AULA 3 — Variáveis na Programação com Python

Prof. Cloves Rocha · ADS & Ciência da Computação

Introdução clara e objetiva sobre o que são variáveis, por que importam e como usá-las em Python. Este conteúdo é pensado para estudantes iniciantes em lógica de programação — explicações didáticas, exemplos práticos e um desafio final para aplicar o aprendido.

O que é uma variável?

Definição

Uma variável é um **nome simbólico** que referencia um espaço de memória onde guardamos um valor. O valor pode mudar ao longo da execução do programa.

Componentes

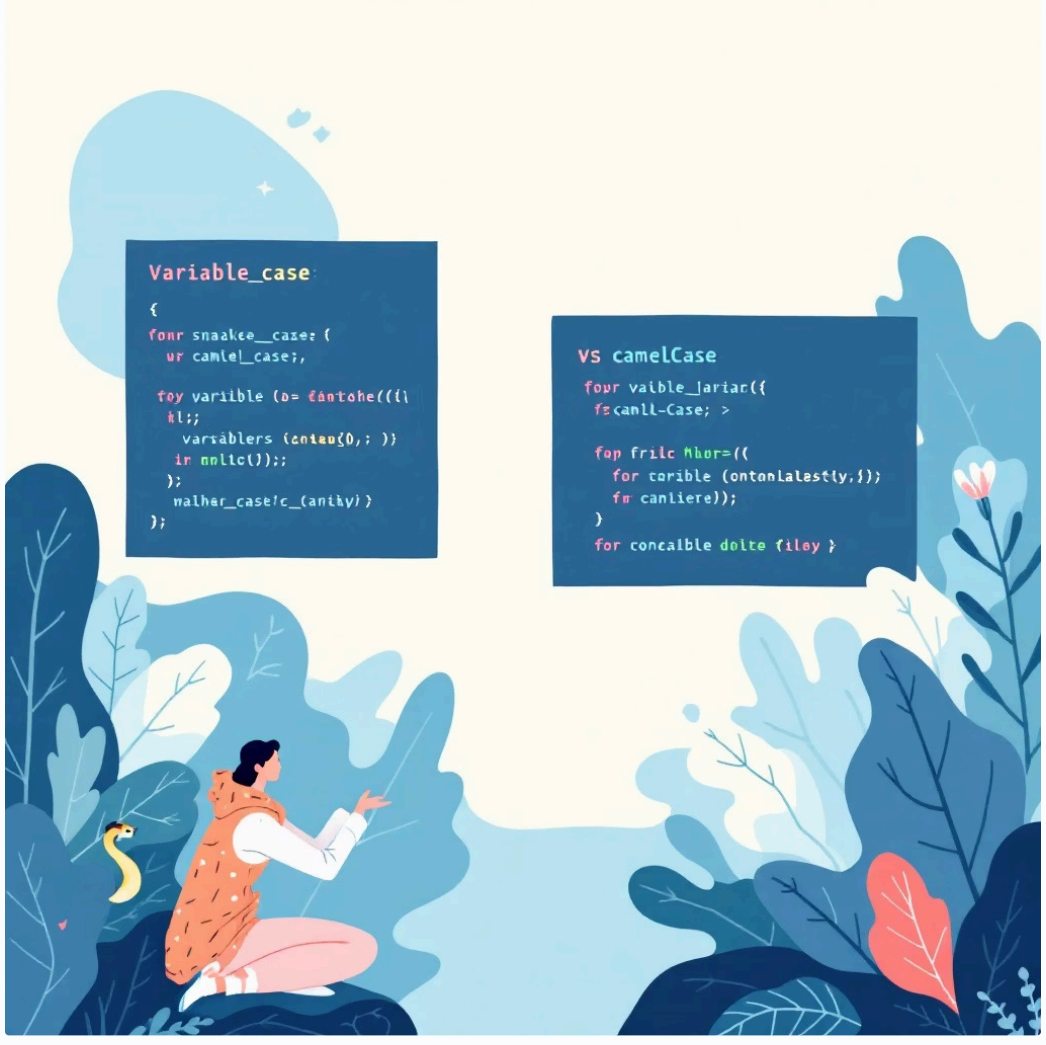
- **Nome**: identificador usado no código (ex.: idade).
- **Tipo**: define quais operações são válidas (ex.: inteiro, float, string, boolean).
- **Valor**: o dado atual armazenado (ex.: idade = 20).

