Programação de Computadores I – BCC701

Conteúdo Teórico em Linguagem Python

Módulo 7 – Estruturas Homogêneas

2020/1

Elaborado pela Comissão de Unificação da Disciplina BCC701,
com a colaboração de professores e estagiários docentes
http://www.decom.ufop.br/bcc701/

DECOM – Departamento de Computação
ICEB – Instituto de Ciências Exatas e Biológicas
UFOP – Universidade Federal de Ouro Preto

Sumário

MÓDULO 7

Estruturas Homogêneas

Em módulos anteriores, aprendemos quais tipos de dados podem ser armazenados e como utilizar esses valores em **Python**. Alguns dos tipos de dados que aprendemos a usar são: Inteiros (**int**); Números de ponto flutuante (**float**); texto (**string**); Lógicos (**boolean**).

Aprendemos também a armazenar esses tipos de dados dentro de variáveis, conforme a nossa demanda, como por exemplo:

```
1  # Declarações de Variáveis
2  nome = 'Pedro'
3  idade = 15
4  matriculado = True
```

Contudo, algumas vezes queremos armazenar toda uma coleção de dados na memória durante a execução de um programa. Suponha que em um programa, você queira registrar o valor da conta de luz dos 15 moradores do seu prédio. Somente com as ferramentas ensinadas até o momento, seriam necessárias 15 variáveis diferentes, cada uma delas contendo o valor da conta de luz de cada morador:

```
1  conta_m0 = input('Digite o valor da conta: ')
2  conta_m1 = input('Digite o valor da conta: ')
3  conta_m2 = input('Digite o valor da conta: ')
4  conta_m3 = input('Digite o valor da conta: ')
5  conta_m4 = input('Digite o valor da conta: ')
6  ...
7  conta_m11 = input('Digite o valor da conta: ')
8  conta_m12 = input('Digite o valor da conta: ')
9  conta_m13 = input('Digite o valor da conta: ')
10 conta_m14 = input('Digite o valor da conta: ')
```

Além de ocupar muito espaço e tornar o código difícil de ler e entender, essa solução dificulta a capacidade de realizar operações sobre todos os dados em conjunto. Se quisermos compreender a média do valor das contas de energia, por exemplo, precisaremos escrever manualmente todas as operações.

Para facilitar o agrupamento de dados durante a execução do programa, existem estruturas chamadas **Vetores** e **Matrizes**.

- Os **Vetores** são estruturas de uma dimensão, contendo uma lista de dados de determinado tipo.
- As **Matrizes** são estruturas de duas dimensões, que armazenam uma tabela endereçada contendo os dados de determinado tipo.

Nesse módulo, compreenderemos como criar e utilizar **vetores** e **matrizes**.

7.1 Vetores

Uma das maneiras mais comuns que utilizamos para agrupar dados é na forma de listas. Fazemos listas de compras nos supermercados, listas de afazeres, listas de objetos para colocar na mala. Da mesma maneira, uma das formas mais simples de armazenar dados de um tipo é através de listas, aqui denominadas **Vetores**. Conforme definido anteriormente, **vetor** é um tipo de dado **homogêneo**, portanto, todos os dados armazenados devem ser do mesmo tipo. Na Figura 1 representamos uma lista de nomes de produtos de um supermercado agrupados em um vetor.

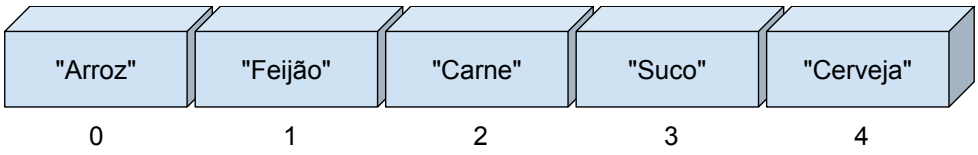

Figura 1: Representação ilustrativa de um vetor de *strings*.

Nessas estruturas de dados, cada "caixa" pode receber um valor, como se fosse uma variável. Assim como as casas de uma rua ou caixas de correio de um prédio, os contêineres de um vetor são **endereçados** através de um número. Em **Python**, o primeiro endereço dentro de um vetor é o $zero$ (0), seguindo sequencialmente com acréscimo de 1 unidade, ou seja, os endereços são valores inteiros a partir de $zero$. Esses endereços são chamados **índices** dos elementos do vetor. Sendo assim, podemos expressar um vetor A com n elementos da seguinte forma:

$$A_n = \begin{bmatrix} a_0 & a_1 & a_2 & .. & a_{n-1} \end{bmatrix}$$

onde: a_i representa o elemento de índice **i**, armazenado no vetor.

7.1.1 Sintaxe básica

Agora que compreendemos o conceito da estrutura de dados **vetor**, precisamos entender como criá-los em **Python**. Em **Python** existem muitas estruturas de dados compostas, buscamos uma estrutura com sintaxe simples para representar o conceito de vetor em nosso curso, a estrutura de dado denominada **lista**. Em uma lista no **Python** os elementos armazenados podem ser de qualquer tipo, permitindo misturar elementos de variados tipos. Porém, como os elementos podem ser de tipos diferentes, esta estrutura do **Python** não é conceitualmente um **vetor**. Sendo assim, nos referiremos às **listas** em **Python** como **vetores** toda vez que seus elementos forem sempre de um mesmo tipo de dados. Em nossos programas, trataremos as variáveis **lista** como **vetores**, garantindo sempre armazenar valores de um único tipo.

Um vetor é declarado de maneira similar a uma variável, porém com colchetes indicando a coleção de dados. Vamos entender isso melhor. Um vetor de inteiros, por exemplo, pode ser declarado já contendo os dados delimitados por colchetes e separados por vírgulas:

```
1 # Vetor de valores numéricos
2 num_vector = [ 2, 4.3, 6, 8.2, 10.8 ]
3
4 # Vetor de booleanos
5 bool_vector = [ True, False, True, True ]
6
7 # Vetor de strings
8 words_vector = [ 'alpha', 'bravo', 'celta' ]
```

Observe que tivemos a preocupação de definir sempre elementos de um mesmo tipo de dados, isso está garantindo que, conceitualmente, todas as variáveis do programa anterior sejam denominadas **vetores** no contexto de nossa disciplina.

