PYTHON

do básico ao avançado

simples e direto

Índice

Introdução ao Python e sua Filosofia

Python é uma linguagem de programação de alto nível, interpretada e com uma sintaxe que prioriza a legibilidade do código. Criada por Guido van Rossum no final dos anos 80, seu nome foi inspirado no grupo de comédia britânico *Monty Python*, o que já revela seu espírito leve e criativo. Ao contrário de outras linguagens que colocam o desenvolvedor frente a regras rígidas, Python foi desenhada para ser elegante e simples, permitindo que você escreva menos código para fazer mais. Por isso, é uma das linguagens mais usadas no mundo, tanto por iniciantes quanto por engenheiros experientes.

Um dos pilares da linguagem é a filosofia expressa no documento "The Zen of Python", acessível digitando import this no terminal Python. Nele, estão princípios como: "Belo é melhor que feio", "Explícito é melhor que implícito" e "Simples é melhor que complexo". Essa filosofia orienta tanto o design da linguagem quanto o modo como os programadores Python pensam suas soluções. Ao aprender Python, você não está apenas aprendendo uma nova sintaxe, mas entrando em uma cultura que valoriza clareza, colaboração e eficiência.

Variáveis, Tipos de Dados e Convenções

Em Python, variáveis são nomes usados para armazenar dados que o programa pode manipular. Ao contrário de linguagens como Java ou C++, você não precisa declarar o tipo da variável antecipadamente. Python é uma linguagem de **tipagem dinâmica**, o que significa que o tipo do dado é inferido no momento da atribuição. Isso permite uma escrita mais rápida e fluida do código.

Os principais **tipos de dados** em Python são:

- `int`: números inteiros (ex: `5`, `-10`)
- `float`: números decimais (ex: `3.14`, `-0.5`)
- `str`: texto (ex: `"Olá, mundo!"`)
- `bool`: valores booleanos (`True` ou `False`)
- `None`: representa ausência de valor (semelhante a `null` em outras linguagens)

Quanto às **convenções de nomes**, recomenda-se usar nomes descritivos, em letras minúsculas e com palavras separadas por underscore (_), como `nome_usuario`, `quantidade_itens` ou `media_final`. Essa prática segue o padrão conhecido como *snake_case*, que facilita a leitura e manutenção do código. Evite nomes como `a`, `x1`, `temp`, a menos que o contexto seja óbvio.

Exemplo 1

Declarando variáveis com diferentes tipos

```python
# Atribuições simples com tipos diferentes
nome = "Rafael"
idade = 28
altura = 1.75
eh_estudante = True
saldo = None  # valor ainda não definido

# Exibindo os valores e seus tipos
print("Nome:", nome, "| Tipo:", type(nome))
print("Idade:", idade, "| Tipo:", type(idade))
print("Altura:", altura, "| Tipo:", type(altura))
print("É estudante?", eh_estudante, "| Tipo:", type(eh_estudante))
print("Saldo:", saldo, "| Tipo:", type(saldo))
```

Exemplo 2

Convenções e boas práticas

```python
# Ruim: nomes genéricos e sem significado
x = "João"
y = 90

# Bom: nomes descritivos e legíveis
nome_aluno = "João"
nota_final = 90

print(f"O aluno {nome_aluno} teve nota {nota_final}.")
```

Operadores Aritméticos, Lógicos e de Comparação

Operadores são símbolos especiais que indicam ao Python para realizar operações específicas com variáveis e valores. Eles são divididos em categorias, e as principais neste estágio são: **aritméticos**, **lógicos** e **de comparação**.

🔹 Operadores Aritméticos

Usados para realizar cálculos matemáticos:

- `+` adição
- `-` subtração
- `*` multiplicação
- `/` divisão
- `//` divisão inteira
- `%` módulo (resto da divisão)
- `**` exponenciação

🔸 Operadores de Comparação

Usados para comparar valores, sempre retornam `True` ou `False`:

- `==` igual
- `!=` diferente
- `>` maior que
- `<` menor que
- `>=` maior ou igual
- `<=` menor ou igual

🔹 Operadores Lógicos

Usados para combinar expressões booleanas:

- `and`: verdadeiro se ambas forem verdadeiras
- `or`: verdadeiro se ao menos uma for verdadeira
- `not`: inverte o valor lógico (True vira False e vice-versa)

Esses operadores são a base para decisões e cálculos dentro de qualquer programa. Combiná-los corretamente permite construir lógicas mais complexas e robustas.

Exemplo 1

Cálculos e verificações com operadores

```python
# Operadores aritméticos
a = 10
b = 3

print("Soma:", a + b)
print("Subtração:", a - b)
print("Multiplicação:", a * b)
print("Divisão:", a / b)
print("Divisão inteira:", a // b)
print("Módulo (resto):", a % b)
print("Exponenciação:", a ** b)

# Operadores de comparação
print("a é igual a b?", a == b)
print("a é maior que b?", a > b)
print("a é diferente de b?", a != b)
```

Exemplo 2

Lógica condicional com operadores lógicos

```python
idade = 20
possui_carteira = True

# Verifica se a pessoa pode dirigir
pode_dirigir = idade >= 18 and possui_carteira

print("Pode dirigir?", pode_dirigir)

# Negando uma condição
estudante = False
print("É estudante?", estudante)
print("Não é estudante?", not estudante)

# Uso de 'or'
tem_convite = False
pagou_entrada = True
pode_entrar = tem_convite or pagou_entrada
print("Pode entrar no evento?", pode_entrar)
```

Continuará....