você possui 25 livros na sua biblioteca  
PS C:\Users\Usuario> |

Ln 8, Col 38 Spaces: 4 UTF-8 CRLF Python 3.8.10 64-bit Go Live Go Live

Já nas constantes os valores são imutáveis ao longo do programa, o valor declarado não se altera. Porém, ao contrário de outras linguagens, Python não tem uma palavra reservada para constantes, então, como convenção de boas práticas as constantes devem ser declaradas com todas as letras maiúsculas.



```
File Edit Selection View Go Run Terminal Help
área_do_circulo.py - Visual Studio Code

minha_biblioteca.py área_do_circulo.py X
C:\Users\Usuario> Documents > Python > área_do_circulo.py > ...
1 PI=3.14
2 raio=0
3 área_do_circulo=0
4
5 print("Digite o raio do círculo que deseja calcular a área")
6 raio=int(input())
7
8 área_do_circulo=PI*(raio*raio)
9
10 print(f"A área do círculo é: {área_do_circulo}")

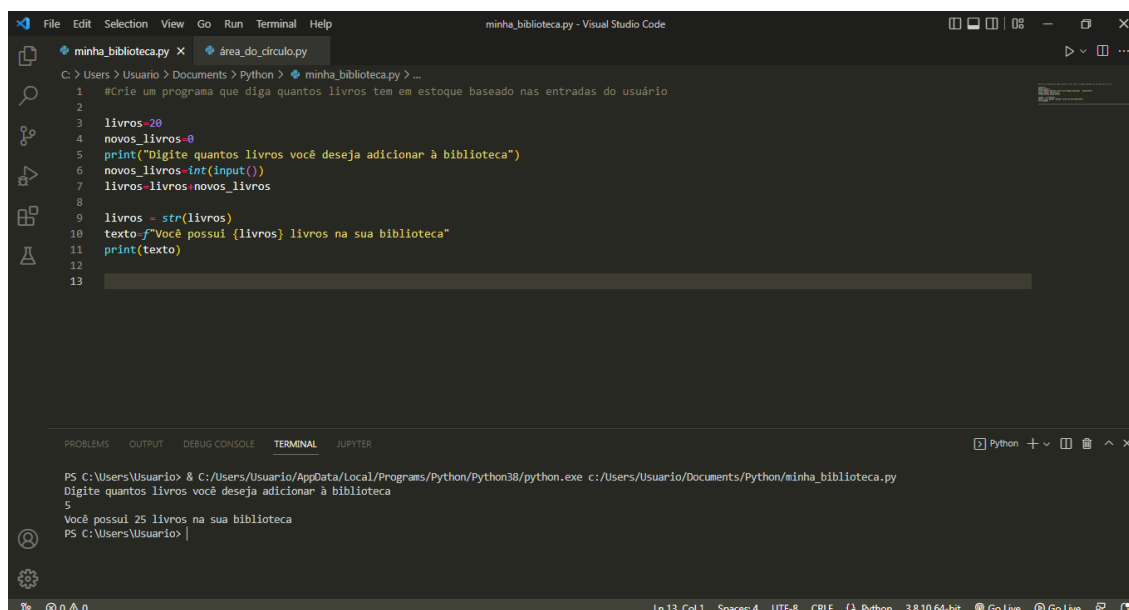
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Python + Python Python 3.8.10 64-bit Go Live Go Live

PS C:\Users\Usuario> & c:/Users/Usuario/AppData/Local/Programs/Python/Python38/python.exe c:/Users/Usuario/Documents/Python/área_do_circulo.py
Digite o raio do círculo que deseja calcular a área
5
A área do círculo é: 78.5
PS C:\Users\Usuario>
```

O programa acima é utilizado para calcular a área de um círculo, para isso é utilizada a fórmula  $A = \pi * R^2$  onde Pi é o valor fixo 3.14, por isso, o valor de Pi foi alocado na constante PI.