Um vetor pode inclusive ser declarado vazio, para ser preenchido posteriormente, conforme o exemplo a seguir. Este recurso será muito utilizado futuramente.

```
1 # Vetor vazio
2 empty_vector = [ ]
```

A seguir, vamos entender melhor como criar, modificar e adicionar dados a um vetor.

7.1.2 Operações com vetores

Na seção anterior, aprendemos a sintaxe básica dos vetores. Agora, aprenderemos a utilizar esses objetos. As principais operações com vetores são:

- Criar um vetor;
- Inserir elementos em um vetor;
- Acessar/modificar elementos de um vetor;
- Remover elementos de um vetor;
- Descobrir a dimensão de um vetor.

Criando um vetor: Conforme demonstramos antes, é possível criar um vetor já com alguns valores alocados ou mesmo um vetor vazio.

```
1 vetor_vazio = [ ]
2 vetor_inteiros = [ 1, 2, 3, 4 ]
```

Inserindo elementos em um vetor: Para inserir elementos em um vetor, utilizamos a função **append**. Veja abaixo como utilizar:

```
1 V = []
2 # Inserindo o número 1 no vetor
3 V.append(1)
```

Acessando/modificando elementos de um vetor: Um elemento de um vetor tem o comportamento semelhante a uma variável simples de qualquer tipo. Para tratar de um elemento individualmente, basta utilizar o índice correspondente ao endereço que se deseja acessar ou modificar. Veja abaixo como funciona:

```
1 V = [ 1, 2, 3, 4, 5 ]
2 print(f'V no endereço 2: {V[2]}')
3 # O programa vai imprimir:
4 # V no endereço 2: 3
5 V[0] = V[0] + 1
6 # O vetor V passa conter: [ 2, 2, 3, 4, 5 ]
```

Removendo elementos de um vetor: Para remover elementos de um vetor, utilizaremos a função pop. Essa função recebe o índice do elemento a ser removido. Além de remover o elemento da posição informada, ela retorna o valor armazenado. Lembrando que o endereçamento de vetores começa no zero.

```
1 V = [ 1, 2, 3, 4, 5 ]
2 x = V.pop(1)
3 # O vetor V passa conter: [ 1, 3, 4, 5 ]
4 print(x) # imprime o valor 2
```

Descobrindo a dimensão de um vetor: Para descobrir a dimensão de um vetor, utilizamos a função **len**. Ela retorna o tamanho do vetor passado como argumento. Veja como utilizar abaixo.

```
1 V = [ 1, 2, 3, 4, 5 ]
2 tamanho = len(V)
3 # tamanho = 5
```

Exemplo 1 (média de N valores):

No começo desse capítulo falamos sobre o problema da média de vários valores. Uma das maneiras de organizar esses dados é dentro de vetores. Vejamos como é possível escrever um programa que calcula a média de um conjunto de valores armazenados em um vetor.

Primeiramente, organizamos nosso problema em seções de entrada, processamento e saída.

1. Entrada:
   (a) Valores para calcular a média;
2. Processamento:
   (a) Somar valores;
   (b) Dividir pelo tamanho do vetor;
3. Saída:
   (a) Valor da média;

A seguir, o programa em **Python**:

```
valores = [ ] # inicialmente vazio (ainda não lemos nenhum número)
run = True # para entrar no laço pelo menos uma vez
#Entrada
while (run):
    continuar = input('Digite o próximo valor (Para terminar digite "N
       " ou "n"): ')
    if (continuar == "N" or continuar == "n"):
        run = False # deve parar de ler valores
    else:
        val = float(inp) # convertendo o valor informado
        valores.append(val) # adicionando o valor no vetor
#Processamento
soma = 0 # valor inicial, ainda não processamos nenhum valor
for i in range(len(valores)):
    soma = soma + valores[i] # somatório cumulativo
media = soma / len(valores)
#Saída
print(f'O valor da média é: {media:%.2f}')
```

Exemplo 2 (elementos de uma função linear):

Como sabemos, uma função linear é dada por uma relação do tipo:

$$f(x) = ax + b$$

Podemos usar dois vetores, um x e um y para representar pares de pontos pertencentes a essa função. Vamos analisar então como fazer um programa que toma os elementos a e b como entrada, além de um número de pares desejado, e gera pares de pontos positivos pertencentes a essa função, imprimindo os valores ao final.

A estrutura do nosso programa é:

1. Entrada:
   (a) Número de pares desejados;
   (b) Valores a e b;
2. Processamento:
   (a) Gerar vetor X;
   (b) Utilizando a, b e X, gerar o vetor Y;
3. Saída:
   (a) Imprimir os pares de pontos usando X e Y.

A seguir, o programa em **Python**:

```python
# Entrada
n = int(input('Digite o numero de pares desejado: '))
a = float(input('Digite o valor do termo "a": '))
b = float(input('Digite o valor do termo "b": '))
# Processamento
X = []
Y = []
for i in range(n):
    X.append(i)
    Y.append(a*i+b)
# Saída
print('Pares de pontos:')
for i in range(n):
    print(f'{X[i]} {Y[i]}')
```

7.1.3 Exercícios propostos – Vetores

Exercício 1

Utilizando vetores, imprimir uma sequência de pontos indicando a distância de queda livre de um corpo, dada uma sequência de tempo de $t = 0$ até um valor dado como entrada pelo usuário. Os valores impressos devem ter uma precisão de uma casa decimal.

Fórmulas:

- $g = 9.807 m/s^2$
- $d = \frac{gt^2}{2}$

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
```

Exemplo 2:

```
Digite o tempo final: 6
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
4 78.5
5 122.6
6 176.5
```

Exercício 2

Uma função de segundo grau é dada pela fórmula:

$$f(x) = ax^2 + bx + c$$

Dadas como entradas:

- Valores a, b e c
- Ponto x_0 inicial
- Ponto x_f final

Criar dois vetores com os dados dos valores da função nesses pontos.
Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -3
Digite o valor final de x: 3
x f(x)
-3 13
-2 6
-1 3
0 4
1 9
2 18
3 31
```

Exemplo 2:

```
Digite o valor de "a": 2
Digite o valor de "b": 0
Digite o valor de "c": 0
Digite o valor inicial de x: -5
Digite o valor final de x: 2
x f(x)
-5 50
-4 32
-3 18
-2 8
-1 2
0 0
1 2
2 8
```

Exercício 3

Implemente um programa que realize a entrada de dados de um vetor. Primeiro o usuário fornece a quantidade de elementos do vetor, em seguida, fornece um valor numérico para cada elemento. Após preenchido o vetor, os valores armazenados no mesmo devem ser impressos na tela com uma precisão de uma casa decimal.

