



Conhecendo a base do python

Tipos em Python

Python possui diferentes tipos de dados para armazenar informações. Aqui estão os principais tipos e para que eles servem:

| Tipo de Dado | Classe em Python |
|--|---|
| Texto (palavras, frases) | <code>str</code> |
| Numérico (números) | <code>int</code> , <code>float</code> , <code>complex</code> |
| Sequência (listas ordenadas) | <code>list</code> , <code>tuple</code> , <code>range</code> |
| Mapa (chave → valor) | <code>dict</code> |
| Coleção (conjuntos de valores únicos) | <code>set</code> , <code>frozenset</code> |
| Booleano (Verdadeiro ou Falso) | <code>bool</code> |
| Binário (dados em formato binário) | <code>bytes</code> , <code>bytearray</code> , <code>memoryview</code> |



Números Inteiros

Números inteiros representados pela classe `int` e possuem precisão ilimitada. São exemplos válidos de números inteiros:

- 1, 10, 100, -1, -10, -100

Os números inteiros são aqueles que **não têm vírgula** e podem ser positivos, negativos ou zero.

📌 Eles são representados pela classe `int` e podem ser **bem grandes**, sem limite de tamanho!

📖 Imagine que você tem **moedas**:

- Se você tem **5 moedas**, é um número inteiro: `5`
- Se você deve **3 moedas**, isso é um número inteiro negativo: `-3`
- Se você não tem moedas, seu número é `0` !



Números de Ponto Flutuante (Float)

Os números de **ponto flutuante** (ou "**números com vírgula**") são usados para representar partes de um número, como **metades, décimos e centésimos**.

📌 Em Python, eles são representados pela classe `float`.



Imagine que você está cortando uma pizza:

🍕 Se você comeu metade da pizza, isso pode ser representado como `0.5`

🍕 Se você comeu um pedaço maior, tipo um terço, pode ser `0.33`



🍕 Se você perdeu 2.5 moedas, o número é `-2.5`



Booleano (Verdadeiro ou Falso)

Os **booleanos** são usados para dizer **se algo é verdadeiro ou falso**.

📌 Em Python, eles são representados pela classe `bool` e só existem dois valores:

-  `True` (Verdadeiro)
-  `False` (Falso)



Imagine um interruptor de luz:



Se a luz está ligada, é Verdadeiro (`True`)



Se a luz está apagada, é Falso (`False`)



Strings (Texto em Python)

As **strings** são usadas para representar palavras, frases ou qualquer sequência de letras, números e símbolos.

📌 Em Python, as strings são representadas pela classe `str` e **sempre ficam entre aspas!**



Imagine que você quer escrever algo em um caderno:

✍️ "Olá, mundo!" → Isso é uma string!

📞 "Telefone: 1234-5678" → Isso também é uma string!

🗣️ "Python é legal!" → Outra string!

Iniciando o Modo Interativo no Python

O **modo interativo** do Python permite **testar comandos rapidamente**, sem precisar salvar um arquivo antes.

📌 Existem **duas formas** de iniciar o modo interativo:

1 Abrindo diretamente o interpretador Python

Basta digitar o seguinte comando no terminal:

```
python
```

💡 Isso abrirá um ambiente onde você pode **escrever código e ver o resultado imediatamente!**

2 Executando um script e entrando no modo interativo

Se você tem um arquivo Python (`app.py`) e quer continuar testando código depois que ele for executado, use:

```
python -i app.p
```

💡 Isso **roda o código do arquivo** e depois te deixa no modo interativo para testar mais comandos!

✨ Agora tente abrir o modo interativo e testar seus primeiros comandos em Python! 🚀

`dir()` – Explorando Objetos em Python

O comando `dir()` ajuda a **descobrir o que existe dentro de um objeto** ou do próprio ambiente Python.

📌 **Como funciona?**

- 🔍 **Sem argumentos**, `dir()` mostra **tudo o que está disponível no seu código** naquele momento.
- 🔍 **Com um argumento**, `dir(objeto)` retorna **todos os atributos e métodos que o objeto possui**.



`help()` – Ajuda Integrada no Python

O comando `help()` é como um **manual de instruções** dentro do Python. Ele serve para aprender sobre **módulos, funções, classes, métodos e variáveis**.

📌 Como funciona?

- 📖 **Sem argumentos**, `help()` entra no **modo interativo**, onde você pode pesquisar qualquer coisa.
- 📖 **Com um argumento**, `help(objeto)` exibe uma explicação sobre o objeto escolhido.



Variáveis em Python

As **variáveis** são como **caixinhas** onde guardamos valores. Esses valores podem **mudar** durante a execução do programa.

📌 Por que usamos variáveis?

