Lista de Exercícios de Revisão

Dicionários, Conjuntos, Estruturas Compostas e Exceções

1) Cadastre `n` produtos em um dicionário, onde a **chave** é o nome do produto (`str`) e o **valor** é o preço (`float`). Depois, peça um nome de produto e exiba o preço correspondente; se não existir, mostre "`Produto não encontrado`".

Execução simulada 1:

```
Quantos produtos? 3
Produto 1 - nome: arroz
Produto 1 - preço: 5.2
Produto 2 - nome: feijao
Produto 2 - preço: 8.5
Produto 3 - nome: cafe
Produto 3 - preço: 9.9

Consultar produto: feijao
Preço: R$ 8.50
```

Execução simulada 2:

```
Quantos produtos? 2
Produto 1 - nome: leite
Produto 1 - preço: 4.7
Produto 2 - nome: pao
Produto 2 - preço: 1.2

Consultar produto: manteiga
Produto não encontrado
```

2) Cadastre `m` alunos em um dicionário `{nome: nota}` (nome `str` e nota `float`). Em seguida:

- exiba o **aluno com maior nota** e **menor nota** (nome e nota),
- exiba a **média da turma** com 2 casas,
- peça o **nome** de um aluno para consulta e mostre a nota; se não existir, "`Aluno não encontrado`".

Execução simulada 1:

```
Quantos alunos? 3
Aluno 1 - nome: Ana
Aluno 1 - nota: 8.5
Aluno 2 - nome: Bruno
Aluno 2 - nota: 6.0
Aluno 3 - nome: Carla
Aluno 3 - nota: 9.2

Maior: Carla (9.2)
Menor: Bruno (6.0)
Média: 7.90

Consultar aluno: Ana
Nota: 8.5
```

Execução simulada 2:

```
Consultar aluno: Pedro
Aluno não encontrado
```

3) Leia **duas linhas** de nomes (separados por espaço). Mostre:

- o **conjunto união**,
- na linha seguinte, a **quantidade total** de nomes distintos.

Execução simulada:

```
Nomes A: Ana João Maria
Nomes B: João Pedro
União: {Ana, João, Maria, Pedro}
Total: 4
```

4) Leia **duas linhas** com palavras-chave. Exiba:

- o **conjunto de comuns** (interseção),
- o **conjunto de exclusivas da primeira** (A−B),
- o **conjunto de exclusivas da segunda** (B−A).
  Se algum conjunto for vazio, mostre `{}`.

Execução simulada:

```
Linha 1: dados ciencia python limpeza
Linha 2: estatistica python dados regressao
Comuns: {python, dados}
Exclusivas da 1: {ciencia, limpeza}
Exclusivas da 2: {estatistica, regressao}
```

5) Leia `k` registros no formato `{"nome": str, "notas": [float, float, float]}` (pergunte 3 notas para cada aluno). Para cada aluno, exiba **nome** e **média** (1 casa). Ao final, exiba a **média da turma** (1 casa).

Execução simulada:

```
Quantos alunos? 2
Aluno 1 - nome: Ana
Nota 1: 8.0
Nota 2: 7.5
Nota 3: 9.0
```

```
Aluno 2 - nome: Bruno
Nota 1: 6.0
Nota 2: 6.5
Nota 3: 7.0

Ana: 8.2
Bruno: 6.5
Média da turma: 7.4
```

6) Leia uma **lista de nomes** (quantidade `q`). Monte internamente um dicionário `{inicial: [nomes...]}` usando a **primeira letra** de cada nome (considere maiúsculas). Depois:

- exiba as **iniciais em ordem alfabética**,
- para cada inicial, exiba os **nomes** ordenados alfabeticamente na mesma linha, separados por vírgula.

Execução simulada:

```
Quantos nomes? 6
Nome 1: Ana
Nome 2: Alice
Nome 3: Bruno
Nome 4: Beatriz
Nome 5: Carla
Nome 6: Carlos

A: Alice, Ana
B: Beatriz, Bruno
C: Carla, Carlos
```

7) Leia uma **linha** contendo inteiros separados por espaço (por exemplo: `10 5 7 20`). Some todos os valores e exiba `Soma = <valor>`. Trate:

- conteúdo **não numérico** (`ValueError → Erro: conteúdo não numérico.`),
- ao final, **sempre** mostre `Fim.` usando `finally`.
  Se der certo, exiba `Soma = <valor>`.

8) Mostre um menu:

```
1 - Dobrar número
2 - Inverter string
3 - Sair
```

Leia a opção e execute:

- `1`: leia um **inteiro**, mostre `Resultado: <n*2>`.
- `2`: leia uma **string**, mostre `Resultado: <string invertida>`.
- `3`: mostre `Encerrando...`
  Trate **entrada não numérica de opção** e **opção inválida** (qualquer coisa fora de 1–3), exibindo mensagens apropriadas e **não encerrando** até o usuário escolher 3.

Execução simulada (trecho):

```
Opção: x
Erro: digite um número válido.
Opção: 5
Erro: opção inválida.
Opção: 1
Número: 7
Resultado: 14
Opção: 3
Encerrando...
```

9) Implemente uma função `validar_data(dia, mes, ano)` que:

- aceita `1 ≤ mes ≤ 12` e `1 ≤ dia ≤ 31`,
- meses **4, 6, 9, 11** não aceitam **31**,
- **fevereiro (2)** aceita **até 29** (não precisa checar bissexto). Se inválida, **lance** `ValueError("data_invalida")`; se válida, **retorne** `True`.

  Em seguida, leia `dia`, `mes`, `ano` do usuário, chame a função, e:
  - se não lançar, mostre **"Data válida"**,
  - se lançar, capture e mostre **"Data inválida"**.

Execução simulada 1:

```
Dia: 31
Mês: 4
Ano: 2025
Data inválida
```

Execução simulada 2:

```
Dia: 29
Mês: 2
Ano: 2026
Data válida
```

10) Implemente **três classes de exceção personalizadas** que herdem de `Exception`:

- `ValorNegativoError` (para `valor < 0`)
- `AcimaDoLimiteError` (para `valor > limite`)
- `SaldoInsuficienteError` (para `valor > saldo`)

Em seguida, implemente uma função `sacar(saldo, valor, limite)` com as regras:

- `valor < 0 → raise ValorNegativoError("Valor Negativo")`
- `valor > limite → raise AcimaDoLimiteError("Acima do Limite")`
- `valor > saldo → raise SaldoInsuficienteError("Saldo Insuficiente")`
- caso válido, **retorne** `novo_saldo = saldo - valor`.

Depois, leia `saldo`, `valor`, `limite`; chame a função e:

- se não lançar, mostre **"**`Novo saldo: R$ <valor>`**"** com 2 casas,
- se lançar, capture e exiba a **mensagem** da exceção.

Execução simulada 1:

```
Saldo: 100.00
Valor do saque: 120.00
Limite por operação: 200.00
Saldo Insuficiente
```

Execução simulada 2:

```
Saldo: 300.00
Valor do saque: 250.00
Limite por operação: 200.00
Acima do Limite
```

Execução simulada 3:

```
Saldo: 500.00
Valor do saque: -50.00
Limite por operação: 300.00
Valor Negativo
```

Execução simulada 4:

```
Saldo: 500.00
Valor do saque: 120.00
Limite por operação: 300.00
Novo saldo: R$ 380.00
```