### 2.1.3 Conversão de tipos

Às vezes os dados estão armazenados em um tipo de dados, porém, precisamos que o dado seja de outro tipo para realizar alguma operação como, por exemplo, quando o número está armazenado como string e queremos realizar alguma operação matemática, nesse caso seria necessário converter a string para int. De forma semelhante o dado pode estar armazenando como int ou float, mas precisamos dele como str para concatenar com algum texto, então, precisamos converter o int ou float para str. Vamos utilizar o primeiro exemplo novamente para demonstrar uma conversão.



```
minha_biblioteca.py - Visual Studio Code
minha_biblioteca.py x área_do_circulo.py
C:\Users\Usuario> Python > Python > minha_biblioteca.py > ...
1 #Crie um programa que diga quantos livros tem em estoque baseado nas entradas do usuário
2
3 livros=20
4 novos_livros=0
5 print("Digite quantos livros você deseja adicionar à biblioteca")
6 novos_livros=int(input())
7 livros=livros+novos_livros
8
9 livros = str(livros)
10 texto=f"Você possui {livros} livros na sua biblioteca"
11 print(texto)
12
13
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Python + - - - - -
PS C:\Users\Usuario> & C:\Users\Usuario\AppData\Local\Programs\Python\Python38\python.exe c:\Users\Usuario\Documents\Python\minha_biblioteca.py
Digite quantos livros você deseja adicionar à biblioteca
5
Você possui 25 livros na sua biblioteca
PS C:\Users\Usuario>
```

No nosso exemplo, convertemos a variável **livros** que inicialmente era um número inteiro (int) em uma string (str) em seguida, realizamos a concatenação em uma outra variável, a variável texto que também é uma str e exibimos essa variável texto para o usuário.

## 2.1.4 Funções de entrada e saída

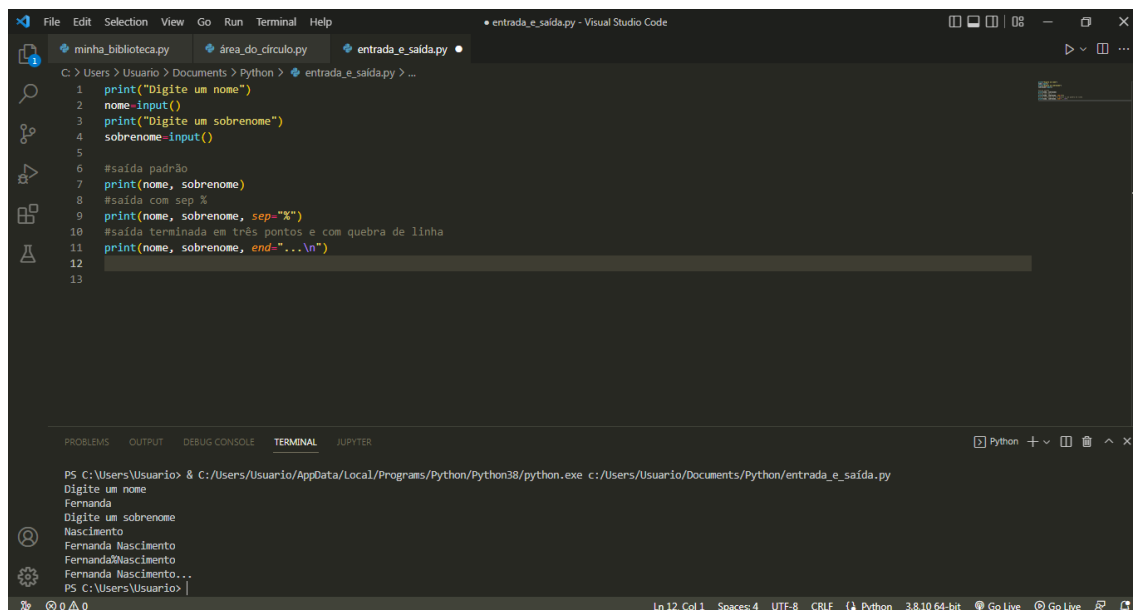
Nos exemplos acima já tivemos contato com as funções de entrada e saída, agora vamos entender um pouco mais sobre elas.

### Função de entrada **input**

É uma função builtin (não precisa ser importada) utilizada para ler dados do teclado. O usuário digita o valor que é lido pela função e salva como string.

### Função de saída **print**

É também uma função builtin utilizada para exibir dados na tela. Exibe o valor do objeto colocado entre parênteses. Os objetos são convertidos para string, separados por sep e terminados em end. (sep e end dão parâmetros opcionais).



```
File Edit Selection View Go Run Terminal Help
• entrada_e_saida.py - Visual Studio Code

C:\Users\Usuario\Documents\Python > entrada_e_saida.py > ...
1 print("Digite um nome")
2 nome=input()
3 print("Digite um sobrenome")
4 sobrenome=input()
5
6 #saída padrão
7 print(nome, sobrenome)
8 #saída com sep %
9 print(nome, sobrenome, sep="%")
10 #saída terminada em três pontos e com quebra de linha
11 print(nome, sobrenome, end="...\n")
12
13

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
Python + - - - - -

PS C:\Users\Usuario> & C:\Users\Usuario\AppData\Local\Programs\Python\Python38\python.exe c:\Users\Usuario\Documents\Python\entrada_e_saida.py
Digite um nome
Fernanda
Digite um sobrenome
Nascimento
Fernanda Nascimento
FernandaNascimento
Fernanda Nascimento...
PS C:\Users\Usuario>
```

