📚 Coleções em Python

🎯 Introdução às Coleções

Coleções são estruturas de dados fundamentais em Python que permitem armazenar múltiplos valores em uma única variável. Cada tipo de coleção possui características específicas que as tornam adequadas para diferentes cenários de programação.

📊 Visão Geral Comparativa

Coleção	Mutável	Ordenada	Duplicados	Uso Típico
Lista	✅ Sim	✅ Sim	✅ Sim	Sequências de elementos modificáveis
Dicionário	✅ Sim	✅ Sim (Python 3.7+)	❌ Chaves únicas	Dados com identificadores únicos
Tupla	❌ Não	✅ Sim	✅ Sim	Dados constantes/imutáveis
Conjunto	✅ Sim	❌ Não	❌ Não	Elementos únicos, operações matemáticas

---

📋 LISTAS (Lists) - Explicação Detalhada

❓ O que são e quando usar?

Listas são sequências ordenadas e mutáveis que permitem armazenar elementos heterogêneos (diferentes tipos). Imagine uma lista de compras ou uma playlist de músicas - você pode adicionar, remover e reordenar itens conforme necessário.

Quando usar listas:

- Quando a ordem dos elementos importa
- Quando você precisa modificar os dados frequentemente
- Para armazenar sequências de itens relacionados

🏗️ Criação e Estrutura Básica

```
python
```

```python
# Exemplos de criação de listas
# ----------------------------------------------------------------
# Lista vazia - ponto de partida comum
tarefas_pendentes = []

# Lista com elementos do mesmo tipo
idades_alunos = [18, 19, 20, 21, 19]

# Lista mista (Python permite diferentes tipos)
registro_aluno = ["Maria Silva", 25, 8.5, True]

# Lista aninhada (matriz ou estrutura multidimensional)
matriz_notas = [
    [8.5, 9.0, 7.5],  # Notas do aluno 1
    [6.0, 7.5, 8.0],  # Notas do aluno 2
    [9.5, 9.0, 9.5]   # Notas do aluno 3
]

print(f"Total de alunos: {len(idades_alunos)}")

print(f"Primeira nota do primeiro aluno: {matriz_notas[0][0]}")
```

🔍 Acesso e Slicing (Fatiamento)

python

```python
# ----------------------------------------------------------------
# 📍 ACESSANDO ELEMENTOS INDIVIDUAIS
# ----------------------------------------------------------------
frutas = ["maçã", "banana", "laranja", "uva", "manga"]

# Acesso por índice positivo (começa em 0)
primeira_fruta = frutas[0]        # "maçã"
terceira_fruta = frutas[2]        # "laranja"

# Acesso por índice negativo (último é -1)
ultima_fruta = frutas[-1]         # "manga"
penultima_fruta = frutas[-2]      # "uva"

# ----------------------------------------------------------------
# ✂️ SLICING (FATIAMENTO) - EXTREMAMENTE ÚTIL!
# ----------------------------------------------------------------
# Sintaxe: lista[início:fim:passo]
# início: inclusivo, fim: exclusivo

primeiras_tres = frutas[0:3]      # ["maçã", "banana", "laranja"]
# Equivalente a:
primeiras_tres = frutas[:3]       # Omitir início começa do 0

do_segundo_ao_fim = frutas[1:]    # ["banana", "laranja", "uva", "manga"]
# Omitir fim vai até o final

apenas_pares = frutas[::2]        # ["maçã", "laranja", "manga"]
# Passo 2: pega elementos de 2 em 2

invertida = frutas[::-1]          # ["manga", "uva", "laranja", "banana",
"maçã"]

# Passo negativo inverte a lista
```

✏️ Modificação de Listas - Operações Comuns

python

```python
# ------------------------------------------------------------------
# ➕ ADIÇÃO DE ELEMENTOS
# ------------------------------------------------------------------
compras = ["pão", "leite"]

# append() - adiciona UM elemento ao FINAL
compras.append("ovos")
# Resultado: ["pão", "leite", "ovos"]

# insert() - adiciona em POSIÇÃO ESPECÍFICA
compras.insert(1, "queijo")
# Resultado: ["pão", "queijo", "leite", "ovos"]
# Cuidado: elementos são deslocados para a direita

# extend() - adiciona VÁRIOS elementos
compras.extend(["café", "açúcar"])
# Resultado: ["pão", "queijo", "leite", "ovos", "café", "açúcar"]
# Equivalente a: compras += ["café", "açúcar"]

# ------------------------------------------------------------------
# ➖ REMOÇÃO DE ELEMENTOS
# ------------------------------------------------------------------
# remove() - remove pelo VALOR (primeira ocorrência)
compras.remove("queijo")
# Resultado: ["pão", "leite", "ovos", "café", "açúcar"]

# pop() - remove pelo ÍNDICE (retorna o elemento removido)
removido = compras.pop(2)          # Remove "ovos"
# removido = "ovos"
# Resultado: ["pão", "leite", "café", "açúcar"]

# pop() sem índice remove o ÚLTIMO
ultimo = compras.pop()             # Remove "açúcar"
# Resultado: ["pão", "leite", "café"]

# del - remove pelo índice (sem retornar)
del compras[0]                     # Remove "pão"
# Resultado: ["leite", "café"]

# clear() - esvazia a lista completamente
compras.clear()

# Resultado: []
```