Exemplos de execução:

Exemplo 1:

```
Informe a dimensão do vetor: 3
Valor do elemento 0: 3.5
Valor do elemento 1: 7
Valor do elemento 2: 10.65
Elementos do vetor:
3.5
7.0
10.7
```

Exemplo 2:

```
Informe a dimensão do vetor: 5
Valor do elemento 0: 6.92
Valor do elemento 1: 4
Valor do elemento 2: 5.271
Valor do elemento 3: 2.34
Valor do elemento 4: 2.319
Elementos do vetor:
6.9
4.0
5.3
2.3
2.3
```

7.2 Matrizes

Na última seção, aprendemos como criar, manipular e utilizar vetores para os mais diversos fins. Nessa seção, aprenderemos a utilizar **matrizes**. Como uma **estrutura homogênea**, uma matriz é uma estrutura de n linhas por m colunas, onde cada elemento armazena um dado de um tipo. Sendo assim, podemos expressar uma matriz A, com dimensão $n \times m$ da seguinte forma:

$$A_{n\times m} = \begin{bmatrix} a_{0,1} & a_{0,1} & a_{0,2} & .. & a_{0,m-1} \\ a_{1,1} & a_{1,1} & a_{1,2} & .. & a_{1,m-1} \\ a_{2,1} & a_{2,1} & a_{2,2} & .. & a_{2,m-1} \\ : & : & : & .. & : \\ a_{n-1,1} & a_{n-1,1} & a_{n-1,2} & .. & a_{n-1,m-1} \end{bmatrix}$$

Podemos interpretar as matrizes como "vetores de vetores". Cada uma das n linhas da matriz corresponderá a um vetor contendo m elementos. Utilizaremos essa interpretação para ilustrar a maneira de criar, acessar e modificar elementos de uma matriz. Também podemos dizer que um **vetor** possui **uma**

dimensão, enquanto uma **matriz** possui **duas dimensões**. As linguagens de programação permitem criar estruturas com mais do que duas dimensões, porém, neste curso nós exploraremos apenas uma e duas dimensões.

7.2.1 Sintaxe básica

De forma similar aos vetores, existem variadas estruturas de dados compostas capazes de representar **matrizes** em **Python**. Novamente, queremos escolher uma estrutura com sintaxe simples. Sendo assim, utilizaremos a mesma estratégia utilizada nos vetores e representaremos uma matriz em **Python** através do tipo **lista**, garantindo que sempre teremos elementos de um mesmo tipo, a diferença é que agora também garantiremos que ela tenha duas dimensões.

Consideraremos uma matriz com tamanho predefinido antes de preencher seus valores. Assim, a interpretação de uma matriz é bem similar à de uma matriz matemática. Um elemento qualquer de uma matriz, em matemática, é representado por $a_{lin,col}$, onde lin representa a linha e col representa a coluna. De maneira similar, um elemento qualquer de uma matriz é representado por:

```
1 # Representação computacional de um elemento de uma matriz
2 A[lin][col]
```

Conforme dito, interpretaremos cada linha da matriz como um vetor. Isso coincide com a representação computacional. Veja abaixo:

```
1 # Matriz
2 A
3 # Linha de uma matriz (resulta em um vetor)
4 A[lin]
5 # Elemento de uma matriz (resulta o valor de um elemento específico)
6 A[lin][col]
```

A seguir, vamos entender melhor como criar, modificar e adicionar dados a uma matriz.

7.2.2 Operações com matrizes

Na seção anterior, aprendemos a sintaxe básica das matrizes. Agora, aprenderemos a utilizar esses objetos. As principais operações com matrizes são:

- Criar uma matriz;
- Acessar/modificar elementos de uma matriz;
- Descobrir a dimensão de uma matriz.

Criando uma matriz: **Python** é uma linguagem muito flexível. Para simplificar, trataremos a criação de matrizes a partir de duas estratégias: Criando de maneira fixa ou criando a partir de entradas do usuário.

Para criar uma matriz de maneira fixa, existem dois caminhos. Primeiramente, é possível escrever todos os elementos previamente. Veja a seguir:

```
# Criando uma matriz
A = [ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
# Matriz resultante:
# 1 2 3
# 4 5 6
# 7 8 9
```

Também é possível criar uma matriz usando uma rotina de código, a partir de um valor fixo ou dado pelo usuário com linhas e colunas. Veja a seguir:

```
# Criando matrizes com rotinas:
linhas = 3
colunas = 3
#Criando matriz de zeros
matriz = [ ] # uma lista vazia para representar a matriz
for lin in range(linhas):
    linha = [ ] # uma lista vazia para representar a linha
    for col in range(colunas):
        linha.append(0) # preenchendo valor da coluna
    matriz.append(linha) # colocando a linha na matriz
```

Na matriz anterior, tanto as dimensões quanto os valores poderiam ser obtidos através de entradas do usuário, com a função **input**. Veja a seguir:

```
# Criando uma matriz com entradas do usuário
linhas = int(input('Quantidade de linhas: '))
colunas = int(input('Quantidade de colunas: '))
matriz = [ ]
for lin in range(linhas):
    linha = [ ]
    for col in range(colunas):
        elemento = float(input(f'Digite o elemento {lin},{col}: '))
        linha.append(elemento)
    matriz.append(linha)
```

Acessando e modificando elementos em uma matriz: Os elementos de uma matriz podem ser acessados e modificados de maneira similar aos dos vetores. Para acessar um elemento, basta utilizar seus índices l e c entre chaves. Veja abaixo:

```
# Acessando o elemento [lin, col] da matriz A para diversos fins
val = A[lin][col]
print(A[lin][col])
result = 5 + A[lin][col]
```

Um elemento qualquer pode ser modificado de maneira idêntica à de uma variável, assim como nos vetores. Veja a seguir:

```
1 # Modificando o valor do elemento [i,j] da matriz A
2 A[i][j] = A[i][j] + 3
3 A[i][j] = A[i][j] * 2
4 A[i][j] = 5
```

Descobrir as dimensões das matrizes Descobrir as dimensões de uma matriz é um pouco diferente do que para os vetores, já que ela tem duas dimensões. O uso da função `len(A)` retorna a quantidade de linhas da matriz A. Para obter a quantidade de colunas, aplicar **len** a uma linha qualquer da própria matriz.

```
1 # Número de linhas
2 len(matriz)
3 #Número de colunas
4 len(matriz[0])
```

Observe que, se a matriz estiver vazia, a linha 0 não existe, portanto, `len(matriz[0])` resultaria em erro. Sendo assim, é prudente realizar este comando apenas quando a quantidade de linhas da matriz for diferente de $zero$, ou seja, `len(matriz) != 0` (em uma decisão).