- **print("Digite um nome")** – Exibe a mensagem Digite um nome na tela
- **nome=input()** – Grava o dado digitado pelo usuário na variável nome
- **print("Digite um sobrenome")** – Exibe a mensagem Digite um sobrenome na tela
- **sobrenome=input()** – Grava o dado digitado pelo usuário na variável sobrenome
- **print(nome, sobrenome)** – Exibe as variáveis nome e sobrenome separadas por espaço (forma padrão)
- **print(nome, sobrenome, sep="%")** – Exibe as variáveis nome e sobrenome separadas pelo caractere % (por causa do argumento sep="%")
- **print(nome, sobrenome, end="...\n")** – Exibe as variáveis nome e sobrenome separadas por espaço e terminadas em três pontos com quebra de linha (por causa do argumento end="...\n")

Todos os exemplos se encontram no diretório

<https://github.com/nascimentofernanda/E-book-Python>



## Exercícios:

2.1 Qual o tipo de variável para texto?

2.2 Quais os tipos de variáveis numéricas?

2.3 Qual a diferença entre variáveis e constantes e qual a boa prática utilizada para diferenciá-las?

2.4 Quais as funções padrão de entrada e saída em Python?

2.5 Crie um programa que leia um número do teclado depois converta esse número para inteiro e float e imprima os resultados para o usuário.

2.6 Crie um programa que leia 3 palavras do teclado e imprima essas palavras:

- Na forma padrão (separadas por espaço);
- Separadas por hífen (-);
- Separadas por espaço e terminadas em ponto de exclamação.

Compartilhe seus exercícios fazendo um Fork/Pull request no repositório do GitHub:

<https://github.com/nascimentofernanda/E-book-Python> Instruções no arquivo README

## 2.2 Tipos de operadores

Agora que já sabemos como ler e armazenar nossos dados, vamos começar a aprender a como manipulá-los.

### 2.2.1 Operadores aritméticos

Como o nome já sugere os operadores aritméticos são utilizados para realizar operações matemáticas entre dados numéricos. Exemplos de operadores:

Adição	+
Subtração	-
Multiplicação	*
Divisão	/

Divisão inteira	//
Módulo	%
Exponenciação	**

```
1 #SOMA
2 print(2+2)
3
4 #SUBTRAÇÃO
5 print(2-2)
6
7 #MULTIPLICAÇÃO
8 print(2*2)
9
10 #DIVISÃO
11 print(2/2)
12
13 #DIVISÃO INTEIRA
14 print(2//2)
15
16 #MÓDULO
17 print(2%2)
18
19 #EXPONENCIAÇÃO
20 print(2**2)
```

res.py"

4  
0  
4  
1.0  
1  
0  
4

PS C:\Users\Usuario>

- Adição: soma os dois números ou variáveis
- Subtração: subtrai os dois números ou variáveis
- Multiplicação: multiplica os dois números ou variáveis
- Divisão: Divide os dois números ou variáveis retornando um float
- Divisão inteira: Divide os dois números ou variáveis retornando um número inteiro
- Módulo: Retorna o resto da divisão entre os dois números ou variáveis
- Exponenciação: Eleva o primeiro número ou variável ao segundo número ou variável

A precedência dos operadores utilizados em operações em Python segue a mesma regra da matemática.

## 2.2.2 Operadores de comparação

Utilizados para comparar dois valores. Retorna um valor booleano.

Igualdade	==
Diferença	!=
Maior que	>

Menor que	<
Maior ou igual	>=
Menor ou igual	<=

```
operadores_de_comparação.py - Visual Studio Code
operadores_matemáticos.py U operadores_de_comparação.py U
F:\Pen drive\Developer\E-books\E-book-Python\Exemplos do E-book\Capítulo 2 > operadores_de_comparação.py
1 #IGUALDADE
2 print(2==3)
3
4 #DIFERENÇA
5 print(2!=3)
6
7 #MAIOR QUE
8 print(2>3)
9
10 #MENOR QUE
11 print(2<3)
12
13 #MAIOR OU IGUAL
14 print(2>=3)
15
16 #MENOR OU IGUAL
17 print(2<=3)
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER GITLENS
Python + - - - - -
PS C:\Users\Usuario> & C:\Users\Usuario\AppData\Local\Programs\Python\Python38\python.exe "f:/Pen drive/Developer/E-books/E-book-Python/Exemplos do E-book/Capítulo 2/operadores_de_comparação.py"
False
True
False
True
False
True
True
PS C:\Users\Usuario>
```