🔎 Operações de Consulta e Informação

python

```python
# ------------------------------------------------------------------
# 📊 INFORMAÇÕES SOBRE A LISTA
# ------------------------------------------------------------------
notas = [8.5, 9.0, 7.5, 8.5, 6.0, 8.5]

# len() - comprimento da lista
total_notas = len(notas)          # 6

# count() - conta ocorrências de um valor
vezes_85 = notas.count(8.5)       # 3

# index() - encontra a PRIMEIRA posição de um valor
posicao_90 = notas.index(9.0)     # 1
# Se o valor não existe, gera erro ValueError

# in - verifica existência (retorna True/False)
tem_60 = 6.0 in notas             # True
tem_100 = 10.0 in notas           # False

# min(), max(), sum() - estatísticas básicas
menor_nota = min(notas)           # 6.0
maior_nota = max(notas)           # 9.0
soma_notas = sum(notas)           # 48.0

media = sum(notas) / len(notas)   # 8.0
```

🔄 Ordenação e Organização

python

```python
# ------------------------------------------------------------------
# 🔠 ORDENAÇÃO DE LISTAS
# ------------------------------------------------------------------
numeros = [5, 2, 8, 1, 9]

# sort() - ordena IN PLACE (modifica a lista original)
numeros.sort()                    # [1, 2, 5, 8, 9]
# A lista original é alterada!
```

```python
# sort(reverse=True) - ordem decrescente
numeros.sort(reverse=True)        # [9, 8, 5, 2, 1]

# sorted() - retorna NOVA lista ordenada
outra_lista = [3, 1, 4, 1, 5]
ordenada = sorted(outra_lista)    # [1, 1, 3, 4, 5]
# outra_lista permanece [3, 1, 4, 1, 5]

# reverse() - inverte a ordem IN PLACE
palavras = ["c", "a", "b"]

palavras.reverse()                # ["b", "a", "c"]
```

⚠ Cuidado: Cópia vs Referência

python

```python
# ----------------------------------------------------------------
# 📋 CÓPIA DE LISTAS - UM DOS ERROS MAIS COMUNS!
# ----------------------------------------------------------------
# ATRIBUIÇÃO cria uma REFERÊNCIA, não uma cópia!
lista_original = [1, 2, 3]
referencia = lista_original      # Aponta para a MESMA lista
referencia[0] = 99

print(lista_original)            # [99, 2, 3] ← ALTERADO!
print(referencia)                # [99, 2, 3]

# ----------------------------------------------------------------
# FORMAS CORRETAS DE COPIAR:
# ----------------------------------------------------------------
# 1. Método copy()
copia1 = lista_original.copy()

# 2. Slicing completo
copia2 = lista_original[:]

# 3. Construtor list()
copia3 = list(lista_original)

# 4. Cópia profunda (para listas aninhadas)
```

```
import copy
lista_complexa = [[1, 2], [3, 4]]

copia_profunda = copy.deepcopy(lista_complexa)
```

🗂️ DICIONÁRIOS (Dictionaries) - Explicação Detalhada

❓ O que são e quando usar?

Dicionários são estruturas que armazenam dados como pares chave-valor. Pense em um dicionário de tradução: você procura uma palavra (chave) e encontra sua tradução (valor).

Quando usar dicionários:

- Quando cada elemento tem um identificador único (ID, CPF, matrícula)
- Para armazenar propriedades de um objeto (nome, idade, email)
- Quando precisa buscar valores rapidamente por uma chave

🏗️ Criação e Estrutura Básica

```
python

# ----------------------------------------------------------------
# 📝 FORMAS DE CRIAR DICIONÁRIOS
# ----------------------------------------------------------------
# Dicionário vazio (para preencher depois)
aluno = {}
produtos = dict()

# Dicionário com dados iniciais
aluno_completo = {
    "nome": "João Silva",
    "idade": 22,
    "matricula": "2023001",
    "curso": "Engenharia",
    "ativo": True,
```

```
    "notas": [8.5, 9.0, 7.5]
}
# As chaves DEVEM ser imutáveis (strings, números, tuplas)

# Usando dict() com pares chave=valor
produto = dict(
    nome="Notebook Gamer",
    preco=3500.00,
    estoque=15,
    categoria="Eletrônicos"
)

print(f"Aluno: {aluno_completo['nome']}")

print(f"Preço do produto: R${produto['preco']:.2f}")
```

🔍 Acesso e Busca de Valores

```
python
# ----------------------------------------------------------------
# 🔑 ACESSANDO VALORES - DUAS FORMAS PRINCIPAIS
# ----------------------------------------------------------------
pessoa = {"nome": "Maria", "idade": 25, "cidade": "São Paulo"}

# 1. Colchetes [] - cuidado: gera KeyError se chave não existe
nome = pessoa["nome"]            # "Maria"
# idade = pessoa["altura"]       # KeyError: 'altura' não existe

# 2. Método get() - mais seguro (retorna None se não encontrar)
nome = pessoa.get("nome")        # "Maria"
altura = pessoa.get("altura")    # None (sem erro)
idade = pessoa.get("idade", 0)   # Se não existir, retorna 0

# 3. setdefault() - obtém valor OU cria com valor padrão
cidade = pessoa.setdefault("cidade", "Não informada")
estado = pessoa.setdefault("estado", "SP")  # Cria "estado": "SP"
print(pessoa)  # Agora tem chave "estado"

# ----------------------------------------------------------------
# 🎯 VERIFICAÇÃO DE EXISTÊNCIA
# ----------------------------------------------------------------
```

```python
# in verifica se chave EXISTE no dicionário
if "nome" in pessoa:
    print(f"Nome: {pessoa['nome']}")

if "telefone" not in pessoa:
    print("Telefone não cadastrado")
```

