Questionário de Lógica de Programação – Do Iniciante ao Avançado

■ NÍVEL INICIANTE – Fundamentos e Estruturas Básicas

1. Entrada e saída simples – Treina leitura e exibição de dados. Leia dois números e mostre a soma, subtração, multiplicação e divisão entre eles.

```
num1 = float(input("Digite o primeiro número: "))

num2 = float(input("Digite o segundo número: "))



# Operações

soma = num1 + num2

subtracao = num1 - num2

multiplicacao = num1 * num2



# Evita erro de divisão por zero

if num2 != 0:

    divisao = num1 / num2

else:

    divisao = "Não é possível dividir por zero."



# Exibe resultados

print("Soma:", soma)

print("Subtração:", subtracao)

print("Multiplicação:", multiplicacao)

print("Divisão:", divisao)
```

2. **Maior e menor número** – Pratica condicionais. Leia três números e mostre o maior e o menor.

```
# Lê três números do usuário

a = float(input("Digite o primeiro número: "))

b = float(input("Digite o segundo número: "))

c = float(input("Digite o terceiro número: "))



# Encontrar o maior

if a >= b and a >= c:

    maior = a

elif b >= a and b >= c:

    maior = b

else:

    maior = c



# Encontrar o menor

if a <= b and a <= c:

    menor = a

elif b <= a and b <= c:

    menor = b

else:

    menor = c



# Exibir resultados
```

```
print("Maior número:", maior)

print("Menor número:", menor)
```

3. **Par ou ímpar** – Usa decisão com operador módulo. Leia um número e informe se é par ou ímpar.

```
# Lê um número do usuário

num = int(input("Digite um número: "))



# Verifica se é par ou ímpar usando módulo

if num % 2 == 0:

    print("O número é PAR.")

else:

    print("O número é ÍMPAR.")
```

4. **Tabuada** – Reforça o uso de laços. Mostre a tabuada de um número de 1 a 10.

```
num = int(input("Digite um número para ver a tabuada (1 a 10): "))



print(f"\nTabuada do {num}:\n")

for i in range(1, 11):

    resultado = num * i

    print(f"{num} x {i} = {resultado}")
```

5. **Soma de 10 números** – Treina repetição e acumuladores. Leia 10 números e mostre a soma total.

```
soma = 0  # acumulador



for i in range(10):
```

```
    num = float(input(f"Digite o {i+1}º número: "))

    soma += num  # acumula o valor



print(f"\nA soma total é: {soma}")
```

6. **Contagem regressiva** – Pratica o laço for. Mostre uma contagem regressiva de 10 a 0.

```
for i in range(10, 0 - 1, -1):

    print(i)
```

7. **Fatorial simples** – Exercita multiplicações em loop. Calcule o fatorial de um número sem usar recursão.

```
n = int(input("Digite um número para calcular o fatorial: "))



fatorial = 1  # acumulador



for i in range(1, n + 1):

    fatorial *= i  # multiplica acumulando



print(f"{n}! = {fatorial}")
```

8. **Vetor – Soma e média** – Treina manipulação de vetores. Leia 10 números e mostre soma e média.

```
numeros = []  # vetor para armazenar os 10 valores



for i in range(10):

    num = float(input(f"Digite o {i+1}º número: "))

    numeros.append(num)
```

```
    soma = sum(numeros)
    media = soma / len(numeros)

    print("\nSoma dos números:", soma)
    print("Média dos números:", media)
```

9. Vetor – Números pares e ímpares – Classifique elementos de um vetor entre pares e ímpares.

```
numeros = []

# leitura dos valores
for i in range(10):
    num = int(input(f"Digite o {i+1}º número: "))
    numeros.append(num)

# vetores para classificação
pares = []
impares = []

# classificação
for n in numeros:
    if n % 2 == 0:
        pares.append(n)
```

```
    else:
        impares.append(n)

# saída
print("\nNúmeros pares:", pares)
print("Números ímpares:", impares)
```

