

Introdução à Programação em



<https://evandro-crr.github.io/intro-python>

O Interpretador do Python

- Não é necessário criar um arquivo para executar código Python.
- Podemos usar o interpretador diretamente.
- Abra um terminal e digite `python`

```
Python 3.12.4 (main, Jun 7 2024, 00:00:00) [GCC 14.1.1 20240607 (Red Hat 14.1.1-5)] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>>
```

Você deve ver uma mensagem parecida com essa.

- O prompt `>>>` indica que o interpretador está pronto para receber um comando.

Variáveis

Um nome que faz referência a um valor

```
>>> mensagem = "Olá! Estou usando Python!"
>>> print(mensagem)
Olá! Estou usando Python!
```

```
>>> mensagem = "Olá! Estou usando Python!"
>>> print(mensagem)
Olá! Estou usando Python!
>>> mensagem = "A mensagem foi substituída"
>>> print(mensagem)
A mensagem foi substituída
```

Nomes Válidos

- Deve conter apenas letras, números e underscore (`_`).
- Espaço não é permitido.
- Não pode começar com números.
- Não pode ser uma *palavra reservada*.

Strings

Uma sequência de caracteres

```
"Isso é uma string"  
'Isso também é uma string'
```

```
'Podemos definir uma string "assim"  
"Ou 'assim'"
```

Pode ser definida com

- aspas simples `'...'`; ou
- aspas duplas `"..."`.

A flexibilidade de usar aspas simples ou duplas permite usar aspas dentro da string.

Manipulação Simples de Strings

```
>>> nome = "Ada LOVELACE"  
>>> print(nome.title())  
Ada Lovelace  
>>> print(nome.upper())  
ADA LOVELACE  
>>> print(nome.lower())  
ada lovelace
```

- `title()` Muda a primeira letra para maiúscula e as demais para minúsculas.
- `upper()` Muda todas as letras para maiúsculas.
- `lower()` Muda todas as letras para minúsculas.

String com Variáveis: *f-string*

É possível construir strings usando o valor de variáveis.

```
>>> primeiro_nome = "João"
>>> sobrenome = "Silva"
>>> nome_completo = f"{primeiro_nome} {sobrenome}"
>>> mensagem = f"Olá, {nome_completo}!"
>>> print(mensagem.upper())
OLÁ, JOÃO SILVA!
```

- Para criar uma f-string é necessário usar o prefixo `f` antes das aspas.
- As variáveis devem estar dentro de chaves `{...}`.
- O prefixo *f* vem de format.

Caracteres Especiais

Quebra de Linha

`\n`

```
>>> print("Lista de tarefas:\n- Estudar Python")
Lista de tarefas:
- Estudar Python
```

```
>>> tarefa1 = "Estudar Python"
>>> tarefa2 = "Fazer a lista de algoritmos"
>>> lista = f"\t- {tarefa1}\n\t- {tarefa2}"
>>> print(f"Lista de tarefas:\n{lista}")
Lista de tarefas:
    - Estudar Python
    - Fazer a lista de algoritmos
```

Tab

`\t`

```
>>> print("Item\tValor")
Item      Valor
```

```
>>> item1 = "Torrada\tR$ 3,60"
>>> item2 = "Leite\tR$ 5,90"
>>> print(f"Item\tValor\n{item1}\n{item2}")
Item      Valor
Torrada   R$ 3,60
Leite     R$ 5,90
```

Removendo

Espaços

"Python " \neq " Python"

```
>>> nome = " Alise "  
>>> nome.rstrip()  
' Alise'  
>>> nome.lstrip()  
'Alise '  
>>> nome.strip()  
'Alise'
```

Prefixo e Sufixo

```
>>> url = "https://docs.python.org"  
>>> url.removeprefix("https://")  
'docs.python.org'
```

```
>>> arquivo = "funciona_final_final.py"  
>>> arquivo.removesuffix(".py")  
'funciona_final_final'
```


Exercícios 1/3

1. Use uma variável para representar o nome de uma pessoa e, em seguida, imprima o nome dessa pessoa em letras minúsculas, maiúsculas e com a primeira letra de cada palavra em maiúscula.
2. Encontre uma citação. Imprima a citação e o nome do autor. Sua saída deve se parecer com o seguinte, incluindo as aspas:

```
Albert Einstein uma vez disse  
"Uma pessoa que nunca cometeu um erro nunca tentou nada novo."
```

Represente o nome da pessoa usando uma variável. Em seguida, componha sua mensagem e a represente com uma nova variável.

Exercícios 2/3

3. Atribua à variável o valor `notas_de_python.txt` e use algum método de manipulação de string para extrair o nome base do arquivo (sem a extensão). Teste o código para outros nomes de arquivo com a mesma extensão.
4. Similar ao exercício 3, implemente um código que remova a extensão de um arquivo Python. Por exemplo, se o valor da variável for `meu_codigo.py`, deve imprimir `meu_codigo`.

Números Inteiros

Operadores

```
>>> 3 + 2
5
>>> 3 - 2
1
>>> 3 * 2
6
>>> -3 // 2
-2
>>> 3 % 2
1
>>> 3**2
9
```

- Soma `+`
- Subtração `-`
- Multiplicação `*`
- Divisão Inteira `//`
- Módulo `%`
- Exponenciação `**`

A divisão inteira e o módulo têm um comportamento diferente de outras linguagens como C++.

Números Reais




Ponto Flutuante

```
>>> 3.5 + 2.0
5.5
>>> 3.5 - 2.0
1.5
>>> 3.5 * 2.0
7.0
>>> 3.5 / 2.0
1.75
>>> -3.5 % 2.0
0.5
>>> 3.5**2.0
12.25
```



Operadores

- Soma +
- Subtração -
- Multiplicação *
- Divisão /
- Módulo %
- Exponenciação **

Precedência de Operadores

1. Parênteses (...)
2. Operador unário -
3. **, 
4. *, /, //, %, 
5. +, -, 

Resolva da

-  direita para a esquerda
-  esquerda para a direita

Resolva no papel! 

-5 + 3 - 2

10 / (2 * 5)

10 / 2 * 5

(8 + 2) * 3 ** 2 / 2

2 ** 3 ** 2

(2 ** 3) ** 2

Exercícios 3/3

5. Em um arquivo, escreva operações de adição, subtração, multiplicação e divisão que resultem no número 8. Certifique-se de colocar suas operações em chamadas `print()` para ver os resultados.
6. Atribua um número a uma variável. Em seguida, usando essa variável, crie uma mensagem usando f-string e imprima.

O Zen do Python, por Tim Peters

```
>>> 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!
```

Introdução à Programação em



<https://evandro-crr.github.io/intro-python>