Por que usar?

Permitem guardar e manipular informações, tornando programas flexíveis e reutilizáveis — essenciais para qualquer algoritmo.

Observação: em Python não precisamos declarar o tipo antes — a linguagem faz inferência automática (tipagem dinâmica).

Boas práticas ao nomear variáveis



Use snake_case

Em Python, prefira `minha_variavel` em vez de `minhaVariavel`.

Nomes significativos

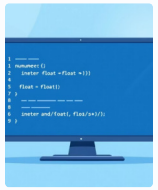
Prefira `preco_produto` a `pp`. Facilita leitura e manutenção.

Evite palavras reservadas

Não use nomes como `def`, `class`, `for`.

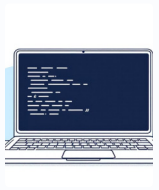
Comprimento razoável, legibilidade e consistência com o estilo Python (PEP8) são essenciais. Comentários curtos podem complementar nomes quando necessário.

Tipos de dados básicos em Python



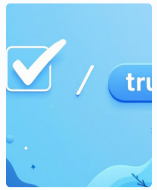
Numéricos

int (inteiros): `10` — float (reais): `3.14`. Usados em cálculos matemáticos.



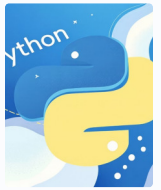
String

Sequência de caracteres: `'Olá'` ou `\"Mundo\"`. Serve para textos, nomes, mensagens.



Booleano

Valores `True` ou `False`. Úteis para decisões e condições.



Estruturas compostas (introdução)

Listas, tuplas e dicionários guardam múltiplos valores. Ex.: `[1, 2, 3]`.

Python converte tipos automaticamente em algumas operações (coerção), mas é recomendado compreender quando a conversão implícita pode gerar resultados inesperados.

Operadores Aritméticos — referência prática

+

Adição — soma dois valores. Ex.: `3 + 4 # resultado 7`

-

Subtração — diferença entre valores. Ex.: `10 - 2 # resultado 8`

*

Multiplicação — produto. Ex.: `5 * 6 # resultado 30`

/

Divisão — quociente (gera float). Ex.: `7 / 2 # resultado 3.5`

%

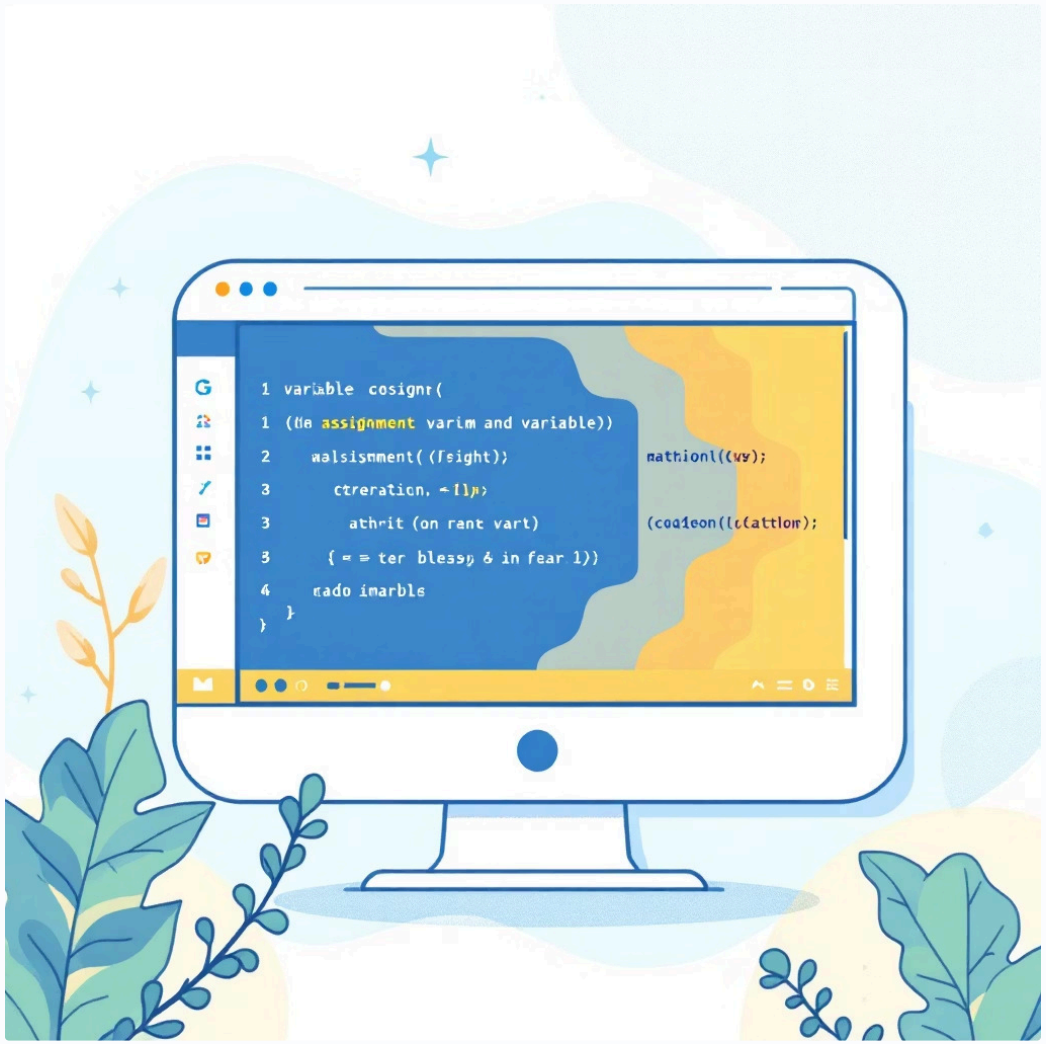
Módulo — resto da divisão inteira. Ex.: `7 % 2 # resultado 1`