10. **Matriz – Soma geral** – Introduz o conceito de matrizes. Crie uma 3x3 e some todos os elementos.

```
    matriz = []  # matriz 3x3
    soma = 0     # acumulador

    # leitura da matriz
    for i in range(3):
        linha = []
        for j in range(3):
            valor = float(input(f"Digite o valor da posição [{i}][{j}]: "))
            linha.append(valor)
            soma += valor
        matriz.append(linha)

    # saída
    print("\nMatriz 3x3:")
    for linha in matriz:
        print(linha)
```

```
print("\nSoma de todos os elementos:", soma)
```

■ NÍVEL INTERMEDIÁRIO – Estruturas, Algoritmos e Recursão

1. **Vetor – Soma dos pares e média dos ímpares** – Pratica filtragem e cálculo em vetores.

```
numeros = []


# leitura do vetor
for i in range(10):
    num = float(input(f"Digite o {i+1}º número: "))
    numeros.append(num)


pares = []
impares = []


# separação dos valores
for n in numeros:
    if n % 2 == 0:
        pares.append(n)
    else:
        impares.append(n)


# cálculos
soma_pares = sum(pares)
media_impares = sum(impares) / len(impares) if impares else 0
```

```
# saída

print("\nSoma dos pares:", soma_pares)

print("Média dos ímpares:", media_impares)
```

2. **Matriz – Diagonal secundária** – Trabalha índices e diagonais. Mostre e some a diagonal secundária de uma 4x4.

```
matriz = []

diagonal_sec = []

soma = 0



# leitura da matriz 4x4

for i in range(4):

    linha = []

    for j in range(4):

        valor = float(input(f"Digite o valor da posição [{i}][{j}]: "))

        linha.append(valor)

    matriz.append(linha)



# obtendo a diagonal secundária

for i in range(4):

    elemento = matriz[i][3 - i]   # posição da diagonal secundária

    diagonal_sec.append(elemento)

    soma += elemento
```

```
# saída

print("\nMatriz 4x4:")

for linha in matriz:

    print(linha)



print("\nDiagonal secundária:", diagonal_sec)

print("Soma da diagonal secundária:", soma)
```

3. **Recursão – Fibonacci reverso** – Usa função recursiva para exibir Fibonacci em ordem inversa.

```
# função recursiva para Fibonacci

def fibonacci(n):

    if n <= 1:

        return n

    return fibonacci(n-1) + fibonacci(n-2)



# programa principal

qtd = int(input("Quantos termos deseja mostrar do Fibonacci? "))



sequencia = []

for i in range(qtd):

    sequencia.append(fibonacci(i))



print("\nFibonacci normal:")

print(sequencia)
```

```
print("\nFibonacci reverso:")

print(sequencia[::-1])
```

4. **Pilha – Verificação de HTML** – Simula uma pilha para validar abertura e fechamento de tags HTML simples.

```
def validar_html(html):

    pilha = []

    i = 0



    while i < len(html):

        # detecta início de uma tag

        if html[i] == '<':

            fechamento = html.find('>', i)

            if fechamento == -1:

                return False  # tag malformada



            tag = html[i+1:fechamento]



            # tag de fechamento

            if tag.startswith('/'):

                nome = tag[1:]

                if not pilha or pilha[-1] != nome:

                    return False

                pilha.pop()
```

```
            else:
                # tag de abertura
                pilha.append(tag)

            i = fechamento
        i += 1

    return len(pilha) == 0


# Programa principal
html = input("Digite o HTML a validar: ")

if validar_html(html):
    print("HTML válido! Todas as tags foram fechadas corretamente.")
else:
    print("HTML inválido! Há tags abertas ou fechamentos incorretos.")
```

5. **Fila – Atendimento de clientes** – Usa fila para simular entrada e saída de clientes.

```
from collections import deque

fila = deque()

while True:
```

```
print("\n==== MENU ====")
print("1 – Chegada de cliente")
print("2 – Atender cliente")
print("3 – Mostrar fila")
print("4 – Sair")

opc = input("Escolha uma opção: ")

if opc == "1":
    nome = input("Nome do cliente: ")
    fila.append(nome)
    print(f"Cliente '{nome}' entrou na fila.")

elif opc == "2":
    if fila:
        atendido = fila.popleft()
        print(f"Cliente '{atendido}' foi atendido.")
    else:
        print("A fila está vazia! Ninguém para atender.")

elif opc == "3":
    print("Fila atual:", list(fila) if fila else "Vazia")
```

```
    elif opc == "4":
        print("Encerrando...")
        break

    else:
        print("Opção inválida, tente novamente!")
```

