lista-01-IF-DEF

February 27, 2026

1 Lista de Exercícios — Estruturas Condicionais + Funções (Python)

1.1 Preparação para Estrutura de Dados

Objetivo: praticar decisões (`if/elif/else`) e criação de funções, desenvolvendo base para busca, ordenação, validação, filas e árvores.

1.1.1 Regras

- Em cada exercício, **implemente a função** no bloco indicado.
- Depois, execute os **testes** logo abaixo.
- Não use `input()` nesta lista (foco em funções reutilizáveis).

  Dica: `pass` é um "placeholder" (não faz nada) e está ali para o Python aceitar a função enquanto você ainda não escreveu a lógica. ## Exercício 1 — Classificação de Número

Crie uma função que receba um número inteiro e retorne:

- `"positivo"`
- `"negativo"`
- `"zero"` def classificar_numero(n): # escreva sua solução aqui pass ### Testes print(classificar_numero(10)) print(classificar_numero(-5)) print(classificar_numero(0)) ## Exercício 2 — Validação de Valor

Crie uma função que receba um número e retorne:

- `"válido"` → se estiver entre 0 e 100 (inclusive)
- `"inválido"` → caso contrário def validar_valor(n): pass ### Testes print(validar_valor(50)) print(validar_valor(100)) print(validar_valor(-1)) print(validar_valor(150)) ## Exercício 3 — Par ou Ímpar

Crie uma função que receba um número inteiro e retorne `"par"` ou `"ímpar"`. def par_ou_impar(n): pass ### Testes print(par_ou_impar(4)) print(par_ou_impar(7)) ## Exercício 4 — Comparador (2 valores)

Crie uma função que receba dois números e retorne:

- `"maior"` → se o primeiro for maior
- `"menor"` → se o primeiro for menor
- `"iguais"` → se forem iguais def comparar(a, b): pass ### Testes print(comparar(5, 3)) print(comparar(2, 8)) print(comparar(4, 4)) ## Exercício 5 — Pode ser Chave?

Crie uma função que receba um número e retorne:

- `True` → se for **positivo**
- `False` → se for **zero ou negativo**

  Ideia: nem todo valor é uma boa "chave" em estruturas (validação!). def pode_ser_chave(n): pass ### Testes print(pode_ser_chave(10)) print(pode_ser_chave(0)) print(pode_ser_chave(-3)) ## Exercício 6 — Categoria de Idade

Crie uma função que receba uma idade e retorne:

- `"criança"` → 0 a 12
- `"adolescente"` → 13 a 17
- `"adulto"` → 18 a 59
- `"idoso"` → 60+ def categoria_idade(idade): pass ### Testes print(categoria_idade(10)) print(categoria_idade(16)) print(categoria_idade(30)) print(categoria_idade(70)) ## Exercício 7 — Tipo de Elemento (faixas)

Crie uma função que classifique um número:

- `"pequeno"` → até 10
- `"médio"` → 11 a 100
- `"grande"` → acima de 100

  Isso prepara para "bucketização" (dividir dados por faixas). def tipo_elemento(n): pass ### Testes print(tipo_elemento(5)) print(tipo_elemento(50)) print(tipo_elemento(200)) ## Exercício 8 — Prioridade (regras)

Crie uma função que receba:

- `idade` (int)
- `possui_deficiencia` (bool)

Retorne: - `"prioridade alta"` se idade 60 **ou** possui deficiência - `"prioridade normal"` caso contrário

  Base para filas e filas de prioridade. def prioridade(idade, possui_deficiencia): pass ### Testes print(prioridade(70, False)) print(prioridade(30, True)) print(prioridade(25, False)) ## Exercício 9 — Maior de 3

Crie uma função que receba 3 números e retorne o **maior**. def maior_de_tres(a, b, c): pass ### Testes print(maior_de_tres(3, 7, 5)) print(maior_de_tres(-1, -2, -3)) ## Exercício 10 — Status de Elemento (comparação com limite)

Crie uma função que receba `valor` e `limite` e retorne:

- `"baixo"` → valor $<$ limite
- `"ok"` → valor $==$ limite
- `"alto"` → valor $>$ limite def status(valor, limite): pass ### Testes print(status(10, 20)) print(status(20, 20)) print(status(30, 20)) ## Exercício 11 — Acesso (duas condições)

Crie uma função que receba:

- `idade`
- `possui_cadastro`

Retorne: - `"acesso permitido"` se idade 18 **e** possui cadastro - `"acesso restrito"` caso contrário def acesso(idade, possui_cadastro): pass ### Testes print(acesso(20, True)) print(acesso(15, True)) print(acesso(25, False)) ## Exercício 12 — Simulação de Inserção (atualizar ou manter)

Crie uma função que receba `valor_atual` e `valor_novo` e retorne:

- `"substituir"` → se `valor_novo` for maior
- `"manter"` → caso contrário

  Ideia: atualização de informação (muito comum em estruturas). def inserir(valor_atual, valor_novo): pass ### Testes print(inserir(10, 20)) print(inserir(30, 10)) print(inserir(15, 15)) ## Exercício 13 — Balanceamento simples (pré-AVL)

Crie uma função que receba `esquerda` e `direita` (alturas) e retorne:

- `"balanceado"` → se a diferença absoluta 1
- `"desbalanceado"` → caso contrário def balanceamento(esquerda, direita): pass ### Testes print(balanceamento(5, 4)) print(balanceamento(10, 3)) print(balanceamento(7, 8)) ## Exercício 14 — Prioridade numérica (3 faixas)

Crie uma função que receba um número (0 a 100) e retorne:

- `"alta"` → 80
- `"média"` → 50 a 79
- `"baixa"` → < 50 def prioridade_numerica(n): pass ### Testes print(prioridade_numerica(90)) print(prioridade_numerica(60)) print(prioridade_numerica(30)) ## Exercício 15 — Verificação de Conflito (duplicidade)

Crie uma função que receba `a` e `b` e retorne:

- `"conflito"` → se forem iguais
- `"ok"` → se forem diferentes

  Base para evitar duplicidades (sets/dicionários). def verificar_conflito(a, b): pass ### Testes print(verificar_conflito(10, 10)) print(verificar_conflito(10, 20))