// e ** (bônus)

`//` divisão inteira (floor): `7 // 2 # 3`. `**` exponenciação: `2 ** 3 # 8`.

Exemplos em Python: teste cada operação no REPL ou em um arquivo .py para observar comportamento com inteiros e floats. Atenção a divisão por zero — gera erro.

Exemplos práticos — pequenos trechos de código



Exemplo 1 — atribuição e operação:

```
a = 10
b = 3
soma = a + b
media = (a + b) / 2
print("Soma:", soma)
print("Média:", media)
```

Exemplo 2 — usando módulo e exponenciação:

```
n = 7
resto = n % 2
quad = n ** 2
print("Resto da divisão por 2:", resto)
print("Quadrado:", quad)
```

Teste variações: troque valores, veja tipos resultantes (int vs float) e pratique até entender como cada operador afeta o resultado.

Erros comuns e como evitá-los

1. Nome de variável inválido

Não comece com número, evite caracteres especiais e palavras reservadas. Ex.: `1a` é inválido.

2. Tipos incompatíveis

Tentar somar string com número: `'2' + 3` causa erro. Faça conversão explícita com `int()` ou `str()`.

3. Divisão por zero

Antes de dividir, verifique se o divisor é zero para evitar exceções (ZeroDivisionError).

4. Confundir atribuição e comparação

Em Python, `=` atribui, `==` compara. Use corretamente em condicionais.

Dica: pratique com pequenos testes, use mensagens de print para depurar valores das variáveis passo a passo.