6. **Ordenação – Bubble Sort invertido** – Ordene 8 números em ordem decrescente com Bubble Sort.

```
nums = []

for i in range(8):
    nums.append(float(input(f"Digite o {i+1}º número: ")))

# Bubble Sort invertido (decrescente)
for i in range(len(nums) - 1):
    for j in range(len(nums) - 1 - i):
        if nums[j] < nums[j+1]:
            nums[j], nums[j+1] = nums[j+1], nums[j]

print("\nVetor ordenado (decrescente):", nums)
```

7. **Busca – Binária em vetor** – Implemente busca binária e conte comparações realizadas.

```
def busca_binaria(vetor, x):
    inicio, fim = 0, len(vetor) - 1
    comparacoes = 0
```

```
    while inicio <= fim:
        meio = (inicio + fim) // 2
        comparacoes += 1

        if vetor[meio] == x:
            return meio, comparacoes
        elif vetor[meio] < x:
            inicio = meio + 1
        else:
            fim = meio - 1

    return -1, comparacoes

# exemplo
vetor = sorted([int(input("Número: ")) for _ in range(8)])
x = int(input("Buscar: "))

pos, comp = busca_binaria(vetor, x)

print("\nVetor:", vetor)
print("Posição encontrada:", pos)
print("Comparações feitas:", comp)
```

8. **Estrutura homogênea – Média de temperaturas** – Identifique o dia mais quente e o mais frio de 7 dias.

```
temps = [float(input(f"Temperatura do dia {i+1}: ")) for i in range(7)]



dia_quente = temps.index(max(temps)) + 1

dia_frio = temps.index(min(temps)) + 1



print("\nMédia:", sum(temps)/7)

print("Dia mais quente:", dia_quente, "-", max(temps), "°C")

print("Dia mais frio:", dia_frio, "-", min(temps), "°C")
```

9. **Estrutura heterogênea – Cadastro de alunos** – Guarde nome, idade e média final em dicionários e mostre alunos com média > 7.

```
for i in range(5):

    nome = input("Nome: ")

    idade = int(input("Idade: "))

    media = float(input("Média final: "))

    alunos.append({"nome": nome, "idade": idade, "media": media})



print("\nAlunos com média > 7:")

for a in alunos:

    if a["media"] > 7:

        print(a["nome"], "-", a["media"])
```

10. **Matriz + Busca – Posição de um número** – Mostre onde determinado número aparece em uma matriz 5x5.

```
matriz = []
```

```
for i in range(5):
    linha = [int(input(f"Valor [{i}][{j}]: ")) for j in range(5)]
    matriz.append(linha)

x = int(input("\nBuscar número: "))

print("\nPosições encontradas:")
for i in range(5):
    for j in range(5):
        if matriz[i][j] == x:
            print(f"({i}, {j})")
```

11. Soma recursiva de um vetor

```
def soma_rec(vetor, i=0):
    if i == len(vetor):
        return 0
    return vetor[i] + soma_rec(vetor, i+1)

vetor = [int(input("Número: ")) for _ in range(6)]

print("\nSoma:", soma_rec(vetor))
```

☑ **12. Ordenação de produtos por preço (Bubble Sort)**

```
produtos = []
```

```
for i in range(5):
    nome = input("Nome do produto: ")
    preco = float(input("Preço: "))
    produtos.append({"nome": nome, "preco": preco})

# bubble sort por preço (crescente)
for i in range(len(produtos) - 1):
    for j in range(len(produtos) - 1 - i):
        if produtos[j]["preco"] > produtos[j+1]["preco"]:
            produtos[j], produtos[j+1] = produtos[j+1], produtos[j]

print("\nProdutos ordenados por preço:")
for p in produtos:
    print(p["nome"], "-", p["preco"])
```

