

Departamento de Engenharia de Sistemas e Computação

Parâmetros e Escopo

Passagem de parâmetros, variáveis globais e locais, vários retornos

---

João Araujo

Algoritmos Computacionais I

Aula 11

Roteiro

- Parâmetros posicionais, nomeados e com valor padrão
- Funções que retornam mais de um resultado (Cap. 4.4)
- Passagem de parâmetros: valor ou referência? (4.5)
- Variáveis globais e locais: escopo (4.6)

Parâmetros com valor padrão e nomeados

```python
def imprime_linha(n=5, carac='*'):
    print(carac * n)

imprime_linha()                 # *****
imprime_linha(3)                # ***
imprime_linha(3, '-')           # ---
imprime_linha(carac='@', n=2)   # @@
```

- **Valor padrão**: usado quando o argumento é omitido
- **Argumentos nomeados**: a ordem deixa de importar e o código fica mais claro
- Parâmetros com padrão devem vir depois dos obrigatórios

Fonte: `code/Cap4/prog.4.12_imprimepadrao.py`

Retornando mais de um resultado

```
import random as r

def gera_coordenadas_aleatorias():
    """Gera par de coordenadas aleatórias."""
    x = r.random()
    y = r.random()
    return x, y

x, y = gera_coordenadas_aleatorias()
print(x, y)
```

- `return x, y` devolve uma **tupla** (veremos na Aula 20)
- O chamador *desempacota*: `x, y = ...`
- Exemplos: quociente e resto; mínimo e máximo; π estimado e o erro

Fonte: `code/Cap4/prog.4.23_calcula_pi_main.py`

Calculando π por Monte Carlo

```
def coordenadas_dentro_circulo(x, y):
    """Testa se coordenadas estão dentro do círculo de raio 1."""
    return m.hypot(x, y) < 1

def calcula_pi(n):
    """Calcula pi e o erro associado a partir de n pontos."""
    conta_circulo = 0
    for i in range(n):
        x, y = gera_coordenadas_aleatorias()
        if coordenadas_dentro_circulo(x, y):
            conta_circulo += 1
    pi = 4 * conta_circulo / n
    erro = m.fabs(pi - m.pi)
    return pi, erro
```

Ideia: a fração de pontos aleatórios no quarto de círculo aproxima $\pi/4$. Cada função faz *uma coisa*. (Exercícios 4.12 e 5.6.)

Fonte: `code/Cap4/prog.4.23_calcula_pi_main.py`

O programa principal: a função `main`

```
def main():
    num = int(input('Quantos pontos devem ser sorteados? '))
    pi, erro = calcula_pi(num)
    print('Com', num, 'pontos, o valor de pi é', pi,
          'com erro', erro)

main()
```

- Convenção: concentrar a entrada/saída em `main()` e deixar o cálculo nas outras funções
- Funções "puras" (só calculam) são fáceis de testar e reutilizar

Passagem de parâmetros: o valor muda?

```
def foo(num):
    num = 0

n = 42
foo(n)
print('n =', n)
```

```
n = 42
```

- Dentro da função, `num` é uma variável local, só um novo nome para o mesmo valor
- Reatribuir `num` *não* altera `n` no programa chamador
- Pergunta clássica: passagem "por valor" ou "por referência"? O livro responde: **nenhuma das duas** (4.5.1)

Fonte: `code/Cap4/prog.4.18_parametros_num.py`

Olhando os endereços com `id()`

```
def foo(n):
    print('n - id:', id(n), 'valor:', n, 'em foo ')
    n = 0
    print('n - id:', id(n), 'valor:', n, 'em foo ')

n = 42
print('n - id:', id(n), 'valor:', n)
foo(n)
print('n - id:', id(n), 'valor:', n)
```

- Ao entrar, `n` da função e `n` global têm o mesmo `id`: apontam para o mesmo objeto 42
- Ao fazer `n = 0`, o nome local passa a apontar para *outro* objeto
- Números e strings são **imutáveis**; listas (Aula 16 em diante) são mutáveis e se comportam de outro jeito!

Fonte: `code/Cap4/prog.4.17_parametros.py`

Escopo: variáveis globais e locais

```
global_y = 1

def foo(par):           # par existe somente dentro de foo()
    local_y = 20
    print('local_y =', local_y)
    print('global_y =', global_y)   # lê a global
    for i in range(1, 2):
        print('i =', i)
    print('i =', i)

foo(global_y)
```

- **Local**: criada na função; só existe lá dentro (inclusive os parâmetros)
- **Global**: criada fora das funções; pode ser *lida* em qualquer lugar

Fonte: `code/Cap4/prog.4.19_escopo.py (Exercício 4.15)`

Sombreamento: o mesmo nome, variáveis diferentes

```
y = 1

def foo():
    y = 42           # cria uma local; não altera a global
    print('y =', y)
    print('x =', x)  # x é global, definida abaixo

x = 84
foo()
print('y =', y)
```

```
y = 42
x = 84
y = 1
```

Fonte: code/Cap4/prog.4.20_escopo1.py

Erros de escopo clássicos

```
y = 1

def foo():
    print('y =', y)    # UnboundLocalError!
    y = 42             # esta atribuição torna y LOCAL em todo foo

foo()
```

UnboundLocalError

Se a função *atribui* a y em qualquer ponto, y é local em *toda* a função, e usá-la antes de atribuir é um erro.

- Para modificar de fato uma global: `global y` (mas evite!)
- Melhor: passar dados por **parâmetros** e devolver com `return`

Fonte: `code/Cap4/prog.4.21_escopo1erro.py`, `prog.4.22_escopo2erro.py`

Boas práticas com funções

- Uma função **faz uma coisa** e tem um nome que diz qual
- Entradas via **parâmetros**, saída via `return`
- Evite variáveis globais mutáveis: dificultam entender e testar o programa
- Funções curtas (cabem na tela)
- Escreva uma docstring e teste a função com casos simples antes de usá-la

Atividade

Identifique as variáveis globais e locais do Programa 4.16 (Exercício 4.15) e explique cada uma em uma frase.

Para praticar: exercícios do livro

Site do livro – Capítulo 4

- Valores de retorno múltiplos: **4.12** (π com mais pontos), **4.13** (dado de 6 lados)
- Escopo: **4.15**
- Revisão de funções: **4.8 a 4.11**

Enunciados e soluções comentadas em
`https://www.araujo.eng.uerj.br/livro/exercicios/cap4/`

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

Resumo da aula

- Parâmetros podem ter valor padrão e ser passados por nome
- `return a, b` devolve duas coisas (tupla)
- Reatribuir um parâmetro não altera a variável do chamador
- Variável atribuída na função é local; prefira parâmetros e `return` a globais
- Próxima aula: organizando programas em módulos