7.2.3 Exercícios propostos – Matrizes

Exercício 4

Repetir a atividade do Exercício 1, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
```

Exemplo 2:

```
Digite o tempo final: 6
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
[4, 78.456]
[5, 122.5875]
[6, 176.526]
```

Exercício 5

Repetir a atividade do Exercício 2, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -5
Digite o valor final de x: 5

['x', 'f(x)']
[-5, 39]
[-4, 24]
[-3, 13]
[-2, 6]
[-1, 3]
[0, 4]
[1, 9]
[2, 18]
[3, 31]
[4, 48]
[5, 69]
```

Exemplo 2:

```
Digite o valor de "a": 4
Digite o valor de "b": 0
Digite o valor de "c": -4
Digite o valor inicial de x: -3
Digite o valor final de x: 6

['x', 'f(x)']
[-3, 32]
[-2, 12]
[-1, 0]
[0, -4]
[1, 0]
[2, 12]
[3, 32]
[4, 60]
[5, 96]
[6, 140]
```

Exercício 6

Fazer um programa que gere uma matriz quadrada de dimensão n, preenchida em todas as posições por um valor v. Ambos n e v são valores informados pelo usuário, como entradas. Imprimir o conteúdo da matriz ao final do programa.

Exemplos de execução:

Exemplo 1:

```
Digite a dimensão da matriz: 4
Digite o valor para preencher os campos: 2
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
```

Exemplo 2:

```
Digite a dimensão da matriz: 6
Digite o valor para preencher os campos: -3
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
```

Exercício 7

Implemente um programa que realize a entrada de dados de uma matriz. Primeiro o usuário fornece as dimensões da matriz, em seguida, fornece um valor textual para cada elemento. Após preenchida a matriz, o usuário fornece uma linha e coluna, e o programa imprime o valor correspondente a estes índices. Considere que os dados informados são sempre válidos.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de linhas: 2
Informe a quantidade de colunas: 3
Valor do elemento 0, 0: Sardinha
Valor do elemento 0, 1: Salame
Valor do elemento 0, 2: Picanha
Valor do elemento 1, 0: Coca-cola
Valor do elemento 1, 1: Guaraná
Valor do elemento 1, 2: Pipoca
Informe uma linha: 1
Informe uma coluna: 2
O valor armazenado em [1,2] é Pipoca
```

7.3 Construindo uma biblioteca de funções úteis

Podemos observar nos exemplos e exercícios sobre vetores e matrizes, apresentados anteriormente, que muitas tarefas serão sempre iguais. Alguns exemplos: criar um vetor ou matriz com valores padrão ou valores definidos pelo usuário e imprimir os valores de um vetor ou matriz. Podemos criar algumas funções para compor uma biblioteca padrão, e toda vez que precisarmos realizar uma destas tarefas, basta chamar a função. A seguir, apresentamos algumas funções úteis que nos ajudarão a criar programas mais rapidamente. Depois apresentamos exemplos de utilização da biblioteca criada.

7.3.1 Criando um vetor com valor padrão

Para esta função, consideraremos como argumentos de entrada a *quantidade de elementos* e o *valor padrão* a ser atribuído a todos eles. Basicamente, a função precisa criar um vetor vazio e, em seguida, repetir a operação de adicionar um novo elemento com o *valor padrão*, ao final, o vetor criado deve ser retornado. O código é apresentado a seguir.

```
1 def criarVetor(qtdElementos, valorPadrao):
2     vetor = [ ]
3     for i in range(qtdElementos):
4         vetor.append(valorPadrao)
5     return vetor
```

Dada a flexibilidade proporcionada pelo **Python**, não é necessário se preocupar com o tipo do *valor padrão*, basta passar o argumento com o valor desejado, independente de seu tipo.

7.3.2 Criando uma matriz com valor padrão

De forma similar, podemos criar uma matriz com um valor padrão, o código fica um pouco diferente, mas a lógica é praticamente a mesma, veja a seguir.

```
1 def criarMatriz(qtdLinhas, qtdColunas, valorPadrao):
2     matriz = [ ]
3     for lin in range(qtdLinhas):
4         linha = [ ]
5         for col in range(qtdColunas):
6             linha.append(valorPadrao)
7         matriz.append(linha)
8     return matriz
```

7.3.3 Criando um vetor com valores de uma string

Quando fazemos a entrada de dados de um vetor lendo o valor de cada elemento, são feitas várias entradas do usuário. Em termos práticos, é uma tarefa enfadonha para o usuário fazer várias entradas. Podemos oferecer a possibilidade do usuário fornecer todos os valores a partir de uma única entrada, estabelecendo para isso um padrão. Em nosso caso, separaremos os elementos com um caractere específico, a vírgula `","`. Neste caso, todos os elementos serão fornecidos em uma **string**, obtida em um único **input**, que precisará ser tratada para preencher o vetor. Apresentamos o código a seguir e logo depois explicamos mais detalhadamente seu funcionamento.