Imagine que você quer guardar **seu nome**, **sua idade** ou **um número** para usar depois no código. Em vez de escrever o valor diretamente, **guardamos ele em uma variável!**

📖 Exemplo:

```
nome = "Alice" # Guardando um nome
idade = 10     # Guardando a idade
pontos = 100   # Guardando uma pontuação
```



Alterando os Valores das Variáveis

📌 **No Python, não precisamos dizer qual é o tipo da variável.** Ele descobre automaticamente!

Mas isso também significa que podemos mudar o valor da variável a qualquer momento.

Exemplo:

```
idade = 10 # Criamos a variável 'idade' com valor 10
print(idade) # Vai mostrar: 10
```

```
idade = 15 # Agora alteramos o valor para 15
print(idade) # Vai mostrar: 15
```

💠 **Importante:**

- O **Python sempre precisa de um valor na variável**, ou seja, não podemos deixar ela vazia.
- Podemos mudar o tipo da variável sem problemas!

```
numero = 5 # Inteiro
print(numero) # 5
```

```
numero = "cinco" # Agora virou uma string
print(numero) # "cinco"
```

💡 **Dica:** Isso torna o Python **muito flexível!** Mas tome cuidado para não confundir os tipos de dados no código. 🚀

Constantes em Python

As **constantes** são parecidas com as **variáveis**, mas com uma diferença: **seu valor nunca muda** durante a execução do programa.

Diferente de algumas linguagens como **Java e C**, que possuem palavras-chave como `final` e `const`, **Python não tem um comando específico para criar constantes.**

Mas então, como criar uma constante?

Em Python, seguimos uma **convenção**:



◆ **Escrevemos o nome da constante todo em letras maiúsculas**, indicando que seu valor **não deve ser alterado**.

Exemplo de constante:

```
PI = 3.14159 # O valor de PI nunca muda
GRAVIDADE = 9.8 # A aceleração da gravidade na Terra
```

◆ Podemos usar essas constantes no código, mas **não devemos alterá-las**.

```
print(PI) # Vai mostrar: 3.14159
print(GRAVIDADE) # Vai mostrar: 9.8
```

 **Dica:** Mesmo que Python **permita mudar o valor**, boas práticas recomendam **NÃO alterar uma constante**, pois isso pode gerar confusão no código. 

Convertendo Tipos em Python

Às vezes, precisamos **mudar o tipo de uma variável** para conseguir manipulá-la de outra forma.

Por que converter tipos?

Imagine que você tem um número armazenado como **texto (str)**, mas precisa fazer um cálculo com ele. Para isso, precisamos **converter o tipo da variável**!

Exemplo prático:

```
idade = "25" # Essa idade está como texto (string)
nova_idade = int(idade) # Agora transformamos em número (int)



print(nova_idade + 5) # Agora podemos somar corretamente!
```

◆ **Principais funções para conversão:**

- `int()` → Converte para número inteiro
- `float()` → Converte para número decimal
- `str()` → Converte para texto (string)
- `bool()` → Converte para verdadeiro (`True`) ou falso (`False`)

Outro exemplo útil:

```
numero = 3.14
numero_texto = str(numero) # Agora o número virou um texto
print("O número é " + numero_texto) # Funciona sem erro!
```

 **Dica:** A conversão de tipos ajuda a evitar **erros** e a fazer cálculos corretamente no Python! 

Função `input()` – Capturando Dados do Usuário

A função `input()` permite que o usuário **digite informações** pelo teclado enquanto o programa está rodando.

Como funciona?

- **Exibe uma mensagem na tela** (se houver um argumento).
- **Lê o que o usuário digita** e retorna o valor como **texto** (`str`).

Exemplo:

```
nome = input("Qual é o seu nome? ") # Usuário digita o nome
print("Olá,", nome, "!") # Exibe a saudação com o nome digitado
```

Saída esperada:


```
Qual é o seu nome? Alice
Olá, Alice !
```

Convertendo Entrada do Usuário

Como o `input()` **sempre retorna um texto (`str`)**, se precisarmos de um número, devemos convertê-lo.

```
idade = input("Quantos anos você tem? ") # Entrada como string
idade = int(idade) # Convertendo para número inteiro

print("Daqui a 5 anos, você terá", idade + 5, "anos!")
```

 **Dica:** Sempre converta a entrada quando precisar trabalhar com **números**!



Função `print()` – Exibindo Informações na Tela

A função `print()` é usada para **mostrar mensagens** na tela. Ela pode exibir **textos, números e variáveis**, ajudando a visualizar informações no programa.