✏️ Modificação de Dicionários

```python
# ----------------------------------------------------------------
# ✏️ ADICIONANDO E ATUALIZANDO VALORES
# ----------------------------------------------------------------
cadastro = {"nome": "Carlos"}

# Adicionar nova chave-valor
cadastro["idade"] = 30
cadastro["profissao"] = "Engenheiro"

# Atualizar valor existente
cadastro["idade"] = 31

# update() - atualiza múltiplos valores de uma vez
cadastro.update({
    "cidade": "Rio de Janeiro",
    "estado": "RJ",
    "idade": 32  # Atualiza valor existente também
})

# ----------------------------------------------------------------
# 🗑️ REMOVENDO ELEMENTOS
# ----------------------------------------------------------------
# pop() - remove pela CHAVE e RETORNA o valor
valor_removido = cadastro.pop("estado")  # Remove "estado": "RJ"

# popitem() - remove o ÚLTIMO par inserido (Python 3.7+)
chave, valor = cadastro.popitem()  # Remove último item

# del - remove sem retornar valor
del cadastro["cidade"]
```

```python
# clear() - remove todos os elementos
backup = cadastro.copy()  # Faz cópia antes de limpar

cadastro.clear()          # Dicionário vazio: {}
```

🔎 Operações de Consulta e Iteração

```python
# ----------------------------------------------------------------
# 📊 MÉTODOS PARA OBTENÇÃO DE INFORMAÇÕES
# ----------------------------------------------------------------
cliente = {
    "id": 1001,
    "nome": "Ana Santos",
    "email": "ana@email.com",
    "vip": True,
    "compras": [150.0, 89.90, 299.99]
}

# keys() - retorna todas as CHAVES
chaves = cliente.keys()
print("Chaves disponíveis:", list(chaves))

# values() - retorna todos os VALORES
valores = cliente.values()
print("Valores:", list(valores))

# items() - retorna todos os PARES (chave, valor)
itens = cliente.items()
print("Itens:", list(itens))

# len() - número de pares chave-valor
total_itens = len(cliente)  # 5

# ----------------------------------------------------------------
# 🔄 ITERAÇÃO EM DICIONÁRIOS
# ----------------------------------------------------------------
print("\n--- Dados do Cliente ---")

# Por padrão, itera sobre as CHAVES
```

```python
for chave in cliente:
    print(f"{chave}: {cliente[chave]}")

# Forma mais explícita (recomendada)
for chave, valor in cliente.items():
    print(f"{chave}: {valor}")

# Apenas valores
for valor in cliente.values():
    print(f"Valor: {valor}")

# Apenas chaves
for chave in cliente.keys():
    print(f"Chave: {chave}")
```

📦 TUPLAS e 🎲 CONJUNTOS - Resumo Essencial

Tuplas (Imutáveis)

```python
# ----------------------------------------------------------------
# 📦 TUPLAS - DADOS IMUTÁVEIS
# ----------------------------------------------------------------
# Criação com parênteses (ou sem)
coordenadas = (10, 20)          # Tupla de coordenadas
cores_rgb = ("vermelho", "verde", "azul")
ponto = 30, 40                  # Tupla sem parênteses também funciona

# Tupla com um elemento (vírgula obrigatória!)
singleton = (42,)               # Correto
nao_tupla = (42)                # ERRADO: é um inteiro, não tupla

# Acesso (igual às listas)
x = coordenadas[0]              # 10
y = coordenadas[1]              # 20

# Desempacotamento (útil para múltiplos retornos)
```

```python
def obter_coordenadas():
    return 100, 200  # Retorna tupla implicitamente

x, y = obter_coordenadas()       # x=100, y=200
# Muito usado para trocar valores:
a, b = 5, 10
a, b = b, a                      # a=10, b=5

# Imutabilidade - NÃO pode modificar!
# coordenadas[0] = 15            # TypeError: não suporta atribuição
```

Conjuntos (Elementos Únicos)

```python
# ----------------------------------------------------------------
# 🎲 CONJUNTOS - ELEMENTOS ÚNICOS
# ----------------------------------------------------------------
# Criação com chaves {} ou set()
vogais = {"a", "e", "i", "o", "u"}
numeros = set([1, 2, 3, 4, 5])

# Conjunto vazio (precisa de set(), não {})
vazio = set()                    # Correto
# vazio = {}                    # ERRADO: cria dicionário!

# Principal uso: remover duplicatas
lista_com_duplicatas = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4]
lista_sem_duplicatas = list(set(lista_com_duplicatas))
# Resultado: [1, 2, 3, 4] (ordem pode variar)

# ----------------------------------------------------------------
# 🎯 OPERAÇÕES MATEMÁTICAS COM CONJUNTOS
# ----------------------------------------------------------------
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# União (todos os elementos)
uniao = A | B                    # {1, 2, 3, 4, 5, 6, 7, 8}
uniao = A.union(B)               # Método equivalente
```

```python
# Interseção (elementos em ambos)
intersecao = A & B                # {4, 5}
intersecao = A.intersection(B)

# Diferença (só em A, não em B)
diferenca = A - B                 # {1, 2, 3}
diferenca = A.difference(B)

# Diferença simétrica (ou em A ou em B, não em ambos)
dif_simetrica = A ^ B             # {1, 2, 3, 6, 7, 8}
```

```
dif_simetrica = A.symmetric_difference(B)
```

🔁 LÓGICA DE ITERAÇÃO - Explicação Pedagógica

❓ Por que iterar?