☑ 13. Vetor – inversão manual sem funções prontas

```
vet = [int(input("Número: ")) for _ in range(6)]

invertido = [0] * len(vet)

for i in range(len(vet)):
    invertido[i] = vet[len(vet) - 1 - i]
```

```
print("\nOriginal:", vet)

print("Invertido:", invertido)
```

☑ 14. Matriz 4x4 – linha e coluna com maior soma

```
matriz = []

for i in range(4):

    matriz.append([int(input(f"Valor [{i}][{j}]: ")) for j in range(4)])



somas_linhas = [sum(l) for l in matriz]

somas_colunas = [sum(matriz[i][j] for i in range(4)) for j in range(4)]



print("\nLinha com maior soma:", somas_linhas.index(max(somas_linhas)), "-",
max(somas_linhas))

print("Coluna com maior soma:", somas_colunas.index(max(somas_colunas)), "-",
max(somas_colunas))
```

☑ 15. Recursão – contar ocorrências de X na lista

```
def contar(vetor, x, i=0):

    if i == len(vetor):

        return 0

    return (1 if vetor[i] == x else 0) + contar(vetor, x, i+1)



vetor = [int(input("Número: ")) for _ in range(10)]
```

```
x = int(input("Valor a contar: "))


print("\nOcorrências:", contar(vetor, x))
```

☑ 16. Pilha – inverter frase caractere por caractere

```
frase = input("Frase: ")

pilha = []


for c in frase:
    pilha.append(c)


invertida = ""

while pilha:
    invertida += pilha.pop()


print("\nInvertida:", invertida)
```

☑ 17. Fila – sistema de senhas

```
from collections import deque


fila = deque()

senha_atual = 1
```

```
while True:
    print("\n1 - Gerar senha")
    print("2 - Chamar próxima")
    print("3 - Mostrar fila")
    print("4 - Sair")
    opc = input("Opção: ")

    if opc == "1":
        fila.append(senha_atual)
        print("Senha gerada:", senha_atual)
        senha_atual += 1

    elif opc == "2":
        if fila:
            print("Atendendo senha:", fila.popleft())
        else:
            print("Fila vazia!")

    elif opc == "3":
        print("Fila:", list(fila))

    elif opc == "4":
        break
```

☑ 18. Selection Sort – ordem crescente

```
vet = [int(input("Número: ")) for _ in range(8)]


for i in range(len(vet)):
    menor = i
    for j in range(i+1, len(vet)):
        if vet[j] < vet[menor]:
            menor = j
    vet[i], vet[menor] = vet[menor], vet[i]


print("\nOrdenado:", vet)
```

☑ 19. Busca linear aprimorada

```
vet = [int(input("Número: ")) for _ in range(8)]
x = int(input("Valor a buscar: "))


for i in range(len(vet)):
    if vet[i] == x:
        print("\nEncontrado na posição:", i)
        break
else:
    print("\nValor não encontrado.")
```

☑ 20. Média e desvio em relação à média

```
notas = [float(input("Nota: ")) for _ in range(6)]


media = sum(notas) / len(notas)


print("\nMédia:", media)

print("Desvios:")


for n in notas:

    print(f"{n} -> desvio = {n - media}")
```

☑ 21. Cadastro de veículos (ano > 2020 e preço < 80000)

```
veiculos = []


for i in range(5):

    modelo = input("Modelo: ")

    ano = int(input("Ano: "))

    preco = float(input("Preço: "))

    veiculos.append({"modelo": modelo, "ano": ano, "preco": preco})


print("\nVeículos filtrados:")

for v in veiculos:
```

```
    if v["ano"] > 2020 and v["preco"] < 80000:
        print(v)
```

☑ 22. Matriz transposta lado a lado

```
mat = []
for i in range(3):
    mat.append([int(input(f"Valor [{i}][{j}]: ")) for j in range(3)])

transposta = [[mat[j][i] for j in range(3)] for i in range(3)]

print("\nOriginal     Transposta")
for i in range(3):
    print(mat[i], "    ", transposta[i])
```

☑ 23. Recursão – soma de dígitos

```
def soma_digitos(n):
    n = abs(n)
    if n < 10:
        return n
    return n % 10 + soma_digitos(n // 10)

num = int(input("Número: "))
print("\nSoma dos dígitos:", soma_digitos(num))
```