 **Como funciona?**

- Recebe um ou mais valores e os exibe na tela
- Por padrão, separa os valores com espaço (`sep`) e finaliza com uma quebra de linha (`end`)



Exemplo básico:

```
print("Olá, mundo!")
```

◆ **Saída esperada:**

```
Olá, mundo!
```



Exibindo múltiplos valores:

```
print("Nome:", "Alice", "Idade:", 25)
```

◆ **Saída esperada:**

Nome: Alice Idade: 25

Personalizando separadores (`sep`) e finais (`end`)

```
print("Python", "é", "divertido!", sep="-") # Sep substitui espaço por "-"
print("Essa linha", end=" ") # Não pula para a próxima linha
print("continua aqui.")
```

◆ Saída esperada:

Python-é-divertido!
Essa linha continua aqui.

💡 **Dica:** A função `print()` é muito útil para **debugar** e entender o que está acontecendo no código! 🚀

+ − O que são Operadores Aritméticos?

Os **operadores aritméticos** são símbolos usados para **fazer contas** em Python, como **adição, subtração, multiplicação e divisão**.

📌 Exemplos de operadores aritméticos:



| Operador | Nome | Exemplo | Resultado |
|-----------------|-----------------|---------------------|------------------|
| <code>+</code> | Adição | <code>5 + 3</code> | <code>8</code> |
| <code>-</code> | Subtração | <code>10 - 4</code> | <code>6</code> |
| <code>*</code> | Multiplicação | <code>6 * 2</code> | <code>12</code> |
| <code>/</code> | Divisão | <code>9 / 3</code> | <code>3.0</code> |
| <code>//</code> | Divisão inteira | <code>9 // 2</code> | <code>4</code> |
| <code>%</code> | Módulo (resto) | <code>10 % 3</code> | <code>1</code> |
| <code>**</code> | Exponenciação | <code>2 ** 3</code> | <code>8</code> |

Exemplo prático:

```
a = 10
b = 3

soma = a + b
multiplicacao = a * b
resto = a % b

print("Soma:", soma) # Mostra 13
print("Multiplicação:", multiplicacao) # Mostra 30
print("Resto da divisão:", resto) # Mostra 1
```

 **Dica:** Os operadores ajudam a fazer **cálculos rápidos e precisos** no Python!


Operadores de Comparação em Python

Os **operadores de comparação** são usados para **comparar dois valores** e retornar um **resultado booleano** (**True** ou **False**).

 **Principais operadores de comparação:**

| Operador | Significado | Exemplo | Retorno |
|--------------------|------------------|------------------------|-------------------|
| <code>==</code> | Igual a | <code>5 == 5</code> | <code>True</code> |
| <code>!=</code> | Diferente de | <code>5 != 3</code> | <code>True</code> |
| <code>></code> | Maior que | <code>10 > 5</code> | <code>True</code> |
| <code><</code> | Menor que | <code>2 < 8</code> | <code>True</code> |
| <code>>=</code> | Maior ou igual a | <code>7 >= 7</code> | <code>True</code> |
| <code><=</code> | Menor ou igual a | <code>4 <= 6</code> | <code>True</code> |

 **Exemplo prático:**

```
a = 10
b = 5
```

```
print(a == b) # False (10 não é igual a 5)
print(a != b) # True (10 é diferente de 5)
print(a > b) # True (10 é maior que 5)
print(a <= 10) # True (10 é menor ou igual a 10)
```

◆ **Os operadores de comparação são muito usados em decisões e condições, como em `if` statements.**

🎯 Exemplo em uma condição:

```
idade = 18

if idade >= 18:
    print("Você pode dirigir!")
else:
    print("Você ainda não pode dirigir.")
```

◆ **Saída esperada:**

Você pode dirigir!

💡 **Dica:** Comparações são essenciais para **verificar condições** e **tomar decisões no código!** 🚀

🔄 Operadores de Atribuição em Python

Os **operadores de atribuição** são usados para **atribuir valores a variáveis**.

📌 O operador mais comum é `=`, mas existem **outras formas** de atualizar valores de maneira mais eficiente!

| Operador | Exemplo | Equivalente a | Resultado |
|-----------------|---------------------|------------------------|--------------------------------------|
| <code>=</code> | <code>x = 5</code> | - | <code>x</code> recebe <code>5</code> |
| <code>+=</code> | <code>x += 3</code> | <code>x = x + 3</code> | Soma e atualiza <code>x</code> |
| <code>-=</code> | <code>x -= 2</code> | <code>x = x - 2</code> | Subtrai e atualiza <code>x</code> |
| <code>*=</code> | <code>x *= 4</code> | <code>x = x * 4</code> | Multiplica e atualiza <code>x</code> |
| <code>/=</code> | <code>x /= 2</code> | <code>x = x / 2</code> | Divide e atualiza <code>x</code> |



| | | | |
|------------------|----------------------|-------------------------|--|
| <code>//=</code> | <code>x //= 3</code> | <code>x = x // 3</code> | Divisão inteira e atualiza <code>x</code> |
| <code>%=</code> | <code>x %= 5</code> | <code>x = x % 5</code> | Resto da divisão e atualiza <code>x</code> |
| <code>**=</code> | <code>x **= 2</code> | <code>x = x ** 2</code> | Potência e atualiza <code>x</code> |

