Etapa 7 — Funções e Procedimentos (Functions and Procedures)

Trilha: Lógica de Programação

1. Objetivos

- Entender o conceito de **função (function)** e **procedimento (procedure)**.
- Aprender a definir assinaturas, parâmetros e retorno.
- Compreender **escopo (scope)** e diferenças entre **parâmetro por valor** e **por referência**.
- Ver boas práticas de modularização e como testar funções isoladamente.

2. Introdução

Modularizar um algoritmo significa **dividi-lo em blocos menores e reutilizáveis**. Funções e procedimentos permitem isolar responsabilidades, melhorar legibilidade, facilitar testes e evitar repetição de código (DRY — *Don't Repeat Yourself*).

- **Função (function):** bloco de código que recebe entradas (parâmetros) e **retorna** um valor.
- **Procedimento (procedure):** bloco de código que executa ações, **sem retorno** (equivalente a void).

3. Assinatura e Estrutura Básica (pseudocódigo)

Função (com retorno):

```
funcao nome_da_funcao(param1, param2, ...)
  // processamento
  retorna valor
fimfuncao
```

Procedimento (sem retorno):

procedimento nome_do_procedimento(param1, ...)

  // ações

fimprocedimento

4. Parâmetros: por valor vs por referência (concept)

- **Por valor:** a função recebe uma *cópia* do dado; alterações internas **não afetam** o valor original.
- **Por referência:** a função recebe uma *referência* ao dado; alterações internas **modificam** o valor original.

> Observação: em pseudocódigo você pode indicar por referência com `&` ou anotação `por_ref`. Ao implementar em uma linguagem real, verifique como a linguagem trata passagem de parâmetros.

5. Escopo de Variáveis (Scope)

- **Local:** visível apenas dentro da função/procedimento; recomendado para evitar efeitos colaterais.
- **Global:** visível em todo o programa; use com cuidado (pode causar dependências implícitas).

Boas práticas: prefira variáveis locais e passe dados explicitamente via parâmetros.

6. Boas práticas de design de funções

- **Uma função = uma responsabilidade (single responsibility).**
- Nome descritivo (verbo + substantivo): `calcular_media`, `buscar_usuario`.
- Limitar número de parâmetros (preferir uma estrutura/objeto se precisar de muitos).
- Validar parâmetros no início da função.
- Documentar entradas, saída e efeitos colaterais.

7. Tratamento de erros dentro de funções

- Verificar valores inválidos logo no começo e retornar códigos de erro ou mensagens claras.
- Evitar que a função cause efeitos colaterais inesperados em variáveis externas.

Exemplo (divisão com validação):

```
funcao dividir(a, b)
  se b = 0 entao
    retorna "ERRO: divisao por zero"
  fimse
  retorna a / b
fimfuncao
```

8. Exemplo Prático 1 — Função soma

```
funcao soma(a, b)
  resultado ← a + b
  retorna resultado
fimfuncao
```

Uso:

```
x ← soma(3, 5)
escreva x   // 8
```

9. Exemplo Prático 2 — Procedimento de exibição

```
procedimento mostrar_boas_vindas(nome)
  escreva "Olá, ", nome, "!"
```

```
fimprocedimento
```

Uso:

```
mostrar_boas_vindas("Ana")
```

10. Exemplo Prático 3 — Fatorial (iterativo)

```
funcao fatorial_iterativo(n)
  se n < 0 entao
    retorna "ERRO: n negativo"
  fimse
  resultado ← 1
  para i de 1 ate n faca
    resultado ← resultado * i
  fimpara
  retorna resultado
fimfuncao
```

11. Exemplo Prático 4 — Fatorial (recursivo)

```
funcao fatorial_recursivo(n)
  se n < 0 entao
    retorna "ERRO: n negativo"
  fimse
  se n = 0 entao
    retorna 1
  fimse
```

```
  retorna n * fatorial_recursivo(n - 1)
fimfuncao
```

> **Atenção:** recursão requer caso base e pode causar *stack overflow* se n for muito grande.

12. Exemplo Prático 5 — Verificar número primo

```
funcao eh_primo(n)
  se n < 2 entao
    retorna falso
  fimse
  para i de 2 ate raiz(n) faca
    se n % i = 0 entao
      retorna falso
    fimse
  fimpara
  retorna verdadeiro
fimfuncao
```

> Observação: `raiz(n)` significa a parte inteira da raiz quadrada de `n`.

13. Exemplo Prático 6 — Parâmetro por referência (exemplo conceitual)

```
procedimento incrementa_em_um(&x)
  x ← x + 1
fimprocedimento

// Uso:
```

a ← 5

incrementa_em_um(a)

// Se passagem por referência: a agora = 6

14. Testes Unitários Básicos (Concept)

- Teste cada função isoladamente com:
    - Casos normais (valores esperados)
    - Casos limites (0, 1, valores extremos)
    - Casos de erro (entradas inválidas)

Exemplo (para soma):

- `soma(2, 3) → 5`
- `soma(0, 0) → 0`
- `soma(-1, 1) → 0`

Documentar estes testes facilita manutenção e validação.

15. Exercícios sugeridos

1. Escreva uma função `media` que receba 3 notas e retorne a média.
2. Crie um procedimento `imprimir_relatorio(nome, media)` que mostre mensagem formatada.
3. Implemente `fatorial_iterativo` e `fatorial_recursivo` e compare resultados e limitações.
4. Escreva `eh_primo` e teste para valores: 1, 2, 17, 18, 19.

5. Faça uma função `trocar(&a, &b)` que troque os valores usando passagem por referência.

16. Erros comuns e armadilhas

- Não validar parâmetros (divisão por zero, índices inválidos).
- Fazer funções com muitas responsabilidades (difícil de testar).
- Misturar lógica de apresentação (I/O) com lógica de negócio — prefira separar.
- Uso indiscriminado de variáveis globais causando efeitos colaterais.

17. Conclusão

Funções e procedimentos são **ferramentas essenciais** para escrever código limpo, modular e testável. Eles ajudam a:

- Reduzir duplicação de código;
- Melhorar legibilidade;
- Facilitar testes (unitários);
- Tornar o desenvolvimento colaborativo mais organizado.

Domine o design de funções simples, a validação de parâmetros e o escopo de variáveis antes de avançar para recursão (Etapa 8).

18. Perguntas de Múltipla Escolha)

1) O que melhor define uma função (function)?
a) Um bloco que só exibe mensagens.
b) Um bloco que recebe parâmetros e retorna um valor.
c) Um bloco que não pode ser testado isoladamente.
d) Uma variável global.

2) Qual a vantagem principal da modularização?
a) Aumentar o tamanho do código.
b) Reutilização e melhor organização.
c) Tornar o código menos legível.
d) Eliminar a necessidade de testes.

3) Em qual situação devemos usar passagem por referência?
a) Quando queremos que a função modifique o valor original.
b) Quando queremos garantir que o original não mude.
c) Quando não há parâmetros.
d) Quando o valor é uma string.

4) Qual é o caso base na recursão de fatorial?
a) n = 1
b) n = 0 (certo)
c) n = -1
d) n = 2

5) O que é uma boa prática ao criar funções?
a) Tornar a função responsável por muitas tarefas.
b) Nomear funções com palavras vagas.
c) Fazer cada função ter uma única responsabilidade.
d) Evitar validação de parâmetros.

19. GABARITO

1 → b
2 → b
3 → a
4 → b
5 → c