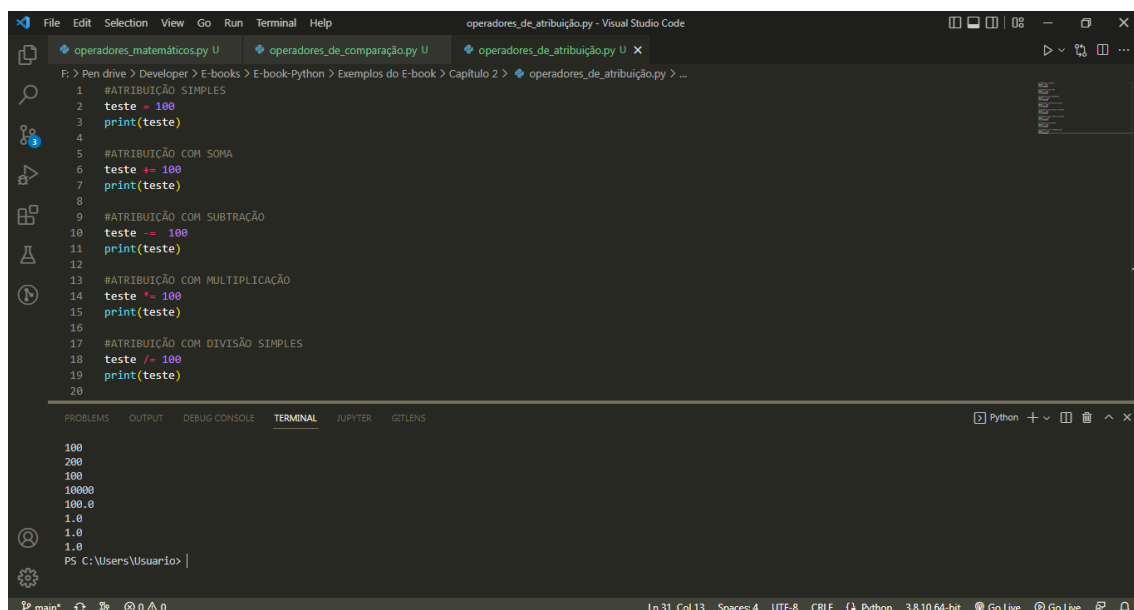
- Igualdade: retorna o valor booleano da igualdade. True se os números ou variáveis forem **iguais** e False se forem **diferentes**.
- Diferença: retorna o valor booleano da diferença. True se os números ou variáveis forem **diferentes** e False se forem **iguais**.
- Maior que: retorna o valor booleano da expressão. True se o primeiro número ou variável for **maior** que o segundo e False se for **menor**.
- Menor que: retorna o valor booleano da expressão. True se o primeiro número ou variável for **menor** que o segundo e False se for **maior**.
- Maior ou igual: retorna o valor booleano da expressão. True se o primeiro número ou variável for **maior ou igual** ao segundo e False se for **menor**.
- Menor ou igual: retorna o valor booleano da expressão. True se o primeiro número ou variável for **menor ou igual** ao segundo e False se for **maior**.

## 2.2.3 Operadores de atribuição

Utilizados para atribuir um valor inicial ou modificar o valor de uma variável.

Atribuição simples	=
Atribuição com soma	+=
Atribuição com subtração	-=
Atribuição com multiplicação	*=

Atribuição com divisão	/=
Atribuição com divisão inteira	//=
Atribuição com resto	%=
Atribuição com exponenciação	**=