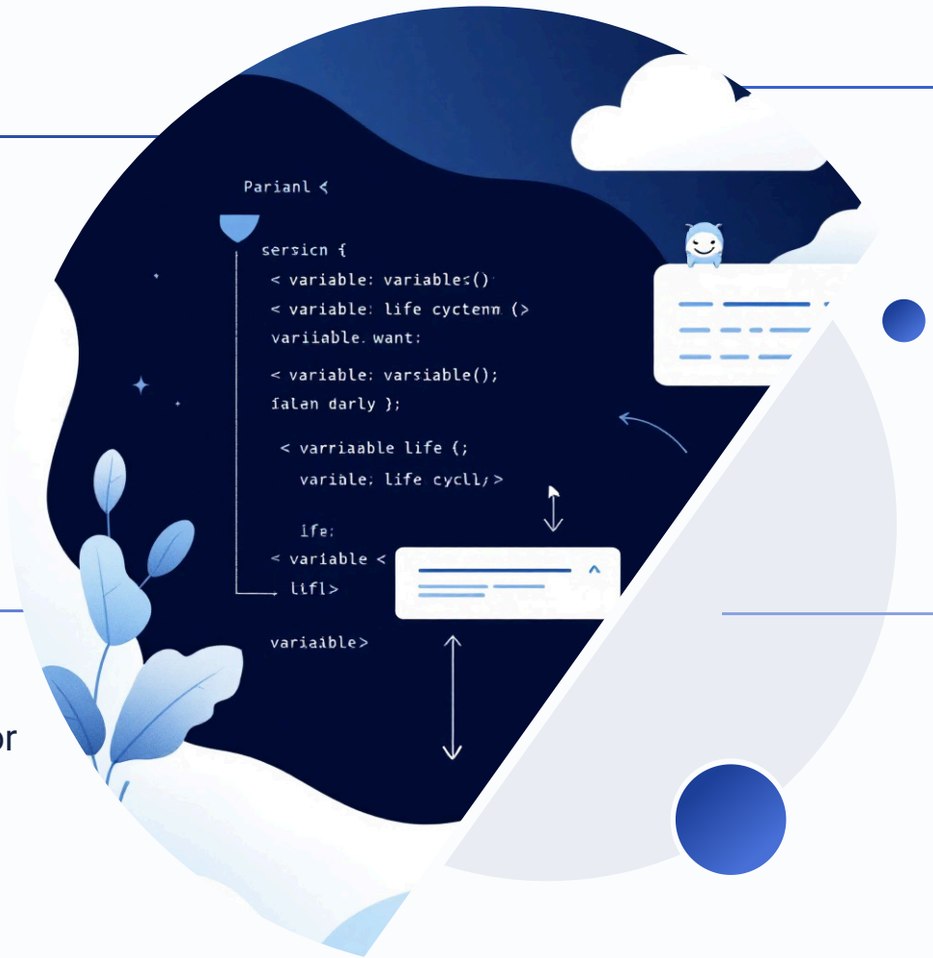
Diagrama: Ciclo de vida de uma variável durante a execução

Declaração

Criação e atribuição inicial
(ex.: idade = 20)

Uso

Operações com a variável
(ex.: calcular média)



Atualização

Atribuição de novo valor
(ex.: idade = idade + 1)

Leitura/Saída

Mostrar ou usar valor final
(ex.: print(idade))

O diagrama ilustra as quatro etapas principais: quando a variável é criada (atribuída pela primeira vez), usada em operações, atualizada com novos valores e finalmente lida/mostrada na saída. Entender esse fluxo ajuda a raciocinar sobre estado e efeitos colaterais.

Desafio 1 — Construindo uma calculadora simples em Python

Seu objetivo: implementar uma calculadora que suporte os operadores desta aula: `+`, `-`, `*`, `/`, `%`. Regras e orientações:

1. **Entrada**: peça ao usuário dois números e o operador (ex.: `input()`).
2. **Validação**: verifique entradas numéricas e trate divisão por zero.
3. **Saída**: mostre o resultado formatado e claro (ex.: `print(f"Resultado: {resultado}")`).
4. **Entrega**: individual ou em dupla — envie o arquivo .py na pasta `desafios` do repositório oficial da disciplina (**link do repositório**).
5. **Apresentação**: prepare uma breve demonstração na próxima aula explicando escolhas de código e tratamento de erros.

Dica de implementação (esqueleto):

```
a = float(input("Digite o primeiro número: "))
b = float(input("Digite o segundo número: "))
op = input("Operador (+, -, *, /, %): ")

if op == "+":
 resultado = a + b
elif op == "-":
 resultado = a - b
elif op == "*":
 resultado = a * b
elif op == "/":
 if b != 0:
 resultado = a / b
 else:
 resultado = None
 print("Erro: divisão por zero.")
elif op == "%":
 if b != 0:
 resultado = a % b
 else:
 resultado = None
 print("Erro: divisão por zero.")
else:
 resultado = None
 print("Operador inválido.")

if resultado is not None:
 print(f"Resultado: {resultado}")
```

Resumo, próximos passos e critérios de avaliação

Resumo

Variáveis armazenam valores mutáveis; tipos básicos: int, float, str, bool; operadores aritméticos permitem manipular números.

Próximos passos

Pratique com exercícios: operações compostas, conversões de tipo e tratamento de erro; explore listas e estruturas na próxima aula.

Avaliação do desafio

Serão avaliados: funcionamento correto, tratamento de casos válidos/ inválidos, legibilidade do código e clareza na apresentação.

Boa sorte! Faça commits frequentes no repositório, com mensagens claras. Traga dúvidas para a próxima aula — revisaremos soluções e pontos de melhoria.