Iteração é o processo de percorrer cada elemento de uma coleção. É essencial para:

- Processar todos os itens de uma lista
- Buscar informações específicas
- Realizar operações em massa
- Transformar dados

🔄 Padrão Básico de Iteração

```
python
```

```python
# ----------------------------------------------------------------
# 🔄 PADRÃO FUNDAMENTAL: for elemento in colecao
# ----------------------------------------------------------------
# Para CADA elemento na coleção, faça algo
frutas = ["maçã", "banana", "laranja"]

for fruta in frutas:              # "Para cada fruta na lista frutas"
    print(f"Fruta: {fruta}")
```

```
    # A variável 'fruta' assume cada valor da lista

# Funciona igual para outras coleções:

# Tuplas, strings, conjuntos, dicionários (nas chaves)
```

🎯 Iteração com Contador: `enumerate()`

python

```
# ------------------------------------------------------------------
# 📍 QUANDO PRECISO DO ÍNDICE E DO VALOR
# ------------------------------------------------------------------
alunos = ["Ana", "Bruno", "Carla"]

# Forma INEFICIENTE (evitar):
indice = 0
for aluno in alunos:
    print(f"{indice}: {aluno}")
    indice += 1

# Forma CORRETA com enumerate():
for indice, aluno in enumerate(alunos):
    print(f"Aluno {indice + 1}: {aluno}")
    # enumerate retorna (índice, valor)

# Podemos começar de outro número:
for numero_linha, aluno in enumerate(alunos, start=1):
    print(f"Linha {numero_linha}: {aluno}")

# Útil em situações como:
for i, nota in enumerate([8.5, 9.0, 7.5]):
    if nota > 8.0:
        print(f"Nota alta na posição {i}: {nota}")
```

🔗 Iteração Paralela: `zip()`

python

```
# ------------------------------------------------------------------
```

```python
# 🤝 QUANDO TENHO VÁRIAS LISTAS RELACIONADAS
# ----------------------------------------------------------------
nomes = ["Ana", "Bruno", "Carla"]
idades = [25, 30, 28]
cursos = ["Medicina", "Direito", "Engenharia"]

# PROBLEMA: Processar listas relacionadas sem zip
for i in range(len(nomes)):      # Funciona, mas é verboso
    print(f"{nomes[i]} tem {idades[i]} anos e faz {cursos[i]}")

# SOLUÇÃO ELEGANTE com zip():
for nome, idade, curso in zip(nomes, idades, cursos):
    print(f"{nome} tem {idade} anos e faz {curso}")

# zip() combina elementos na mesma posição

# Para até a menor lista terminar
```

🎭 Iteração em Dicionários - Padrões Comuns

python

```python
# ----------------------------------------------------------------
# 📚 ITERAÇÃO EM DICIONÁRIOS - 3 PADRÕES
# ----------------------------------------------------------------
produto = {
    "nome": "Notebook",
    "preco": 3500.00,
    "estoque": 10,
    "categoria": "Eletrônicos"
}

# 1. Apenas valores (quando as chaves não importam)
total_valor = 0
for valor in produto.values():
    if isinstance(valor, (int, float)):
        total_valor += valor
print(f"Soma dos valores numéricos: {total_valor}")

# 2. Apenas chaves (quero processar chaves específicas)
for chave in produto:
    if chave.startswith("p"):  # Chaves que começam com 'p'
```

```
        print(f"Chave encontrada: {chave}")

# 3. Pares completos (mais comum)
for chave, valor in produto.items():
    print(f"Campo '{chave}' = {valor}")

    # Ideal para transformar, filtrar ou processar dados
```

🔄 Iteração Aninhada (Matrizes)

```
python
# ----------------------------------------------------------------
# 🧱 ITERAÇÃO ANINHADA - PERCORRER MATRIZES
# ----------------------------------------------------------------
# Lista de listas (matriz 3x3)
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Para cada linha na matriz
for linha in matriz:
    # Para cada elemento na linha
    for elemento in linha:
        print(elemento, end=" ")
    print()  # Nova linha após cada linha da matriz
# Output:
# 1 2 3
# 4 5 6
# 7 8 9

# Exemplo prático: soma de todas as notas
notas_alunos = [
    [8.5, 9.0, 7.5],  # Notas do aluno 1
    [6.0, 7.5, 8.0],  # Notas do aluno 2
    [9.5, 9.0, 9.5]   # Notas do aluno 3
]

soma_total = 0
contador = 0
```

```
for notas in notas_alunos:          # Para cada aluno
    for nota in notas:              # Para cada nota do aluno
        soma_total += nota
        contador += 1

media_geral = soma_total / contador

print(f"Média geral da turma: {media_geral:.2f}")
```

🎨 COMPREHENSIONS - Operações Elegantes em Uma Linha

❓ O que são e por que usar?

Comprehensions são formas concisas de criar ou transformar coleções em Python. Elas combinam iteração e filtragem em uma única expressão.

Vantagens:

- Código mais limpo e legível
- Maior desempenho em muitos casos
- Expressividade (diz O QUE fazer, não COMO)

📋 List Comprehensions

```
python
# ----------------------------------------------------------------
# 📋 LIST COMPREHENSIONS - TRANSFORMAÇÕES CONCISAS
# ----------------------------------------------------------------
# Sintaxe básica: [expressão for item in iterável]

# 📌 EXEMPLO 1: Transformação simples
# De: números de 0 a 9
# Para: quadrados desses números
```

```python
quadrados = [x**2 for x in range(10)]
# Equivalente a:
# quadrados = []
# for x in range(10):
#     quadrados.append(x**2)

# 📌 EXEMPLO 2: Com filtro (condicional no final)
# Apenas números pares ao quadrado
pares_ao_quadrado = [x**2 for x in range(10) if x % 2 == 0]
# [0, 4, 16, 36, 64]
# Lê-se: "x ao quadrado para cada x em range(10) se x for par"

# 📌 EXEMPLO 3: Com condicional ternário
# Classifica números como "par" ou "ímpar"
classificacao = ["par" if x % 2 == 0 else "ímpar" for x in range(5)]
# ["par", "ímpar", "par", "ímpar", "par"]

# 📌 EXEMPLO 4: Aninhado (duas dimensões)
combinacoes = [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x != y]
# [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]

# 📌 EXEMPLO PRÁTICO: Processando dados
nomes = ["  ana ", " PEDRO", "Carla  "]
nomes_limpos = [nome.strip().title() for nome in nomes]
# ["Ana", "Pedro", "Carla"]
```