```
1  def preencherVetor(valores, tipo):
2      vetor = [ ]
3      valores = valores.split(',')
4      for i in range(len(valores)):
5          if tipo == 'int':
6              valor = int(valores[i].strip())
7          elif tipo == 'float':
8              valor = float(valores[i].strip())
9          else:
10             valor = valores[i].strip()
11         vetor.append(valor)
12     return vetor
```

Nossa função criará um novo vetor contendo os elementos passados na **string** de entrada `valores`, convertendo os valores conforme o `tipo`, passado como segundo argumento.

Como todos os valores são passados em uma única **string**, separados pela vírgula, é necessário "dividi-los" de acordo com o caractere `","`. Isto é feito com o uso da função **split**. Porém, o uso desta função é diferente daquilo que estamos acostumados, ela funciona sobre um valor textual, com o uso de um ponto separando o **valor** da **função**, que receberá como argumento o texto utilizado para fazer a separação da **string**: `valores.split(',')`, onde a **string** armazenada na variável `valores` será dividida, tendo como referência a vírgula, resultando em um vetor com os valores divididos. Por exemplo, para a **string** `valores = "1, 2, 6, 5"`, o resultado de `valores.split(',')` será o vetor `[ "1", " 2", " 6", " 5", ]`. Observe que, antes da linha 3, o valor da variável `valores` será uma **string**, e depois da linha 3 ela passará a armazenar um vetor de **strings**, resultado do **split**.

Em seguida, o laço for vai processar cada valor **string** contido no vetor `valores`. De acordo com o `tipo` informado na chamada da função, será feita uma conversão para **int** ou **float**, ou então será mantido o tipo **string**. Agora, usamos a função **strip**, também aplicada a uma **string** de forma similar à **split**, mas o resultado é a elimininação dos espaços à direita e à esquerda da **string**, por exemplo, para o segundo elemento o valor `" 2"` resultará em `"2"`.

Concluindo o laço, o valor processado é inserido no vetor que será retornado ao final.

7.3.4 Criando uma matriz com valores de uma string

De forma similar à função anterior, criaremos outra função, agora para uma matriz. Neste caso, separaremos os elementos de cada coluna com a vírgula (`","`) e cada linha com um ponto-e-vírgula (`";"`). Neste caso, a lógica é muito parecida, mas primeiro deveremos dividir as linhas e, para cada linha, teremos que dividir as colunas. Apresentamos o código a seguir.

```
1  def preencherMatriz(valores, tipo):
2      matriz = [ ]
3      linhas = valores.split(';')
4      for lin in range(len(linhas)):
5          colunas = linhas[lin].split(',')
6          vetor = [ ]
7          for col in range(len(colunas)):
8              if tipo == 'int':
9                  valor = int(colunas[col].strip())
10             elif tipo == 'float':
11                 valor = float(colunas[col].strip())
12             else:
13                 valor = colunas[col].strip()
14             vetor.append(valor)
15         matriz.append(vetor)
16     return matriz
```

7.3.5 Imprimir um vetor na tela

Outra tarefa que pode ser comum ao manipular vetores é imprimir seu conteúdo na tela. Estabeleceremos como padrão de impressão o mesmo usado para a atribuição, com o uso de colchetes para delimitar e vírgula para separar os elementos, por exemplo: `"[ 1, 2, 3 ]"`. Veja o código da função a seguir.

```
1  def imprimeVetor(vetor):
2      print('[', end='')
3      if len(vetor) > 0:
4          print(f' {vetor[0]}', end='')
5          for i in range(1, len(vetor)):
6              print(f', {vetor[i]}', end='')
7      print(' ]', end='')
```

O resultado desta função é apenas a impressão dos valores contidos no vetor na tela, portanto, nenhum valor é retornado por ela, assim, não utilizamos a cláusula `return`. Outra questão importante é que, se o vetor estiver vazio, apenas `"[ ]"` deverá ser impresso na tela, isso é garantido pela decisão da linha 3. Caso o vetor não esteja vazio, o primeiro valor é impresso sem o uso do separador `","` (fora do laço **for**), mas do segundo em diante inserimos o separador (dentro do laço). Todos os **print**'s usados na função utilizam um segundo argumento, `end=''`, indicando que não será inserida uma quebra de linha (padrão para o **print**) após imprimir a mensagem definida pelo primeiro argumento.

7.3.6 Imprimir uma matriz na tela

De forma similar a impressão do vetor, implementaremos duas funções para imprimir o conteúdo de uma matriz na tela. Em uma primeira função, usaremos o mesmo padrão utilizado na atribuição, imprimindo tudo em uma linha, por exemplo: `"[ [ 1, 2, 3 ], [ 4, 5, 6 ] ]"`. Por isso, chamaremos esta função de `imprimeMatrizEmLinha`, e apresentamos o código a seguir.

```
1 def imprimeMatrizEmLinha(matriz):
2     print('[ ', end='')
3     if len(matriz) > 0:
4         imprimeVetor(matriz[0])
5         for lin in range(1, len(matriz)):
6             print(', ', end='')
7             imprimeVetor(matriz[lin])
8     print(' ]', end='')
```

Um ponto bem interessante aqui é que utilizamos a função de impressão de um vetor para imprimir a matriz, uma vez que cada linha da matriz pode ser considerada como um vetor, que segue o mesmo padrão de impressão de vetor. A lógica é a mesma, mas aqui ao invés de imprimir um único elemento por vez, imprimimos uma linha por vez, chamando a função `imprimeVetor`.

Em uma segunda função, faremos a impressão da matriz em formato de tabular, separando as linhas. Esta função se chamará simplesmente `imprimeMatriz`, e o código é apresentado a seguir.

```
1 def imprimeMatriz(matriz):
2     if len(matriz) > 0:
3         imprimeVetor(matriz[0])
4         print('')
5         for lin in range(1, len(matriz)):
6             imprimeVetor(matriz[lin])
7             print('')
```

A diferença para a função anterior é que não delimitamos toda a matriz por `"[ ]"` e a cada linha da matriz nós imprimimos uma quebra de linha na tela com a instrução `print('')`, que não imprime nada na tela, mas que por padrão quebra uma linha após a sua execução, devido ao funcionamento da função **print**.

7.3.7 Criando e usando a biblioteca

Criar a biblioteca consiste em salvar todas as definições de funções em um arquivo ".py", como por exemplo "biblioteca.py". E, para usa-la, você pode importar o arquivo de biblioteca em seu programa, um outro arquivo ".py", como por exemplo "teste.py". No entanto, esta forma simples de utilização da biblioteca exige que os dois arquivos estejam juntos, na mesma pasta do computador. Porém, isso já é suficiente para que possamos resolver os exercícios que serão propostos, por isso, não aprofundaremos este assunto.

No programa a seguir testamos nossas funções, salvas no arquivo "biblioteca.py". Nele, importamos o arquivo da biblioteca, com a instrução `import biblioteca as bib`, definindo um **apelido** para a biblioteca: **bib**. Isto significa que, quando formos usar uma função da biblioteca, podemos usar `"bib."` no lugar de `"biblioteca."`. Por exemplo, para criar o primeiro vetor, contendo 5 elementos com valor 3, usamos `bib.criarVetor(5, 3)` e em seguida, para imprimir seus valores na tela, usamos `bib.imprimeVetor(v)`. Para deixar a saída mais agradável de se ver, utilizamos **print** com mensagens e quebras de linhas estratégicas (`"\n"`). O resultado da execução (*saída*) deste programa você pode ver logo em seguida.