☑ **24. Lista de funcionários – ordenar por salário decrescente (Bubble Sort)**

```
func = []


for i in range(5):
    nome = input("Nome: ")
    salario = float(input("Salário: "))
    func.append({"nome": nome, "salario": salario})


# bubble sort decrescente
for i in range(len(func) - 1):
    for j in range(len(func) - 1 - i):
        if func[j]["salario"] < func[j+1]["salario"]:
            func[j], func[j+1] = func[j+1], func[j]


print("\nFuncionários ordenados por salário (decrescente):")
for f in func:
    print(f["nome"], "-", f["salario"])
```

■ NÍVEL AVANÇADO – Estruturas Dinâmicas e Algoritmos Complexos

1. Recursão – Torre de Hanói (mostra passo a passo)

```
def hanoi(n, origem, destino, auxiliar):
    if n == 1:
        print(f"Mover disco 1 de {origem} para {destino}")
```

```
        return
    hanoi(n-1, origem, auxiliar, destino)
    print(f"Mover disco {n} de {origem} para {destino}")
    hanoi(n-1, auxiliar, destino, origem)

n = int(input("Número de discos: "))
hanoi(n, "A", "C", "B")
```

2. Recursão – Permutações de uma lista (gera todas)

```
def permutacoes(lst, inicio=0):
    if inicio == len(lst) - 1:
        print(lst)
        return
    for i in range(inicio, len(lst)):
        lst[inicio], lst[i] = lst[i], lst[inicio]
        permutacoes(lst, inicio + 1)
        lst[inicio], lst[i] = lst[i], lst[inicio]  # backtrack

arr = input("Elementos separados por espaço: ").split()
permutacoes(arr)
```

3. Busca – Ternária (vetor ordenado)

```
def busca_ternaria(vetor, x):
    l, r = 0, len(vetor) - 1
    comparacoes = 0
```

```
    while l <= r:
        t1 = l + (r - l) // 3
        t2 = r - (r - l) // 3
        comparacoes += 1
        if vetor[t1] == x:
            return t1, comparacoes
        comparacoes += 1
        if vetor[t2] == x:
            return t2, comparacoes
        if x < vetor[t1]:
            r = t1 - 1
        elif x > vetor[t2]:
            l = t2 + 1
        else:
            l = t1 + 1
            r = t2 - 1
    return -1, comparacoes


vet = sorted([int(x) for x in input("Digite valores (espaço): ").split()])
x = int(input("Valor a buscar: "))
pos, comp = busca_ternaria(vet, x)
print("Vetor:", vet)
print("Posição:", pos, "Comparações:", comp)
```

4. Ordenação – Merge Sort (divisão e conquista)

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    i = j = 0
    merged = []
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i]); i += 1
        else:
            merged.append(right[j]); j += 1
    merged.extend(left[i:]); merged.extend(right[j:])
    return merged


arr = [int(x) for x in input("Elementos: ").split()]
print("Ordenado:", merge_sort(arr))
```

5. Ordenação – Quick Sort (in-place)

```
def quick_sort(arr, low=0, high=None):
    if high is None:
        high = len(arr) - 1
```

```
    if low < high:
        p = partition(arr, low, high)
        quick_sort(arr, low, p - 1)
        quick_sort(arr, p + 1, high)

def partition(arr, low, high):
    pivot = arr[high]
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i+1], arr[high] = arr[high], arr[i+1]
    return i + 1

arr = [int(x) for x in input("Elementos: ").split()]
quick_sort(arr)
print("Ordenado:", arr)
```

6. Estrutura Encadeada – Lista ligada simples (classe)

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None
```

```
class LinkedList:
    def __init__(self):
        self.head = None

    def inserir_inicio(self, val):
        novo = Node(val)
        novo.next = self.head
        self.head = novo

    def remover(self, val):
        prev = None
        cur = self.head
        while cur:
            if cur.val == val:
                if prev:
                    prev.next = cur.next
                else:
                    self.head = cur.next
                return True
            prev, cur = cur, cur.next
        return False
```

```
    def mostrar(self):
        cur = self.head
        vals = []
        while cur:
            vals.append(cur.val)
            cur = cur.next
        print(" -> ".join(map(str, vals)))

ll = LinkedList()
ll.inserir_inicio(3); ll.inserir_inicio(5); ll.inserir_inicio(7)
ll.mostrar()
print("Removendo 5:", ll.remover(5))
ll.mostrar()
```