🗂️ Dictionary Comprehensions

```python
# ----------------------------------------------------------------
# 🗂️ DICTIONARY COMPREHENSIONS
# ----------------------------------------------------------------
# Sintaxe: {chave: valor for item in iterável}

# 📌 EXEMPLO 1: Criando dicionário a partir de lista
# Números e seus quadrados
quadrados_dict = {x: x**2 for x in range(5)}
# {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}

# 📌 EXEMPLO 2: Filtrando dicionário existente
```

```python
produtos = {
    "mouse": 50,
    "teclado": 150,
    "monitor": 800,
    "fone": 80,
    "webcam": 120
}

# Apenas produtos acima de R$100
caros = {nome: preco for nome, preco in produtos.items() if preco > 100}
# {"teclado": 150, "monitor": 800, "webcam": 120}

# 📌 EXEMPLO 3: Transformando valores
# Aplicar 10% de desconto para produtos acima de R$100
com_desconto = {
    nome: preco * 0.9 if preco > 100 else preco
    for nome, preco in produtos.items()
}
# {"mouse": 50, "teclado": 135.0, "monitor": 720.0, "fone": 80, "webcam":
108.0}

# 📌 EXEMPLO PRÁTICO: Invertendo chave-valor (se valores são únicos)
traducao_pt_en = {"gato": "cat", "cachorro": "dog", "casa": "house"}
traducao_en_pt = {ingles: portugues for portugues, ingles in
traducao_pt_en.items()}
# {"cat": "gato", "dog": "cachorro", "house": "casa"}
```

🎲 Set Comprehensions

```python
# ----------------------------------------------------------------
# 🎲 SET COMPREHENSIONS
# ----------------------------------------------------------------
# Sintaxe: {expressão for item in iterável}

# 📌 EXEMPLO 1: Criar conjunto de quadrados
quadrados_set = {x**2 for x in range(10)}
# {0, 1, 4, 9, 16, 25, 36, 49, 64, 81}

# 📌 EXEMPLO 2: Remover duplicatas e transformar
```

```python
nomes = ["Ana", "ana", "Bruno", "bruno", "Carla"]
nomes_unicos = {nome.title() for nome in nomes}
# {"Ana", "Bruno", "Carla"} - Unicidade + padronização

# 📌 EXEMPLO 3: Filtro complexo
numeros = {x for x in range(20) if x % 2 == 0 and x % 3 == 0}
# Números pares E múltiplos de 3

# {0, 6, 12, 18}
```

📊 Quando NÃO usar Comprehensions

```
python
# ----------------------------------------------------------------
# ⚠️ CUIDADO: NÃO USE COMPREHENSIONS PARA TUDO!
# ----------------------------------------------------------------
# RUIM: Comprehension complexa e ilegível
resultado = [x**2 for x in [y for y in range(10) if y % 2 == 0] if x > 10]
# Difícil de entender e manter

# BOM: Divida em etapas claras
numeros_pares = [y for y in range(10) if y % 2 == 0]
quadrados = [x**2 for x in numeros_pares]
resultado = [q for q in quadrados if q > 10]
# Mesmo resultado, muito mais legível

# RUIM: Comprehension com muitos efeitos colaterais
# [print(x) for x in range(5)]  # Cria lista desnecessária

# BOM: Use loop tradicional para ações
for x in range(5):
    print(x)
```