```
1  import biblioteca as bib
2
3  v = bib.criarVetor(5, 3)
4  print('Conteúdo do vetor:')
5  bib.imprimeVetor(v)
6
7  m = bib.criarMatriz(2, 3, 0.5)
8  print('\n\nConteúdo da matriz (em linha):')
9  bib.imprimeMatrizEmLinha(m)
10 print('\n\nConteúdo da matriz (tabular):')
11 bib.imprimeMatriz(m)
12
13 v = bib.preencherVetor('1.0, 1.2, 1.4', 'float')
14 print('\nConteúdo do vetor:')
15 bib.imprimeVetor(v)
16
17 m = bib.preencherMatriz('1.0, 1.2, 1.4 ; 2.0, 2.2, 2.4', 'float')
18 print('\n\nConteúdo da matriz (em linha):')
19 bib.imprimeMatrizEmLinha(m)
20 print('\n\nConteúdo da matriz (tabular):')
21 bib.imprimeMatriz(m)
```

Saída:

```
Conteúdo do vetor:
[ 3, 3, 3, 3, 3 ]

Conteúdo da matriz (em linha):
[ [ 0.5, 0.5, 0.5 ], [ 0.5, 0.5, 0.5 ] ]

Conteúdo da matriz (tabular):
[ 0.5, 0.5, 0.5 ]
[ 0.5, 0.5, 0.5 ]

Conteúdo do vetor:
[ 1.0, 1.2, 1.4 ]

Conteúdo da matriz (em linha):
[ [ 1.0, 1.2, 1.4 ], [ 2.0, 2.2, 2.4 ] ]

Conteúdo da matriz (tabular):
[ 1.0, 1.2, 1.4 ]
[ 2.0, 2.2, 2.4 ]
```

7.4 Exercícios propostos

Resolva os exercícios a seguir fazendo uso da biblioteca de funções úteis.

Exercício 8

Implemente um programa para contabilizar a pontuação em uma Gincana, seguindo o algoritmo:

1. Pergunte a quantidade de equipes.
2. Crie um vetor, cada elemento representa uma equipe e possui pontuação inicial $zero$.
3. Pergunte o índice da equipe que pontuou.
4. Enquanto o índice for válido:
   (a) Pergunte a quantidade de pontos.
   (b) Atualize, de forma cumulativa, a pontuação da equipe no vetor.
   (c) Pergunte o índice da equipe que pontuou.
5. Imprima o conteúdo do vetor.

Tanto os índices quanto a pontuação devem ser valores inteiros.
Exemplos de execução:

Exemplo 1:

```
Quantidade de equipes: 5
Índice da equipe: 2
Quantidade de pontos: 10
Índice da equipe: 1
Quantidade de pontos: 5
Índice da equipe: 0
Quantidade de pontos: 30
Índice da equipe: 2
Quantidade de pontos: 25
Índice da equipe: -1
Pontuação das equipes:
[ 30, 5, 35, 0, 0 ]
```

Exemplo 2:

```
Quantidade de equipes: 6
Índice da equipe: 5
Quantidade de pontos: 4
Índice da equipe: 1
Quantidade de pontos: 3
Índice da equipe: 1
Quantidade de pontos: 4
Índice da equipe: 6
Pontuação das equipes:
[ 0, 7, 0, 0, 0, 4 ]
```

Exercício 9

Implemente um programa para ajudar na leitura de consumo de energia feito pela CEMIG nos imóveis de uma cidade. O registro é feito pelo mapeamento de ruas e imóveis e armazenado em uma matriz, as linhas representam as ruas e as colunas representam os números dos imóveis. Primeiro o programa solicita que seja fornecida como entrada uma **string** que contenha os valores iniciais, preenchendo uma matriz com estes valores. Em seguida, o programa pergunta quantas novas leituras individuais serão fornecidas. Para cada leitura individual, o programa pergunta o índice da rua (linha), o índice do imóvel (coluna), e o valor lido pelo funcionário. O programa deve então fornecer a diferença da última leitura referente ao imóvel do valor lido pelo funcionário, e em seguida atualizar a matriz com o valor lido para o imóvel.

Tanto os índices quanto as leituras devem ser valores inteiros. Não é necessário validar dados de entrada, considere que serão sempre válidos.

Exemplo de execução:

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 6
 - Diferença: 2
Leitura 2:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 10
 - Diferença: 4
Leitura 3:
 - Índice da rua: 2
 - Índice do imóvel: 2
 - Valor lido: 5
 - Diferença: 3
Leitura 4:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 5
 - Diferença: 0
Leitura 5:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 7
 - Diferença: 2
```

Exercício 10

Implemente um programa similar ao anterior, Exercício 9. Mas, agora a entrada de cada leitura deve ser feita por um único **input**, armazenando os valores em um vetor, onde, índice 0 representa a rua, índice 1 representa o imóvel e índice 2 representa o valor lido. O restante do programa funciona da mesma maneira, gerando o mesmo resultado, conforme o exemplo de execução a seguir.

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1: 0, 2, 6
 - Diferença: 2
Leitura 2: 0, 2, 10
 - Diferença: 4
Leitura 3: 2, 2, 5
 - Diferença: 3
Leitura 4: 0, 0, 5
 - Diferença: 0
Leitura 5: 0, 0, 7
 - Diferença: 2
```

7.5 Solução dos Exercícios Propostos

Exercício 1

Utilizando vetores, imprimir uma sequência de pontos indicando a distância de queda livre de um corpo, dada uma sequência de tempo de $t = 0$ até um valor dado como entrada pelo usuário. Os valores impressos devem ter uma precisão de uma casa decimal.

Fórmulas:

- $g = 9.807m/s^2$
- $d = \frac{gt^2}{2}$

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
```

Exemplo 2:

```
Digite o tempo final: 6
t D(t)
0 0.0
1 4.9
2 19.6
3 44.1
4 78.5
5 122.6
6 176.5
```

Solução:

```
1  ft = int(input('Digite o tempo final: '))
2  g = 9.807
3  T = []
4  D = []
5  for i in range(ft+1):
6      T.append(i)
7      D.append(g*i*i/2)
8  print('t D(t)')
9  for i in range(ft+1):
10     print(f'{T[i]} {D[i]:.1f}')
```

