


Departamento de Engenharia de Sistemas e Computação

# Subalgoritmos

## Funções, fluxo de execução e return

---

João Araujo

Algoritmos Computacionais I

Aula 10

# Roteiro

- Por que dividir um programa em partes? (Cap. 4)
- Fluxo de execução (4.1)
- Definindo e chamando funções (4.3)
- Funções sem retorno: procedimentos
- Funções com retorno: `return`
- Documentação com *docstrings*

# Dividir para conquistar

- Um problema grande vira **vários problemas pequenos**
- Cada parte é um **subalgoritmo**: em Python, uma função
- Benefícios:
  - **reuso**: escreve uma vez, usa várias
  - **legibilidade**: nomes explicam o que cada parte faz
  - **teste**: verifica-se cada função isoladamente

**Já usamos funções!**

`print()`, `input()`, `int()`, `range()`, `len()`, `type()`...
Agora vamos *criar* as nossas.

# Definindo e chamando uma função

```
def imprime_linha():
    print('**********')

imprime_linha()
imprime_linha()
```

```
**********
**********
```

- `def nome(parâmetros):` → define (apenas guarda; ainda não executa)
- `nome()` → chama (executa o corpo)
- O corpo é um bloco indentado

Fonte: `code/Cap4/prog.4.8_imprimeb.py`

# Fluxo de execução

```
def a():
    print("início de a")
    b()
    print("fim de a")

def b():
    print("em b")

print("começo")
a()
print("fim")
```

**Ordem de execução**

1. as definições `def` são lidas (nada executa)
2. `print("começo")`
3. chama `a()` → pula para o corpo
4. chama `b()` → pula de novo
5. `b` termina → volta a `a`
6. `a` termina → volta ao programa
7. `print("fim")`

# Funções com parâmetros

```
def imprime_linha(n):
    print('*' * n)

imprime_linha(5)
imprime_linha(20)
```

```
*****
********************
```

- **Parâmetro**: nome na definição (`n`); **argumento**: valor na chamada (5)
- `'*' * n` repete a string $n$ vezes
- Exercício 4.5: usar funções para desenhar outras figuras com asteriscos

Fonte: `code/Cap4/prog.4.9_imprimenb.py`

# Funções sem retorno (procedimentos)

```python
def conceito(nota):
    """Imprime o conceito correspondente à nota."""
    if nota >= 9:
        print("A")
    elif nota >= 7:
        print("B")
    elif nota >= 5:
        print("C")
    else:
        print("D")

conceito(8.5)
```

- A função faz algo (imprime) mas não devolve valor ao chamador
- Exercício 4.7 do livro (nota → conceito)

# Funções com retorno: `return`

```
def quadrado(x):
    return x * x

a = quadrado(7)
print(a)                    # 49
print(quadrado(3) + 1)      # 10
```

**`print` $\neq$ `return`**

`print` *mostra* na tela; `return` *devolve* o valor para quem chamou, que pode usá-lo em contas. Ao executar `return`, a função termina.

Exercício 4.8: função que recebe um inteiro e devolve seu quadrado.

# Funções matemáticas: conversão de temperaturas

```python
def fahrenheit(celsius):
    """Converte Celsius para Fahrenheit."""
    return 9 / 5 * celsius + 32

def celsius(fahrenheit):
    """Converte Fahrenheit para Celsius."""
    return 5 / 9 * (fahrenheit - 32)

print(fahrenheit(100))      # 212.0
print(celsius(212))         # 100.0
```

- Exercícios 4.9 e 4.10; a "ida e volta" é um bom teste: `celsius(fahrenheit(x))` deve devolver `x`

# Fatorial como função

```
def fatorial(n):
    """Calcula o fatorial de n."""
    fat = 1
    for i in range(1, n + 1):
        fat = fat * i
    return fat

n = int(input('Qual o número para calcular o fatorial? '))
print('O fatorial de', n, 'é', fatorial(n))
```

- Compare com o programa sem função (Aula 8): a lógica foi *empacotada* com um nome
- Com a função pronta, calcular $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ é trivial

Fonte: `code/Cap4/prog.4.3_func_fatorial2.py`

# Função com várias decisões e retorno

```python
import math

def distancia(x1, y1, x2, y2):
    """Distância entre os pontos (x1, y1) e (x2, y2)."""
    return math.sqrt((x2 - x1)**2 + (y2 - y1)**2)

print(distancia(0, 0, 3, 4))      # 5.0
```

- Exercício 4.11: implemente e teste com o triângulo 3–4–5
- Funções podem chamar outras funções (inclusive de módulos importados)

# Documentando: docstrings

```
def area_circulo(raio):
    """Devolve a área do círculo de raio dado.

    raio -- número não negativo, em qualquer unidade.
    """
    return 3.14159 * raio ** 2

help(area_circulo)
```

- A primeira string do corpo é a **docstring**: documenta *o que* a função faz
- `help()` a mostra; boa prática desde o Capítulo 9 do livro
- Nomes de funções: verbos em minúsculas com _ (`calcula_media`)

# Atividade: de um programa longo a funções

Reescreva o programa da Aula 9 (média com validação) em três funções:

1. `le_nota(i)`: pede a nota $i$ até ser válida e a devolve
2. `calcula_media(soma, n)`: devolve a média
3. `situacao(media)`: devolve `"Aprovado"`, `"Prova final"` ou `"Reprovado"`

**Critério de qualidade**

O programa principal deve ter menos de 10 linhas e ser *legível como uma receita*.

# Para praticar: exercícios do livro

## Site do livro – Capítulo 4

- Funções sem retorno: **4.5** (figura com asteriscos), **4.6** (árvore de Natal com `for`), **4.7** (conceito)
- Funções com retorno: **4.8** (quadrado), **4.9** (Fahrenheit), **4.10** (Celsius), **4.11** (distância)

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

- Função = bloco nomeado e reutilizável: `def nome(parâmetros):`
- Definir não executa; *chamar* executa e depois o fluxo retorna
- `return` devolve um valor; `print` apenas imprime
- Docstrings documentam a função
- Próxima aula: parâmetros, escopo e múltiplos retornos