💡 SITUAÇÕES CONTEXTUALIZADAS COM EXPLICAÇÕES DETALHADAS

📊 Caso 1: Sistema de Gerenciamento Escolar

```python
"""
PROBLEMA: Um professor precisa gerenciar notas dos alunos,
calcular estatísticas e identificar situações especiais.
"""

# ----------------------------------------------------------------
# 📁 ESTRUTURA DE DADOS: Lista de dicionários
# Cada aluno é um dicionário, todos em uma lista
# ----------------------------------------------------------------
alunos = [
    {
        "id": 1,
        "nome": "Ana Silva",
        "notas": [8.5, 9.0, 7.5, 8.0],
        "faltas": 2,
        "ativo": True
    },
    {
        "id": 2,
        "nome": "Bruno Oliveira",
        "notas": [6.0, 7.5, 8.0, 5.5],
        "faltas": 5,
        "ativo": True
    },
    {
        "id": 3,
        "nome": "Carla Santos",
        "notas": [9.5, 9.0, 9.5, 10.0],
        "faltas": 1,
        "ativo": True
    },
    {
        "id": 4,
        "nome": "Daniel Costa",
        "notas": [4.0, 5.5, 3.5, 6.0],
        "faltas": 10,
        "ativo": False
    }
```

```
]

# ------------------------------------------------------------------
# 🎯 ETAPA 1: Calcular média de cada aluno
# Exemplo de como processar dados aninhados
# ------------------------------------------------------------------
print("=== CÁLCULO DE MÉDIAS ===")

for aluno in alunos:  # Para cada aluno na lista
    if aluno["ativo"]:  # Só processa alunos ativos
        notas = aluno["notas"]  # Lista de notas do aluno
        media = sum(notas) / len(notas)
        aluno["media"] = round(media, 2)  # Adiciona nova chave

        print(f"{aluno['nome']}: Média = {aluno['media']}")
        # Adicionamos uma NOVA informação à estrutura existente

# ------------------------------------------------------------------
# 🎯 ETAPA 2: Identificar situação do aluno
# Exemplo de tomada de decisão baseada em dados
# ------------------------------------------------------------------
print("\n=== SITUAÇÃO DOS ALUNOS ===")

for aluno in alunos:
    if aluno.get("ativo", False):  # Usa get() com valor padrão
        media = aluno.get("media", 0)
        faltas = aluno.get("faltas", 0)

        # Lógica de decisão baseada em múltiplas condições
        if media >= 7 and faltas <= 4:
            situacao = "APROVADO"
        elif media >= 5 and faltas <= 6:
            situacao = "RECUPERAÇÃO"
        else:
            situacao = "REPROVADO"

        aluno["situacao"] = situacao
        print(f"{aluno['nome']}: {situacao}")

# ------------------------------------------------------------------
# 🎯 ETAPA 3: Análise estatística da turma
# Exemplo de agregação de dados
```

```python
# ------------------------------------------------------------------
print("\n=== ESTATÍSTICAS DA TURMA ===")

# Usando list comprehension para filtrar dados
alunos_ativos = [a for a in alunos if a["ativo"]]
medias = [a["media"] for a in alunos_ativos if "media" in a]

if medias:  # Verifica se há médias calculadas
    media_turma = sum(medias) / len(medias)
    melhor_media = max(medias)
    pior_media = min(medias)

    # Encontrar aluno com melhor média (usando max com função personalizada)
    melhor_aluno = max(alunos_ativos, key=lambda x: x.get("media", 0))

    print(f"Média da turma: {media_turma:.2f}")
    print(f"Melhor média: {melhor_media:.2f} ({melhor_aluno['nome']})")
    print(f"Pior média: {pior_media:.2f}")

# ------------------------------------------------------------------
# 🎯 ETAPA 4: Gerar relatório formatado
# Exemplo de saída estruturada
# ------------------------------------------------------------------
print("\n" + "="*50)
print("RELATÓRIO FINAL DA TURMA".center(50))
print("="*50)

for aluno in sorted(alunos_ativos, key=lambda x: x["nome"]):
    print(f"\n{aluno['nome']}:")
    print(f"  Média: {aluno.get('media', 'N/A'):>6}")
    print(f"  Faltas: {aluno.get('faltas', 0):>5}")
    print(f"  Situação: {aluno.get('situacao', 'PENDENTE'):>12}")

    # Análise detalhada das notas
    if "notas" in aluno:
        notas_str = ", ".join([f"{n:.1f}" for n in aluno["notas"]])
        print(f"  Notas: [{notas_str}]")
```

🛒 Caso 2: Sistema de Carrinho de Compras com Estoque

```python
"""
PROBLEMA: E-commerce precisa gerenciar estoque,
processar vendas e calcular totais.
"""

# ------------------------------------------------------------------
# 📁 ESTRUTURA DE DADOS: Dicionário com dicionários aninhados
# Produtos organizados por ID para busca rápida
# ------------------------------------------------------------------
estoque = {
    101: {
        "nome": "Mouse Gamer RGB",
        "preco": 89.90,
        "quantidade": 50,
        "categoria": "Periféricos",
        "ativo": True
    },
    102: {
        "nome": "Teclado Mecânico",
        "preco": 249.90,
        "quantidade": 30,
        "categoria": "Periféricos",
        "ativo": True
    },
    103: {
        "nome": "Monitor 24\" Full HD",
        "preco": 899.90,
        "quantidade": 15,
        "categoria": "Monitores",
        "ativo": True
    },
    104: {
        "nome": "Headset Gamer",
        "preco": 159.90,
        "quantidade": 0,  # Esgotado
        "categoria": "Áudio",
        "ativo": False  # Produto inativo
    }
}

# ------------------------------------------------------------------
```

```python
# 🛒 SIMULAÇÃO DE CARRINHO DE COMPRAS
# Representado como lista de tuplas (id_produto, quantidade)
# ----------------------------------------------------------------
carrinho = [
    (101, 2),  # 2 mouses
    (102, 1),  # 1 teclado
    (103, 1)   # 1 monitor
]

# ----------------------------------------------------------------
# 🎯 FUNÇÃO: Exibir carrinho com cálculos
# Exemplo de processamento de dados relacionais
# ----------------------------------------------------------------
def exibir_carrinho(carrinho, estoque):
    """Exibe o carrinho de compras formatado"""
    print("🛒 SEU CARRINHO DE COMPRAS")
    print("-" * 40)

    total_carrinho = 0
    itens_validos = []

    for id_produto, quantidade in carrinho:
        # Verifica se produto existe no estoque
        produto = estoque.get(id_produto)

        if not produto:
            print(f"⚠️  Produto ID {id_produto} não encontrado!")
            continue

        if not produto["ativo"]:
            print(f"⏸️  Produto '{produto['nome']}' está indisponível")
            continue

        if quantidade > produto["quantidade"]:
            print(f"❌ Estoque insuficiente de '{produto['nome']}'")
            continue

        # Cálculo do subtotal
        subtotal = produto["preco"] * quantidade
        total_carrinho += subtotal

        # Armazena item válido
```

```
        itens_validos.append((id_produto, quantidade))

        # Exibe item formatado
        print(f"✅ {produto['nome']}")
        print(f"   Quantidade: {quantidade}")
        print(f"   Preço unitário: R${produto['preco']:.2f}")
        print(f"   Subtotal: R${subtotal:.2f}")
        print()

    print("-" * 40)
    print(f"TOTAL DO CARRINHO: R${total_carrinho:.2f}")

    return itens_validos, total_carrinho

# ----------------------------------------------------------------
# 🎯 FUNÇÃO: Processar venda e atualizar estoque
# Exemplo de modificação segura de dados
# ----------------------------------------------------------------
def processar_venda(carrinho, estoque):
    """Processa a venda e atualiza o estoque"""
    print("\n🔄 PROCESSANDO VENDA...")

    for id_produto, quantidade in carrinho:
        produto = estoque[id_produto]

        # Atualiza quantidade em estoque
        produto["quantidade"] -= quantidade

        # Registra histórico da venda
        produto.setdefault("vendas", []).append({
            "quantidade": quantidade,
            "data": "2024-01-15"  # Em sistema real, seria datetime.now()
        })

        print(f"✓ {produto['nome']}: Estoque atualizado")
        print(f"  Antes: {produto['quantidade'] + quantidade} unidades")
        print(f"  Depois: {produto['quantidade']} unidades")

    print("\n✅ Venda processada com sucesso!")

# ----------------------------------------------------------------
# 🎯 FUNÇÃO: Gerar relatório de estoque
```

```python
# Exemplo de análise de dados com múltiplos critérios
# ----------------------------------------------------------------
def gerar_relatorio_estoque(estoque):
    """Gera relatório detalhado do estoque"""
    print("\n" + "="*50)
    print("📊 RELATÓRIO DE ESTOQUE".center(50))
    print("="*50)

    # Categorias disponíveis
    categorias = set()
    for produto in estoque.values():
        if produto["ativo"]:
            categorias.add(produto["categoria"])

    print(f"\nCategorias ativas: {', '.join(sorted(categorias))}")

    # Análise por categoria
    for categoria in sorted(categorias):
        print(f"\n📁 CATEGORIA: {categoria}")
        print("-" * 30)

        # Filtra produtos da categoria
        produtos_categoria = [
            p for p in estoque.values()
            if p["categoria"] == categoria and p["ativo"]
        ]

        for produto in sorted(produtos_categoria, key=lambda x: x["nome"]):
            status = "🟢" if produto["quantidade"] > 10 else \
                     "🟡" if produto["quantidade"] > 0 else \
                     "🔴"

            print(f"{status} {produto['nome']}:")
            print(f"  Preço: R${produto['preco']:.2f}")
            print(f"  Estoque: {produto['quantidade']} unidades")
            print(f"  Valor em estoque: R${produto['preco'] *
produto['quantidade']:.2f}")

# ----------------------------------------------------------------
# 📋 EXECUÇÃO DO SISTEMA
# ----------------------------------------------------------------
# 1. Exibir carrinho
```

```
carrinho_valido, total = exibir_carrinho(carrinho, estoque)

# 2. Processar venda (apenas se houver itens válidos)
if carrinho_valido:
    confirmacao = input("\nDeseja finalizar a compra? (s/n): ")
    if confirmacao.lower() == 's':
        processar_venda(carrinho_valido, estoque)

        gerar_relatorio_estoque(estoque)
```