Exercício 2

Uma função de segundo grau é dada pela fórmula:

$$f(x) = ax^2 + bx + c$$

Dadas como entradas:

- Valores a, b e c
- Ponto x_0 inicial
- Ponto x_f final

Criar dois vetores com os dados dos valores da função nesses pontos.
Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -3
Digite o valor final de x: 3
x f(x)
-3 13
-2 6
-1 3
0 4
1 9
2 18
3 31
```

Exemplo 2:

```
Digite o valor de "a": 2
Digite o valor de "b": 0
Digite o valor de "c": 0
Digite o valor inicial de x: -5
Digite o valor final de x: 2
x f(x)
-5 50
-4 32
-3 18
-2 8
-1 2
0 0
1 2
2 8
```

Solução:

```
a = int(input('Digite o valor de "a": '))
b = int(input('Digite o valor de "b": '))
c = int(input('Digite o valor de "c": '))
x0 = int(input('Digite o valor inicial de x: '))
x1 = int(input('Digite o valor final de x: '))
X = []
Y = []
for i in range(x0, x1+1):
    X.append(i)
    Y.append(a*i*i + b*i + c)
print('x f(x)')
for i in range(len(Y)):
    print(f'{X[i]} {Y[i]}')
```

Exercício 3

Implemente um programa que realize a entrada de dados de um vetor. Primeiro o usuário fornece a quantidade de elementos do vetor, em seguida, fornece um valor numérico para cada elemento. Após preenchido o vetor, os valores armazenados no mesmo devem ser impressos na tela com uma precisão de uma casa decimal.

Exemplos de execução:

Exemplo 1:

```
Informe a dimensão do vetor: 3
Valor do elemento 0: 3.5
Valor do elemento 1: 7
Valor do elemento 2: 10.65
Elementos do vetor:
3.5
7.0
10.7
```

Exemplo 2:

```
Informe a dimensão do vetor: 5
Valor do elemento 0: 6.92
Valor do elemento 1: 4
Valor do elemento 2: 5.271
Valor do elemento 3: 2.34
Valor do elemento 4: 2.319
Elementos do vetor:
6.9
4.0
5.3
2.3
2.3
```

Solução:

```
N = int(input('Informe a dimensão do vetor: '))
vetor = [ ]
for i in range(N):
    vetor.append(float(input(f'Valor do elemento {i}: ')))
print('Elementos do vetor: ')
for i in range(N):
    print(f'{vetor[i]:.1f}')
```

Exercício 4

Repetir a atividade do Exercício 1, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o tempo final: 3
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
```

Exemplo 2:

```
Digite o tempo final: 6
['t', 'D(t)']
[0, 0.0]
[1, 4.9035]
[2, 19.614]
[3, 44.1315]
[4, 78.456]
[5, 122.5875]
[6, 176.526]
```

Solução:

```
ft = int(input('Digite o tempo final: '))
g = 9.807
data = []
for i in range(ft+1):
    linha = []
    linha.append(i)
    linha.append(g*i*i/2)
    data.append(linha)
print('[\'t\', \'D(t)\']')
for i in range(ft+1):
    print(f'[{data[i][0]}, {data[i][1]}]')
```

Exercício 5

Repetir a atividade do Exercício 2, da Seção 7.1.3, utilizando uma matriz para armazenar os dados. Os dados devem estar separados por colunas.

Exemplos de execução:

Exemplo 1:

```
Digite o valor de "a": 2
Digite o valor de "b": 3
Digite o valor de "c": 4
Digite o valor inicial de x: -5
Digite o valor final de x: 5

['x', 'f(x)']
[-5, 39]
[-4, 24]
[-3, 13]
[-2, 6]
[-1, 3]
[0, 4]
[1, 9]
[2, 18]
[3, 31]
[4, 48]
[5, 69]
```

Exemplo 2:

```
Digite o valor de "a": 4
Digite o valor de "b": 0
Digite o valor de "c": -4
Digite o valor inicial de x: -3
Digite o valor final de x: 6

['x', 'f(x)']
[-3, 32]
[-2, 12]
[-1, 0]
[0, -4]
[1, 0]
[2, 12]
[3, 32]
[4, 60]
[5, 96]
[6, 140]
```

Solução:

```
#Entrada
a = int(input('Digite o valor de "a": '))
b = int(input('Digite o valor de "b": '))
c = int(input('Digite o valor de "c": '))
x0 = int(input('Digite o valor inicial de x: '))
x1 = int(input('Digite o valor final de x: '))
data = []
for i in range(x0, x1+1):
    linha = []
    linha.append(i)
    linha.append(a*i*i + b*i + c)
    data.append(linha)
print('[\'x\', \'f(x)\']')
for i in range(len(data)):
    print(f'[{data[i][0]}, {data[i][1]}]')
```

Exercício 6

Fazer um programa que gere uma matriz quadrada de dimensão n, preenchida em todas as posições por um valor v. Ambos n e v são valores informados pelo usuário, como entradas. Imprimir o conteúdo da matriz ao final do programa.

Exemplos de execução:

Exemplo 1:

```
Digite a dimensão da matriz: 4
Digite o valor para preencher os campos: 2
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
[2, 2, 2, 2]
```

Exemplo 2:

```
Digite a dimensão da matriz: 6
Digite o valor para preencher os campos: -3
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
[-3, -3, -3, -3, -3, -3]
```

Solução:

```
n = int(input('Digite a dimensão da matriz: '))
v = int(input('Digite o valor para preencher os campos: '))
matriz = []
for i in range(n):
    linha = []
    for j in range(n):
        linha.append(v)
    matriz.append(linha)
for i in range(n):
    print(f'[{matriz[i][0]}', end='')
    for j in range(1, n):
        print(f', {matriz[i][j]}', end='')
    print(']')
```

Exercício 7

Implemente um programa que realize a entrada de dados de uma matriz. Primeiro o usuário fornece as dimensões da matriz, em seguida, fornece um valor textual para cada elemento. Após preenchida a matriz, o usuário fornece uma linha e coluna, e o programa imprime o valor correspondente a estes índices. Considere que os dados informados são sempre válidos.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de linhas: 2
Informe a quantidade de colunas: 3
Valor do elemento 0, 0: Sardinha
Valor do elemento 0, 1: Salame
Valor do elemento 0, 2: Picanha
Valor do elemento 1, 0: Coca-cola
Valor do elemento 1, 1: Guaraná
Valor do elemento 1, 2: Pipoca
Informe uma linha: 1
Informe uma coluna: 2
O valor armazenado em [1,2] é Pipoca
```

Solução:

```
qtdLin = int(input('Informa a quantidade de linhas: '))
qtdCol = int(input('Informa a quantidade de colunas: '))
matriz = [ ]
for lin in range(qtdLin):
    linha = [ ]
    for col in range(qtdCol):
        valor = input(f'Valor do elemento {lin}, {col}: ')
        linha.append(valor)
    matriz.append(linha)
```

```
10 lin = int(input('Informe uma linha: '))
11 col = int(input('Informe uma coluna: '))
12 print(f'O valor armazenado em [{lin},{col}] é {matriz[lin][col]}')
```

Exercício 8

Implemente um programa para contabilizar a pontuação em uma Gincana, seguindo o algoritmo:

1. Pergunte a quantidade de equipes.
2. Crie um vetor, cada elemento representa uma equipe e possui pontuação inicial $zero$.
3. Pergunte o índice da equipe que pontuou.
4. Enquanto o índice for válido:
   (a) Pergunte a quantidade de pontos.
   (b) Atualize, de forma cumulativa, a pontuação da equipe no vetor.
   (c) Pergunte o índice da equipe que pontuou.
5. Imprima o conteúdo do vetor.

Tanto os índices quanto a pontuação devem ser valores inteiros.
Exemplos de execução:

Exemplo 1:

```
Quantidade de equipes: 5
Índice da equipe: 2
Quantidade de pontos: 10
Índice da equipe: 1
Quantidade de pontos: 5
Índice da equipe: 0
Quantidade de pontos: 30
Índice da equipe: 2
Quantidade de pontos: 25
Índice da equipe: -1
Pontuação das equipes:
[ 30, 5, 35, 0, 0 ]
```

Exemplo 2:

```
Quantidade de equipes: 6
Índice da equipe: 5
Quantidade de pontos: 4
Índice da equipe: 1
Quantidade de pontos: 3
Índice da equipe: 1
Quantidade de pontos: 4
Índice da equipe: 6
Pontuação das equipes:
[ 0, 7, 0, 0, 0, 4 ]
```

Solução:

```
1  import biblioteca as bib
2
3  N = int(input('Quantidade de equipes: '))
4  pontuacao = bib.criarVetor(N, 0)
5  equipe = int(input('Índice da equipe: '))
6  while equipe >= 0 and equipe < N:
7      pontos = int(input('Quantidade de pontos: '))
8      pontuacao[equipe] = pontuacao[equipe] + pontos
9      equipe = int(input('Índice da equipe: '))
10 print('Pontuação das equipes:')
11 bib.imprimeVetor(pontuacao)
```

Exercício 9

Implemente um programa para ajudar na leitura de consumo de energia feito pela CEMIG nos imóveis de uma cidade. O registro é feito pelo mapeamento de ruas e imóveis e armazenado em uma matriz, as linhas representam as ruas e as colunas representam os números dos imóveis. Primeiro o programa solicita que seja fornecida como entrada uma **string** que contenha os valores iniciais, preenchendo uma matriz com estes valores. Em seguida, o programa pergunta quantas novas leituras individuais serão fornecidas. Para cada leitura individual, o programa pergunta o índice da rua (linha), o índice do imóvel (coluna), e o valor lido pelo funcionário. O programa deve então fornecer a diferença da última leitura referente ao imóvel do valor lido pelo funcionário, e em seguida atualizar a matriz com o valor lido para o imóvel.

Tanto os índices quanto as leituras devem ser valores inteiros. Não é necessário validar dados de entrada, considere que serão sempre válidos.

Exemplo de execução:

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 6
 - Diferença: 2
Leitura 2:
 - Índice da rua: 0
 - Índice do imóvel: 2
 - Valor lido: 10
 - Diferença: 4
Leitura 3:
 - Índice da rua: 2
 - Índice do imóvel: 2
 - Valor lido: 5
 - Diferença: 3
```

```
Leitura 4:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 5
 - Diferença: 0
Leitura 5:
 - Índice da rua: 0
 - Índice do imóvel: 0
 - Valor lido: 7
 - Diferença: 2
```

Solução:

```
import biblioteca as bib

valores = input('Valores iniciais: ')
matriz = bib.preencherMatriz(valores, 'int')
N = int(input('Quantidade de novas leituras: '))
for i in range(N):
    print(f'Leitura {i+1}:')
    lin = int(input(' - Índice da rua: '))
    col = int(input(' - Índice do imóvel: '))
    valor = int(input(' - Valor lido: '))
    diferenca = valor - matriz[lin][col]
    matriz[lin][col] = valor
    print(f' - Diferença: {diferenca}')
```

Exercício 10

Implemente um programa similar ao anterior, Exercício 9. Mas, agora a entrada de cada leitura deve ser feita por um único **input**, armazenando os valores em um vetor, onde, índice 0 representa a rua, índice 1 representa o imóvel e índice 2 representa o valor lido. O restante do programa funciona da mesma maneira, gerando o mesmo resultado, conforme o exemplo de execução a seguir.

Exemplo 1:

```
Valores iniciais: 5, 10, 4 ; 6, 3, 8 ; 7, 8, 2
Quantidade de novas leituras: 5
Leitura 1: 0, 2, 6
 - Diferença: 2
Leitura 2: 0, 2, 10
 - Diferença: 4
Leitura 3: 2, 2, 5
 - Diferença: 3
Leitura 4: 0, 0, 5
 - Diferença: 0
Leitura 5: 0, 0, 7
 - Diferença: 2
```

Solução:

```
1  import biblioteca as bib
2
3  valores = input('Valores iniciais: ')
4  matriz = bib.preencherMatriz(valores, 'int')
5  N = int(input('Quantidade de novas leituras: '))
6  for i in range(N):
7      leitura = input(f'Leitura {i+1}: ')
8      vetor = bib.preencherVetor(leitura, 'int')
9      lin = vetor[0]
10     col = vetor[1]
11     diferenca = vetor[2] - matriz[lin][col]
12     matriz[lin][col] = vetor[2]
13     print(f' - Diferença: {diferenca}')
```