Etapa 7 — Funções e Procedimentos (Functions and Procedures)

Trilha: Lógica de Programação

Por que modularizar?

- Melhora **leitura** e **manutenção** do código
- Permite **reuso** de lógica (reuse)
- Facilita **testes** e depuração (debugging)
- Reduz duplicação de código

Conceitos principais

- **Função (Function):** bloco com entrada (parameters) e retorno (return)
- **Procedimento (Procedure):** bloco que executa ação, sem retorno (void)
- **Assinatura (signature):** nome + parâmetros esperados

Estrutura básica (pseudocódigo)

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

ou

```
procedimento nome_procedimento(param1)
    // ações
fimprocedimento
```

Exemplo: função soma (com retorno)

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

- **Uso:**

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

Exemplo: procedimento de exibição (sem retorno)

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

- **Uso:**

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

Parâmetros: por valor vs por referência (concept)

- **Por valor**: cópia do dado; mudanças internas não afetam o original
- **Por referência**: função recebe referência; alterações afetam o dado original.

Escopo de variáveis (Scope)

- **Local:** variável dentro da função (visível só ali)
- **Global:** variável acessível em todo o programa (usar com cuidado)
- Boas práticas: prefira variáveis locais

Boas práticas de design de função

- Função com **uma única responsabilidade (single responsibility)**
- Nomes claros e verbos descritivos (calcular_media, buscar_item)
- Limitar número de parâmetros (preferir objetos/estruturas se necessário)
- Documentar entradas/saídas

Tratamento de erros dentro da função

- Validar parâmetros no início
- Retornar códigos de erro ou lançar exceção conceitual
- Evitar efeitos colaterais inesperados
- Exemplo:

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

Exemplos práticos compostos

- Função para fatorial (iterativa/recursiva)
- Função para verificar número primo
- Procedimento para imprimir relatório

Exemplo: fatorial (iterativo)

```
funcao fatorial(n)
    resultado ← 1
    para i de 1 ate n faca
        resultado ← resultado * i
    fimpara
    retorna resultado
fimfuncao
```

Exemplo: fatorial (recursivo)

```
funcao fatorial(n)
    se n = 0 entao
        retorna 1
    fimse
    retorna n * fatorial(n - 1)
fimfuncao
```

Testes unitários básicos (concept)

- Teste funções isoladamente com casos: entradas típicas, limites, erros.
- Exemplo: soma(2,3) = 5, soma(0,0) = 0
- Documente casos de teste simples no comentário

Vantagens práticas da modularização

- Facilita **reutilização** entre projetos
- Permite **colaboração** (vários devs trabalhando em funções diferentes)
- Melhora **legibilidade** e acelera manutenção

Dica final / checklist antes de criar uma função

- A função tem responsabilidade única?
- O nome é descritivo?
- Parâmetros mínimos e claros?
- Validações de entrada?
- Documentação/simples comentário?

Conclusão

- Funções e procedimentos permitem **modularizar** o **algoritmo**, dividindo o código em partes menores e reutilizáveis.
- Melhoram **organização**, **legibilidade** e **manutenção** do código.
- **Funções (Functions)** retornam valores; **Procedimentos (Procedures)** executam ações.
- Boas práticas incluem:
  - Uma função = uma responsabilidade (single responsibility).
  - Nomes claros e verbos descritivos.
  - Validação de parâmetros e uso de variáveis locais.
- O uso correto dessas estruturas é essencial para a **recursão** e **programação estruturada**.
- **Resumo:**

“Modularizar é pensar em partes simples que, juntas, resolvem problemas complexos.”