🔍 Caso 3: Análise de Texto com Múltiplas Coleções

```
python
"""
PROBLEMA: Analisar um texto para extrair estatísticas,
contar palavras e identificar padrões.
"""

# ----------------------------------------------------------------
# 📝 TEXTO DE EXEMPLO PARA ANÁLISE
# ----------------------------------------------------------------
texto = """
Python é uma linguagem de programação de alto nível,
interpretada, de script, imperativa, orientada a objetos,
funcional, de tipagem dinâmica e forte. Foi lançada por
Guido van Rossum em 1991. Atualmente, Python é uma das
linguagens mais populares do mundo, sendo usada em áreas
como ciência de dados, inteligência artificial,
desenvolvimento web, automação e muito mais.

A filosofia do Python é resumida no Zen de Python, que
enfatiza legibilidade, simplicidade e clareza do código.
"""

# ----------------------------------------------------------------
# 🎯 ETAPA 1: Limpeza e preparação do texto
# Exemplo de processamento de strings com listas
# ----------------------------------------------------------------
print("=== ETAPA 1: PREPARAÇÃO DO TEXTO ===\n")

# 1. Converter para minúsculas (padronização)
```

```python
texto_limpo = texto.lower()

# 2. Remover pontuação
import string
for pontuacao in string.punctuation + "\n":
    texto_limpo = texto_limpo.replace(pontuacao, " ")

# 3. Dividir em palavras
palavras = texto_limpo.split()

print(f"Texto original tem {len(texto)} caracteres")
print(f"Após limpeza: {len(palavras)} palavras")
print(f"Primeiras 10 palavras: {palavras[:10]}")

# ---------------------------------------------------------------
# 🎯 ETAPA 2: Análise básica de frequência
# Exemplo de contagem com dicionário
# ---------------------------------------------------------------
print("\n=== ETAPA 2: ANÁLISE DE FREQUÊNCIA ===\n")

# Método manual com dicionário
frequencia = {}
for palavra in palavras:
    frequencia[palavra] = frequencia.get(palavra, 0) + 1

# Método usando Counter (mais eficiente)
from collections import Counter
frequencia_counter = Counter(palavras)

print("As 5 palavras mais comuns:")
for palavra, contagem in frequencia_counter.most_common(5):
    print(f"  '{palavra}': {contagem} ocorrências")

# ---------------------------------------------------------------
# 🎯 ETAPA 3: Estatísticas avançadas
# Exemplo de múltiplas análises combinadas
# ---------------------------------------------------------------
print("\n=== ETAPA 3: ESTATÍSTICAS AVANÇADAS ===\n")

# Usando conjunto para palavras únicas
palavras_unicas = set(palavras)
```

```python
# Análise por tamanho de palavra
contagem_por_tamanho = {}
for palavra in palavras:
    tamanho = len(palavra)
    contagem_por_tamanho[tamanho] = contagem_por_tamanho.get(tamanho, 0) + 1

print(f"Total de palavras únicas: {len(palavras_unicas)}")
print(f"Densidade léxica: {(len(palavras_unicas)/len(palavras))*100:.1f}%")

print("\nDistribuição por tamanho de palavra:")
for tamanho in sorted(contagem_por_tamanho.keys()):
    contagem = contagem_por_tamanho[tamanho]
    porcentagem = (contagem / len(palavras)) * 100
    print(f"  {tamanho:2d} letras: {contagem:3d} palavras
({porcentagem:5.1f}%)")

# ----------------------------------------------------------------
# 🎯 ETAPA 4: Análise semântica simples
# Exemplo de busca e categorização
# ----------------------------------------------------------------
print("\n=== ETAPA 4: ANÁLISE TEMÁTICA ===\n")

# Definir categorias de interesse
categorias = {
    "programação": {"python", "linguagem", "código", "script", "programação"},
    "características": {"alto", "nível", "interpretada", "dinâmica", "forte"},
    "aplicações": {"dados", "inteligência", "artificial", "web", "automação"}
}

# Contar palavras por categoria
contagem_categorias = {categoria: 0 for categoria in categorias}

for palavra in palavras_unicas:
    for categoria, palavras_chave in categorias.items():
        if palavra in palavras_chave:
            contagem_categorias[categoria] += 1

print("Foco do texto por categoria:")
for categoria, contagem in sorted(contagem_categorias.items()):
    if contagem > 0:
        print(f"  {categoria.title()}: {contagem} termos relacionados")
```

```python
# ------------------------------------------------------------------
# 🎯 ETAPA 5: Relatório consolidado
# Exemplo de estruturação de resultados
# ------------------------------------------------------------------
print("\n" + "="*60)
print("📈 RELATÓRIO COMPLETO DA ANÁLISE DE TEXTO".center(60))
print("="*60)

relatorio = {
    "estatisticas_gerais": {
        "total_palavras": len(palavras),
        "palavras_unicas": len(palavras_unicas),
        "densidade_lexica":
f"{(len(palavras_unicas)/len(palavras))*100:.1f}%",
        "palavra_mais_comum": frequencia_counter.most_common(1)[0][0],
        "tamanho_medio_palavra": sum(len(p) for p in palavras) / len(palavras)
    },
    "distribuicao_tamanho": contagem_por_tamanho,
    "foco_tematico": contagem_categorias
}

# Exibir relatório formatado
for secao, dados in relatorio.items():
    print(f"\n{secao.replace('_', ' ').title()}:")
    print("-" * 40)

    if isinstance(dados, dict):
        for chave, valor in dados.items():
            chave_formatada = chave.replace('_', ' ').title()
            print(f"  {chave_formatada}: {valor}")
    else:
        print(f"  {dados}")

# ------------------------------------------------------------------
# 🎯 BÔNUS: Visualização simples dos dados
# ------------------------------------------------------------------
print("\n" + "="*60)
print("📊 VISUALIZAÇÃO DE DISTRIBUIÇÃO".center(60))
print("="*60)

# Gráfico de barras simples no terminal
print("\nDistribuição das 10 palavras mais comuns:")
```

```
for palavra, contagem in frequencia_counter.most_common(10):
    barra = "" * int((contagem / frequencia_counter.most_common(1)[0][1]) *
40)
    print(f"{palavra:15} {barra} {contagem}")
```