7. Estrutura Encadeada – Pilha e fila com classes (POO)

```
class Pilha:
    def __init__(self): self._dados = []
    def push(self, x): self._dados.append(x)
    def pop(self): return self._dados.pop() if self._dados else None
    def topo(self): return self._dados[-1] if self._dados else None
    def vazia(self): return len(self._dados) == 0

class Fila:
    def __init__(self): from collections import deque; self._dados = deque()
```

```
    def enfileirar(self, x): self._dados.append(x)
    def desenfileirar(self): return self._dados.popleft() if self._dados else None
    def vazia(self): return len(self._dados) == 0

p = Pilha(); p.push(1); p.push(2); print(p.pop())
f = Fila(); f.enfileirar("A"); f.enfileirar("B"); print(f.desenfileirar())
```

8. Árvore binária – Inserção e busca (BST)

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = self.right = None

def insert(root, key):
    if root is None:
        return Node(key)
    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root

def search(root, key):
    if not root: return False
```

```
    if root.key == key: return True
    return search(root.left, key) if key < root.key else search(root.right, key)

keys = [int(x) for x in input("Chaves para inserir: ").split()]
r = None
for k in keys: r = insert(r, k)
x = int(input("Buscar: "))
print("Encontrado:" , search(r, x))
```

9. Grafo – Lista de adjacência (dicionário)

```
def adicionar_aresta(g, u, v, bidirecional=True):
    g.setdefault(u, []).append(v)
    if bidirecional:
        g.setdefault(v, []).append(u)

g = {}
adicionar_aresta(g, "A", "B")
adicionar_aresta(g, "A", "C")
adicionar_aresta(g, "B", "D")
print("Grafo (adjacência):")
for nodo, viz in g.items():
    print(nodo, "->", viz)
```

10. Backtracking – Caminho em matriz 0/1 (canto superior esquerdo ao inferior direito)

```
def caminho(matriz):
```

```
    n = len(matriz)
    visit = [[False]*n for _ in range(n)]
    path = []

    def back(i, j):
        if i < 0 or j < 0 or i >= n or j >= n: return False
        if matriz[i][j] == 1 or visit[i][j]: return False
        path.append((i,j)); visit[i][j] = True
        if i == n-1 and j == n-1: return True
        for di, dj in [(1,0),(0,1),(-1,0),(0,-1)]:
            if back(i+di, j+dj): return True
        path.pop(); visit[i][j] = False
        return False

    if back(0,0): return path
    return None

mat = []
n = int(input("Tamanho n da matriz (ex: 5): "))
print("Digite linhas com 0 (livre) e 1 (bloqueado), separados por espaço:")
for _ in range(n):
    mat.append([int(x) for x in input().split()])
res = caminho(mat)
```

```
print("Caminho encontrado:" , res)
```

11. Manipulação de arquivos – Cadastro de produtos (.json)

```
import json


def salvar(produtos, arquivo="produtos.json"):
    with open(arquivo, "w", encoding="utf-8") as f:
        json.dump(produtos, f, ensure_ascii=False, indent=2)


def carregar(arquivo="produtos.json"):
    try:
        with open(arquivo, "r", encoding="utf-8") as f:
            return json.load(f)
    except FileNotFoundError:
        return []


produtos = carregar()
produtos.append({"nome":"Banana","preco":2.5})
salvar(produtos)
print("Produtos salvos. Conteúdo atual:", carregar())
```

12. Tratamento de exceções – Divisão segura

```
def divisao_segura(a, b):
    try:
        return a / b
```

```
    except ZeroDivisionError:
        print("Erro: divisão por zero.")
        return None
    except TypeError:
        print("Erro: tipos inválidos.")
        return None

a = float(input("A: ")); b = float(input("B: "))
print("Resultado:", divisao_segura(a, b))
```

13. POO + Estrutura heterogênea – Sistema de cadastro (alunos/professores/disciplinas)

