

Departamento de Engenharia de Sistemas e Computação

Bases da Programação

Identificadores, dados numéricos, tipos e constantes

João Araujo

Algoritmos Computacionais I

Aula 05

Roteiro

- Identificadores e as regras do Python (Cap. 2.6)
- Dados numéricos: inteiros, grandes inteiros, ponto flutuante (2.7)
- Dados não numéricos: texto e lógicos
- Tipos de variáveis; tipagem dinâmica (2.8)
- Constantes (2.9)

Variáveis e identificadores

- **Variável:** nome que se refere a um valor guardado na memória
- **Identificador:** o nome escolhido pelo programador (de variáveis, funções, módulos...)

Regras em Python

- Começa com letra ou _; depois letras, dígitos ou _
- Sem espaços ou símbolos como -, $, @
- Diferencia maiúsculas de minúsculas: `nota` $\neq$ `Nota`
- Não pode ser uma *palavra reservada*: `if`, `for`, `while`, `def`, `True`...

Identificadores válidos ou não?

Identificador	Válido?	Motivo
`media_final`	sim	
`nota1`	sim	dígito não é o primeiro caractere
`1nota`	não	começa com dígito
`area do circulo`	não	espaços
`valor-total`	não	- é o operador de subtração
`for`	não	palavra reservada
`Nota` e `nota`	sim	são variáveis *diferentes*

Boa prática: nomes *descritivos* (`area_triang`, e não `x1`). Exercício 2.8: reescrever identificadores incorretos.

Atribuição

```
base = 5              # cria a variável base
altura = 6
area_triang = (base * altura) / 2
base = base + 1       # atualiza o valor
a, b = 3, 4           # atribuição múltipla
a, b = b, a           # troca de valores
```

- = não é igualdade matemática: é *atribuição* (calcula o lado direito e guarda no esquerdo)
- `base = base + 1` é válido: lê o valor antigo, soma, guarda o novo

Fonte: `code/Cap2/prog.2.5_area_triangulo_v3.py`

Dados numéricos

`int`

Inteiros: 42, -7, 0.
Sem limite fixo em Python!

`float`

Reais (ponto flutuante): `3.14`, `-0.5`, `1e-3`, `2.5e8`.

Operador	Operação	Exemplo
`+ - *`	soma, subtração, multiplicação	`3 * 4` → `12`
`/`	divisão real	`7 / 2` → `3.5`
`//`	divisão inteira	`7 // 2` → `3`
`%`	resto	`7 % 2` → `1`
`**`	potência	`2 ** 10` → `1024`

Grandes inteiros: o Universo em milímetros

```
# Tamanho do Universo observável em milímetros
tamanho_universo = 93000000000         # anos-luz
segundos_por_min = 60
segundos_por_h = 60 * segundos_por_min
segundos_por_dia = 24 * segundos_por_h
segundos_por_ano = 365 * segundos_por_dia
velocidade_luz_em_mms = 299792458000  # mm/s
tamanho_universo_mm = tamanho_universo * velocidade_luz_em_mms * segundos_por_ano
print("O Universo tem aproximadamente", tamanho_universo_mm, "milímetros.")
```

- O resultado tem 27 dígitos; em C ou Java isso daria *overflow*
- Variantes: googol $= 10^{100}$, googolplexo, folha de papel dobrada 107 vezes (Exercícios 2.11 a 2.15)

Fonte: `code/Cap2/prog.2.6_universo.py`

Ponto flutuante: cuidado com a precisão

```
print(0.1, "+", 0.2, "=", 0.1 + 0.2)
```

```
0.1 + 0.2 = 0.30000000000000004
```

- Já vimos na Aula 2: 0,1 não tem representação binária exata
- Para *mostrar* resultados use formatação: `"{:.2f}".format(x)` ou `f"{x:.2f}"`
- Compare `float` com tolerância, nunca com ==

Fonte: `code/Cap2/prog.2.7_erropf.py`

Dados não numéricos

`str` (texto)

Entre aspas simples ou duplas:
`'Olá'`, `"Python"`
`'42'` é texto, não número!

`bool` (lógico)

Só dois valores: `True` e `False`.
Resultado de comparações.

Também existe `complex`

Números complexos: `3 + 4j`. Útil em Engenharia Elétrica (impedâncias, fasores).

Comparações e valores lógicos

```
x = 1; y = 2; z = 1
print("x == y?", x == y)      # False
print("x == z?", x == z)      # True
print("x != z?", x != z)      # False
print("x < y?", x < y)        # True
print("y >= z?", y >= z)      # True
x = 2
print("1 < x < 3:", 1 < x < 3)   # True: comparação encadeada
```

- Operadores: ==, !=, <, <=, >, >=
- Atenção: = atribui; == compara!

Fonte: code/Cap2/prog.2.8_compara.py, prog.2.9_compara_v2.py

Tipos e tipagem dinâmica

```
x = 23
print(x, type(x))      # 23 <class 'int'>
x = 12.4
print(x, type(x))      # 12.4 <class 'float'>
x = "Teste"
print(x, type(x))      # Teste <class 'str'>
```

- Em Python **o tipo pertence ao valor**, não à variável: a mesma variável pode apontar para tipos diferentes
- Conversões explícitas: `int("42")`, `float("3.5")`, `str(7)`
- `type(x)` mostra o tipo; `int(3.9)` → 3 (trunca)

Fonte: `code/Cap2/prog.2.11_variaveis_dinamicas.py`

Constantes

- Valores que **não devem mudar** durante o programa (π, velocidade da luz, taxas)
- Python não tem constantes "de verdade"; por convenção usamos nomes em MAIÚSCULAS
- Vantagem: mudar o valor em um único lugar e deixar o código legível

Constantes: identidade de Euler

```
PI = 3.1415
E = 2.71828
I = 1j
print("0 =", E**(PI*I) + 1)
print("0 = {:.2f}".format(E**(PI*I) + 1))
```

```
0 = (4.490366412035485e-09+9.476672816471575e-05j)
0 = 0.00+0.00j
```

- O resultado deveria ser exatamente 0: o erro vem da aproximação de π e e
- O módulo `math` traz `math.pi` e `math.e` mais precisos (Aula 13; Exercício 2.16)

Fonte: `code/Cap2/prog.2.12_euler.py`, `prog.2.13_eulerformatado.py`

Para praticar: exercícios do livro

Site do livro – Capítulo 2

- Identificadores: **2.8**
- Programas com dados numéricos: **2.9** (área do quadrado), **2.10** (segundos desde a meia-noite), **2.11 a 2.15** (Universo, googol, folha dobrada)
- Constantes: **2.16 a 2.18** (valores de π e e, hipotenusa, esfera de raio 5)

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap2/`

Exercícios adicionais (sem solução): `https://www.araujo.eng.uerj.br/livro/extras/`

Resumo da aula

- Identificador: letras, dígitos e _; não pode começar com dígito nem ser palavra reservada
- Tipos básicos: `int`, `float`, `str`, `bool`, `complex`
- Python tem tipagem dinâmica e inteiros de tamanho ilimitado
- Constantes por convenção em MAIÚSCULAS
- Próxima aula: estrutura básica de um programa, entrada e saída