🚨 BOAS PRÁTICAS E ARMADILHAS COMUNS

✅ Para Listas:

- Faça: Use `lista.copy()` ou `lista[:]` para cópias reais
- Evite: `lista.insert(0, x)` em loops grandes (é lento)
- Cuidado: Modificar lista enquanto itera sobre ela
- python

```
# ERRADO:
for item in lista:
    if condicao(item):
        lista.remove(item)  # Pode pular elementos!

# CORRETO:
nova_lista = [item for item in lista if not condicao(item)]
# OU itere sobre uma cópia
for item in lista[:]:
    if condicao(item):
```

- ```
          lista.remove(item)
  ```

✅ Para Dicionários:

- Sempre: Use `.get()` em vez de `[]` quando chave pode não existir
- Evite: Modificar dicionário enquanto itera sobre `.keys()` ou `.values()`
- Prefira: `dict comprehension` para transformações

✅ Para Iteração:
  ```

- Use `enumerate()` quando precisar do índice
- Use `zip()` para iterar sobre múltiplas listas em paralelo
- Prefira loops explícitos quando a lógica é complexa

✅ Para Comprehensions:

- Mantenha legibilidade acima de concisão
- Divida comprehensions complexas em múltiplas etapas
- Use para transformações e filtros simples

🎓 CONCLUSÃO E PRÓXIMOS PASSOS

🔄 Resumo dos Conceitos Chave:

1. Listas: Sequências ordenadas e mutáveis, ideais para dados que mudam
2. Dicionários: Pares chave-valor, perfeitos para dados com identificadores
3. Tuplas: Dados imutáveis, úteis para constantes e múltiplos retornos
4. Conjuntos: Elementos únicos, excelentes para operações matemáticas

🧠 Padrões Mentais para Escolha:

- "Preciso manter ordem e modificar?" → Lista
- "Tenho um identificador único para cada item?" → Dicionário
- "São dados constantes que não mudam?" → Tupla
- "Preciso garantir unicidade ou operações de conjunto?" → Conjunto

📚 Prática Recomendada:

1. Recrie os exemplos deste documento
2. Modifique os casos contextualizados com seus próprios dados
3. Crie pequenos projetos usando cada tipo de coleção
4. Experimente converter entre tipos de coleções

🔍 Para Aprofundar:

- Módulo `collections` (Counter, defaultdict, OrderedDict)
- Listas de dicionários vs dicionários de listas
- Performance de diferentes operações
- Uso de geradores (generators) para grandes conjuntos de dados