Disciplina: Introdução a Programação

Aula 10: Estruturas de dados homogêneas bidimensionais

Apresentação

Na aula anterior, vimos os vetores, que nos permitem armazenar valores de um mesmo tipo em uma única variável. Eles são estruturas unidimensionais que permitem o armazenamento linear de valores.

Isso significa que, se você precisasse escrever um algoritmo ou um programa capaz de calcular o valor do determinante de uma matriz, precisaria de um tipo de variável que lhe permitisse manipular mais de uma dimensão, já que matrizes são compostas por linhas e colunas.

Esse tipo de variável existe e é o tema desta aula.

Bons estudos!

Objetivos

- Definir as matrizes e sua funcionalidade;
- Avaliar o comportamento de algoritmos e programas com matrizes;
- Escrever códigos com matrizes como recurso de armazenamento.

Matrizes

Na aula anterior, estudamos os vetores e aprendemos que são uma variável capaz de armazenar um conjunto de dados do mesmo tipo, todos sob o mesmo identificador, em espaços contíguos da memória principal, que podem ser individualmente referenciados por meio de um índice. Vimos que os vetores são unidimensionais, ou seja, nos permitem trabalhar com somente uma dimensão na qual os dados são linearmente armazenados.

Mas, e se precisássemos trabalhar com mais de uma dimensão para dar conta, por exemplo, de construir um código para um jogo de damas?

Seria necessário tratar a posição das peças considerando a linha e a coluna em que se encontram, não é verdade?

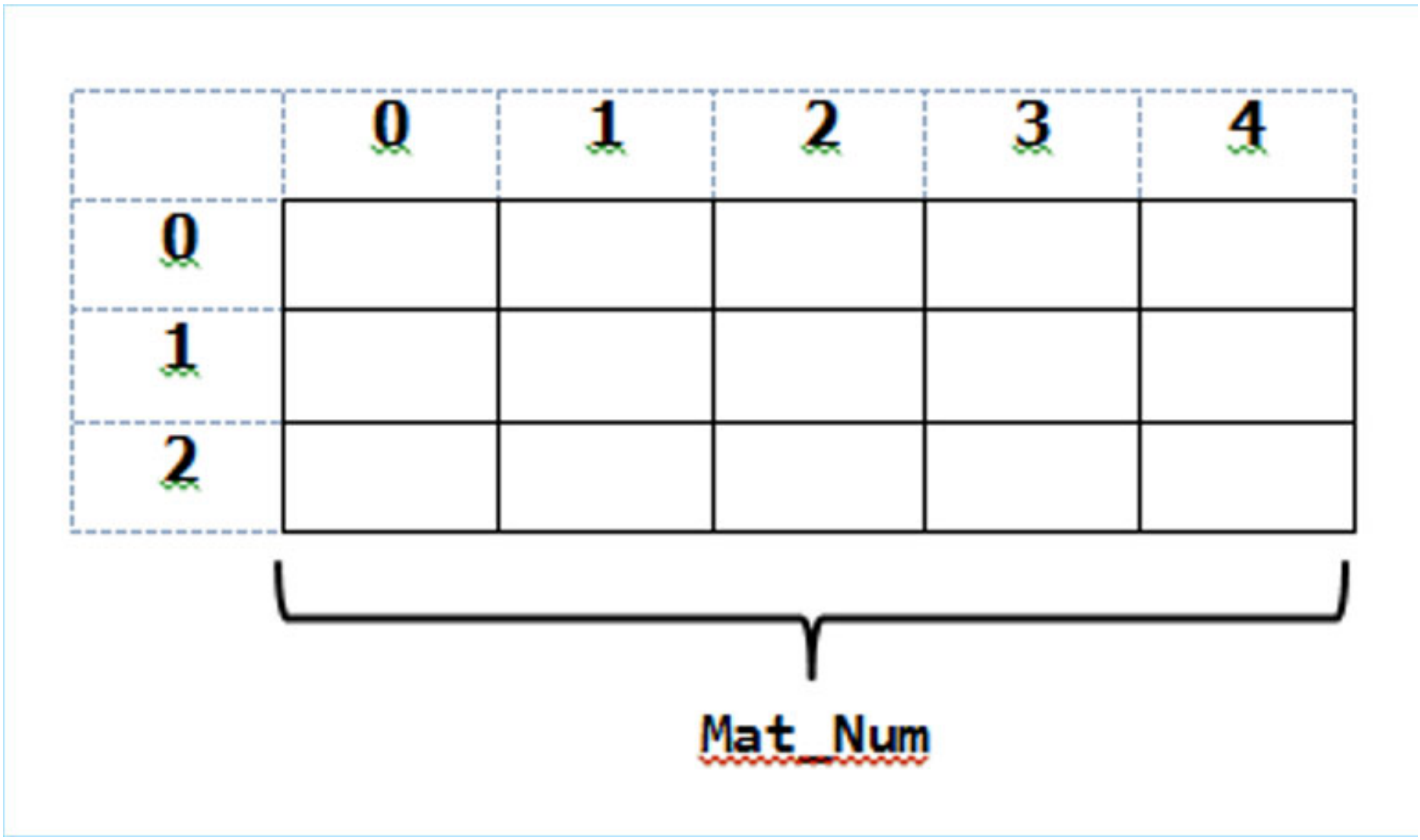
Nesse caso, precisaríamos de outro tipo de variável: as matrizes.