Exemplo prático:

```
x = 10
x += 5 # Equivalente a x = x + 5
print(x) # Resultado: 15

x *= 2 # Equivalente a x = x * 2
print(x) # Resultado: 30

x //= 3 # Equivalente a x = x // 3
print(x) # Resultado: 10
```

 **Dica:** Os operadores de atribuição são úteis para **atualizar valores de forma rápida e eficiente!** 

Operadores Lógicos em Python

Os **operadores lógicos** são usados para combinar **expressões booleanas** (`True` ou `False`).

Principais operadores lógicos:

| Operador | Significado | Exemplo | Resultado |
|------------------|---|-----------------------------|--------------------|
| <code>and</code> | E (Verdadeiro se ambas forem <code>True</code>) | <code>True and False</code> | <code>False</code> |
| <code>or</code> | OU (Verdadeiro se pelo menos uma for <code>True</code>) | <code>True or False</code> | <code>True</code> |
| <code>not</code> | NÃO (Inverte o valor booleano) | <code>not True</code> | <code>False</code> |

Exemplo prático:

```
a = True
b = False

print(a and b) # False (ambos precisam ser True)
print(a or b)  # True (basta um ser True)
print(not a)   # False (inverte o valor de a)
```

🎯 Exemplo em uma condição:

```
idade = 20
possui_cnh = True

if idade >= 18 and possui_cnh:
    print("Você pode dirigir!")
else:
    print("Você não pode dirigir.")
```

💎 Saída esperada:

Você pode dirigir!

💡 **Dica:** Os operadores lógicos ajudam a criar **condições mais complexas** no código! 🚀

ID Operadores de Identidade em Python

Os **operadores de identidade** são usados para verificar se **dois objetos ocupam o mesmo espaço na memória**. Eles não comparam os valores diretamente, mas sim **se as variáveis apontam para o mesmo objeto**.

📌 Principais operadores de identidade:

| Operador | Significado | Exemplo | Resultado |
|-----------------|---|---------------------|---|
| <code>is</code> | Verdadeiro se os objetos forem o mesmo | <code>a is b</code> | <code>True</code> se <code>a</code> e <code>b</code> forem o mesmo objeto |

| | | | |
|---------------------|---|-------------------------|---|
| <code>is not</code> | Verdadeiro se os objetos forem diferentes | <code>a is not b</code> | <code>True</code> se <code>a</code> e <code>b</code> forem objetos diferentes |
|---------------------|---|-------------------------|---|



Exemplo prático:

```
a = [1, 2, 3]
b = a # 'b' recebe a referência de 'a'
c = [1, 2, 3] # 'c' é uma nova lista, mas com os mesmos valores

print(a is b) # True (ambos apontam para o mesmo objeto)
print(a is c) # False (são listas diferentes na memória)
print(a == c) # True (os valores dentro das listas são iguais)
```

Explicação:

- `is` verifica **se são o mesmo objeto na memória**.
- `==` verifica **se os valores são iguais**.
- `is not` verifica **se os objetos são diferentes**.

 **Dica:** Use `is` apenas quando precisar verificar **se duas variáveis apontam para o mesmo objeto!** 

Operadores de Associação em Python

Os **operadores de associação** são usados para verificar **se um valor está presente dentro de uma sequência**, como listas, tuplas ou strings.

Principais operadores de associação:

| Operador | Significado | Exemplo | Resultado |
|---------------------|---|------------------------------------|-------------------|
| <code>in</code> | Verdadeiro se o valor estiver na sequência | <code>"a" in "banana"</code> | <code>True</code> |
| <code>not in</code> | Verdadeiro se o valor NÃO estiver na sequência | <code>5 not in [1, 2, 3, 4]</code> | <code>True</code> |

Exemplo prático:


```
frutas = ["maçã", "banana", "laranja"]

print("banana" in frutas) # True (está na lista)
print("uva" in frutas)    # False (não está na lista)
print("abacaxi" not in frutas) # True (não está na lista)
```

Exemplo com strings:

```
frase = "Python é incrível!"

print("Python" in frase) # True (a palavra "Python" está na frase)
print("Java" not in frase) # True (a palavra "Java" não está na frase)
```

 **Dica:** Os operadores de associação são muito úteis para **verificar rapidamente se um item existe em uma coleção!** 