```
class Pessoa:
    def __init__(self, nome, idade):
        self.nome = nome; self.idade = idade

class Aluno(Pessoa):
    def __init__(self, nome, idade, matricula):
        super().__init__(nome, idade)
        self.matricula = matricula
        self.notas = []

    def media(self):
        return sum(self.notas)/len(self.notas) if self.notas else 0
```

```
class Professor(Pessoa):
    def __init__(self, nome, idade, area):
        super().__init__(nome, idade)
        self.area = area

class Disciplina:
    def __init__(self, nome, professor):
        self.nome = nome
        self.professor = professor
        self.alunos = []

# exemplo de uso
prof = Professor("Ana", 40, "Matemática")
d = Disciplina("Álgebra", prof)
a = Aluno("João", 18, "2025A1"); a.notas = [7,8,9]
d.alunos.append(a)
print("Disciplina:", d.nome, "Professor:", d.professor.nome)
print("Alunos aprovados (média>7):", [al.nome for al in d.alunos if al.media()>7])
```

14. Banco de dados em memória – registros com dicionários e filtros

```
registros = [
    {"id":1, "nome":"Produto A", "preco":50},
    {"id":2, "nome":"B", "preco":120},
    {"id":3, "nome":"C", "preco":30}
```

```
]

# filtro preço < 100
filt = [r for r in registros if r["preco"] < 100]
# ordenar por preco desc
ordenado = sorted(registros, key=lambda x: x["preco"], reverse=True)
print("Filtrados:", filt)
print("Ordenados:", ordenado)
15. Mini sistema – Simulação de supermercado (cadastro, compra, total e troco)
produtos = {}
def cadastrar(codigo, nome, preco): produtos[codigo] = {"nome":nome, "preco":preco}
cadastrar("001","Pão",3.5); cadastrar("002","Leite",5.0)

carrinho = []
while True:
    cod = input("Código do produto (enter para terminar): ")
    if not cod: break
    if cod in produtos:
        qtd = int(input("Quantidade: "))
        carrinho.append((cod, qtd))
    else:
        print("Produto não existe.")
```

```
total = sum(produtos[c]["preco"] * q for c,q in carrinho)

print("Total a pagar:", total)

pago = float(input("Valor pago: "))

print("Troco:", pago - total if pago >= total else "Pagamento insuficiente")
```

16. Mini sistema – Agenda de contatos (cadastrar, editar, salvar em arquivo)

```
import json

ARQ = "contatos.json"


def carregar():

    try:

        with open(ARQ,"r",encoding="utf-8") as f: return json.load(f)

    except FileNotFoundError:

        return []


def salvar(contatos):

    with open(ARQ,"w",encoding="utf-8") as f:
json.dump(contatos,f,ensure_ascii=False,indent=2)


contatos = carregar()

contatos.append({"nome":"Maria","telefone":"(11)9999-9999"})

salvar(contatos)

print("Contatos:", carregar())
```

17. Desafio final – Integração total (esqueleto de sistema simples)

```
# Exemplo: mini-biblioteca integrando estruturas
```

```
# - livros (lista de dicionários)

# - busca (linear)

# - empréstimos (fila)

# - salvar em arquivo


import json

from collections import deque


ARQ = "biblioteca.json"

def carregar():

    try:

        with open(ARQ,"r",encoding="utf-8") as f: return json.load(f)

    except FileNotFoundError: return {"livros": [], "emprestimos": []}


def salvar(db):

    with open(ARQ,"w",encoding="utf-8") as f:
json.dump(db,f,ensure_ascii=False,indent=2)


db = carregar()

# operações

def cadastrar_livro(titulo, autor):

    db["livros"].append({"id": len(db["livros"])+1, "titulo":titulo, "autor":autor, "disponivel":
True})

    salvar(db)
```

```
def buscar_titulo(substr):
    return [l for l in db["livros"] if substr.lower() in l["titulo"].lower()]


# simular empréstimo com fila (apenas exemplo simples)
fila = deque()
fila.append({"usuario":"Alice","livro_id":1})
print("Busca por 'Python':", buscar_titulo("Python"))
salvar(db)
```