As matrizes são estruturas bidimensionais que podem ser consideradas vetores de vetores. Elas armazenam valores de um mesmo tipo e também precisam que a localização do elemento seja identificada para que o mesmo possa ser armazenado ou recuperado. Ao contrário dos vetores, que aparecem acompanhados de somente um índice, as matrizes precisam de dois índices: um para identificar a linha do elemento e outro para identificar a coluna do mesmo.

Vejamos um exemplo:

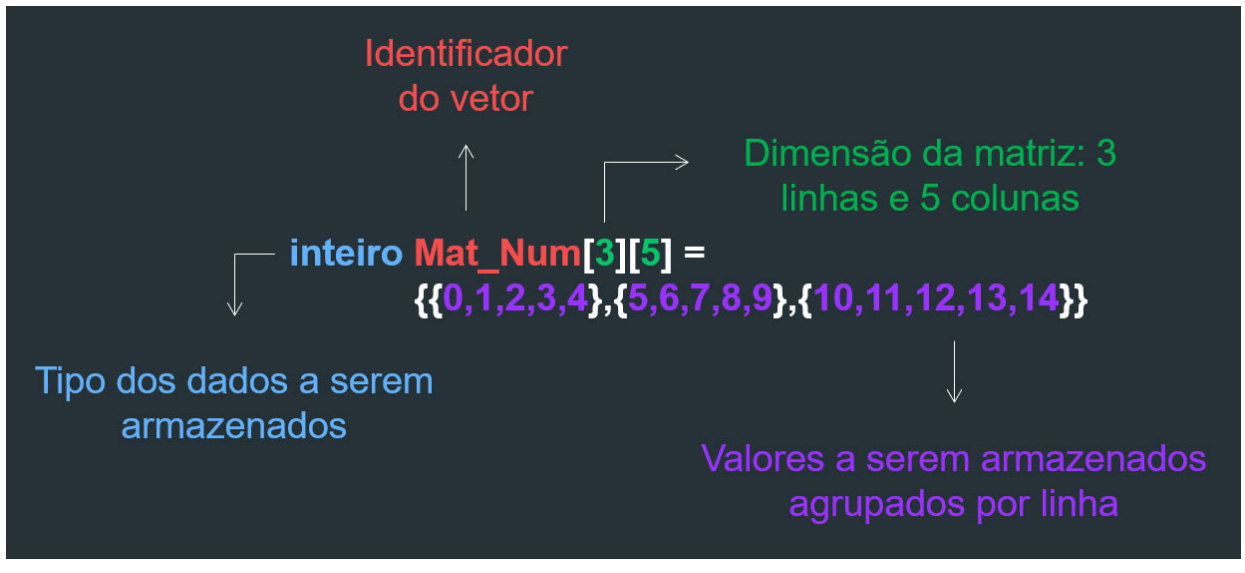
Exemplo

Imagine que você precisa criar uma matriz capaz de armazenar 15 elementos do tipo inteiro e deseja chamá-la de **Mat_Num**. Essa matriz deve ter tr linhas e cinco colunas. Se fossemos representá-la graficamente, ela teria a seguinte aparência:



Cada retângulo representa um elemento da matriz. Os números de 0 a 2 representam as linhas, e os números de 0 a 4 representam colunas. Assim como vimos nos vetores, o primeiro índice é 0. Assim, a posição 0,0 é a que identifica o elemento armazenado na primeira coluna da primeira linha. Cada um desses retângulos pode armazenar um valor do tipo inteiro, caso a matriz seja declarada como sendo desse tipo. É importante ressaltar que, assim como os vetores, as matrizes também são **estruturas de dados homogêneas**, o que significa que são capazes de armazenar um conjunto de valores de um mesmo tipo.

Para declarar uma matriz no Portugol Studio, é preciso informar seu tipo, seu identificador, sua dimensão e, **opcionalmente**, os valores que a mesma irá armazenar. Observe o exemplo, no qual é criada a matriz **Mat_Num**, capaz de armazenar quinze elementos do tipo inteiro, organizados em três linhas e cinco colunas:



Atenção

Tanto no Portugol Studio quanto no C++, a definição do número de linhas e do número de colunas da matriz deve ser estar entre colchetes **[ ]**; e o conjunto de elementos a serem armazenados na mesma deve estar definido entre chaves **{ }**.

A declaração de matrizes no C++ segue a mesma sintaxe do Portugol Studio. É preciso informar seu tipo, seu identificador, sua dimensão e, opcionalmente, os dados que nela serão armazenados. Observe o exemplo de criação da matriz **Mat_Num**:

```
int Mat_Num[3][5] = {{0,1,2,3,4},{5,6,7,8,9},{10,11,12,13,14}};
```

Atenção

Como sabemos, o conjunto de elementos a serem armazenados na matriz deve estar definido entre chaves **{ }**. Observe, entretanto, que há novas chaves dentro das chaves que delimitam todos os elementos da matriz. A função delas é agrupar os elementos por linha. Veja:

```
int Mat_Num[3][5] = {{0,1,2,3,4},{5,6,7,8,9},{10,11,12,13,14}};
```

Legenda:
Elementos da linha 0
Elementos da linha 1
Elementos da linha 2

Conforme dito anteriormente, não é obrigatório armazenar valores na matriz no momento de sua criação. Caso queira somente criar a variável, pare a definição antes do sinal de atribuição. Veja:

*** Portugol Studio**

```
caracter Mat_vogais[2][5] // Matriz vogais, com 2 linhas e 5 colunas

inteiro Mat_num[5][10] // Matriz Mat_num, com 5 linhas e 10 colunas

real notas[35][3] // Matriz notas, com 35 linhas e 3 colunas
```

*** C++**

```
char Mat_vogais[2][5]; // Matriz vogais, com 2 linhas e 5 colunas

int Mat_num[5][10]; // Matriz Mat_num, com 5 linhas e 10 colunas

float notas[35][3]; // Matriz notas, com 35 linhas e 3 colunas
```

É muito importante saber que, para acessar as diferentes posições da matriz para fins de armazenamento ou recuperação de dados, é imprescindível informar tanto a linha quanto a coluna do elemento. Veja os exemplos a seguir:

*** Portugol Studio**

```
//Armazena a letra “E” na segunda coluna da primeira linha da matriz Mat_vogais

Mat_vogais[0][1] = ‘E’

//Armazena o número 35 na quinta coluna da terceira linha da matriz Mat_num

Mat_num[2][6] = 35

//Armazena o número 8.5 na primeira coluna da vigésima linha da matriz notas

notas[19][0] = 8.5
```