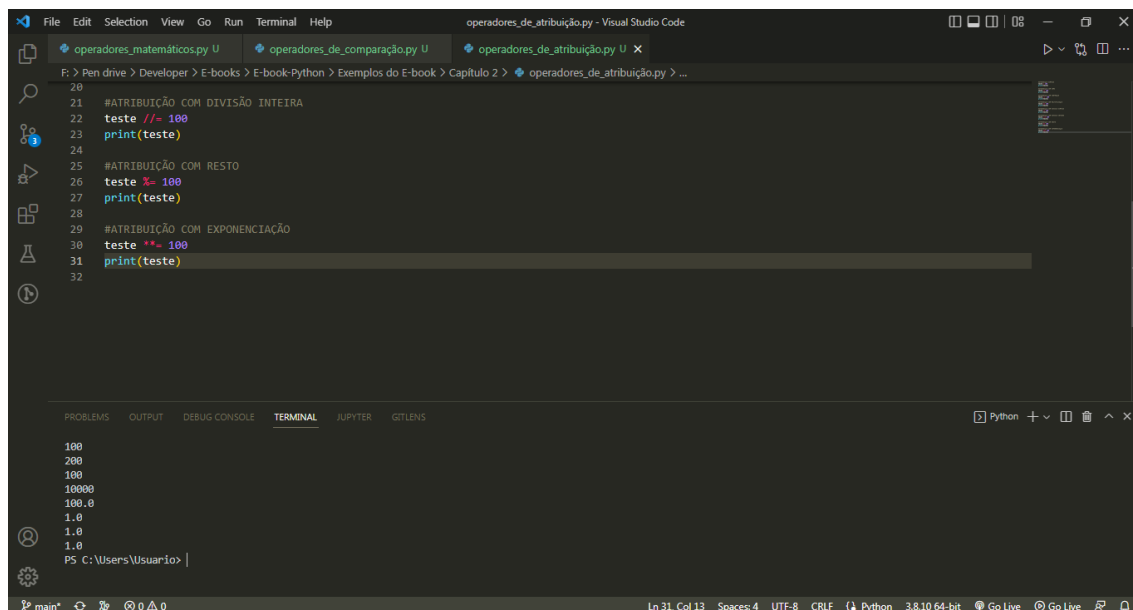


The screenshot shows a Visual Studio Code window with a Python file named `operadores_de_atribuição.py`. The code contains five test cases for assignment operators, each starting with a comment and a variable named `teste` initialized to 100. The operations performed are: simple assignment (`=`), addition (`+=`), subtraction (`-=`), multiplication (`*=`), and division (`/=`). The terminal output shows the results of these operations: 100, 200, 100, 10000, and 100.0.

```
1 #ATRIBUIÇÃO SIMPLES
2 teste = 100
3 print(teste)
4
5 #ATRIBUIÇÃO COM SOMA
6 teste += 100
7 print(teste)
8
9 #ATRIBUIÇÃO COM SUBTRAÇÃO
10 teste -= 100
11 print(teste)
12
13 #ATRIBUIÇÃO COM MULTIPLICAÇÃO
14 teste *= 100
15 print(teste)
16
17 #ATRIBUIÇÃO COM DIVISÃO SIMPLES
18 teste /= 100
19 print(teste)
20
```

Terminal Output:

```
100
200
100
10000
100.0
```



```
20
21 #ATRIBUIÇÃO COM DIVISÃO INTEIRA
22 teste //= 100
23 print(teste)
24
25 #ATRIBUIÇÃO COM RESTO
26 teste %= 100
27 print(teste)
28
29 #ATRIBUIÇÃO COM EXPONENCIAÇÃO
30 teste **= 100
31 print(teste)
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER GITLENS

Python + -

```
100
200
100
10000
100.0
1.0
1.0
1.0
PS C:\Users\Usuario>
```

Ln 31, Col 13 Spaces: 4 UTF-8 CRLF Python 3.8.10 64-bit Go Live Go Live

- Atribuição simples: atribui um valor à variável
- Atribuição com soma: atribui à variável o valor dela mesma somada ao valor à direita dos sinais
- Atribuição com subtração: atribui à variável o valor dela mesma subtraindo o valor à direita dos sinais
- Atribuição com multiplicação: atribui à variável o valor dela mesma multiplicando com o valor à direita dos sinais
- Atribuição com divisão simples: atribui à variável o valor dela mesma dividindo pelo valor à direita dos sinais retornando um float
- Atribuição com divisão inteira: atribui à variável o valor dela mesma dividindo pelo valor à direita dos sinais retornando um valor inteiro
- Atribuição com resto: atribui à variável o valor do resto dela mesma dividindo pelo valor à direita dos sinais
- Atribuição com exponenciação: atribui à variável o valor dela mesma elevado pelo valor à direita dos sinais retornando um valor inteiro

---

---

## Referências:

[https://diretoaponto-tech.com.br/a/Linguagem/Programa%C3%A7%C3%A3o/camelcase%20ou%20snake\\_case](https://diretoaponto-tech.com.br/a/Linguagem/Programa%C3%A7%C3%A3o/camelcase%20ou%20snake_case)

<https://wiki.python.org.br/PythonBrasil>