*** C++**

```
//Exibe o elemento armazenado na 2ª linha, 4ª coluna, da matriz Mat_vogais

cout <<Mat_vogais[1][3];

//Guarda em RESULT a soma dos valores da matriz Mat_num armazenados na //1ª linha, 4ª coluna

RESULT = Mat_num[0][3]+num[1][9];

//Guarda o valor informado via teclado na 24ª linha, 2ª coluna, da matriz notas

cin >>notas[25][2];
```

Dica

Lembre-se de que tanto no Portugol Studio quanto no C++, a primeira linha e a primeira coluna da matriz são identificadas pelo índice **0** . Assim, uma matriz 2x3 (duas linhas e três colunas) terá as linhas variando de 0 a 1, e as colunas variando de 0 a 2.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Atividade

1 - Leia os enunciados a seguir e crie os algoritmos que solucionem os problemas propostos. Siga a sintaxe do Portugol Studio:

a) Receba uma matriz de elementos inteiros dispostos em 3 linhas e 4 colunas. Em seguida, exiba a quantidade de números múltiplos de 5 armazenados na matriz.

b) Receba duas matrizes de duas linhas e quatro colunas cada. Feito isso, gere uma terceira matriz cujos elementos serão o produto entre a primeira matriz e a segunda. Exiba os valores da matriz gerada.

c) Receba uma matriz quadrada 4x4 (quatro linhas e quatro colunas) de elementos do tipo real. Em seguida, pergunte ao usuário a linha da matriz que ele deseja visualizar. Com essa informação, exiba a linha desejada.

d) Receba uma matriz 4x4 de números inteiros. Gere e exiba uma segunda matriz na qual as linhas são as colunas da matriz 1, e as colunas são as linhas da matriz 1. Observe:

1	2	3	4	1	5	9	13
5	6	7	8	2	6	10	14
9	10	11	12	3	7	11	15
13	14	15	16	4	8	12	16
Matriz 1				Matriz 2			

e) Receba uma matriz 3x2 e exiba a quantidade de elementos ímpares armazenados nas linhas pares.

2 - Reveja os enunciados da atividade 1. Agora, escreva os programas em C++ que resolvam as questões propostas:

a) Receba uma matriz de elementos inteiros dispostos em 3 linhas e 4 colunas. Em seguida, exiba a quantidade de números múltiplos de 5 armazenados na matriz.

b) Receba duas matrizes de duas linhas e quatro colunas cada. Feito isso, gere uma terceira matriz cujos elementos serão o produto entre a primeira matriz e a segunda. Exiba os valores da matriz gerada.

c) Receba uma matriz quadrada 4x4 (quatro linhas e quatro colunas) de elementos do tipo real. Em seguida, pergunte ao usuário a linha da matriz que ele deseja visualizar. Com essa informação, exiba a linha desejada.

d) Receba uma matriz 4x4 de números inteiros. Gere e exiba uma segunda matriz na qual as linhas são as colunas da matriz 1, e as colunas são as linhas da matriz 1. Observe:

1	2	3	4	1	5	9	13
5	6	7	8	2	6	10	14
9	10	11	12	3	7	11	15
13	14	15	16	4	8	12	16

Matriz 1 Matriz 2

e) Receba uma matriz 3x2 e exiba a quantidade de elementos ímpares armazenados nas linhas pares.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Preenchimento e exibição da matriz

Imagine que você precisa receber e armazenar as notas de teste, prova e trabalho de 35 alunos de uma turma.

Talvez a matriz notas criada no exemplo anterior possa servir, não é verdade?

Afinal, ela foi definida como uma matriz de 35 linhas e 3 colunas, para elementos do tipo real. Vamos ver como podemos fazer para preencher e exibir os elementos dessa matriz. Observe o algoritmo escrito conforme as regras do Portugol Studio:

*** Portugol Studio**

```
1 programa
2 {
3  funcao inicio()
4  {
5    inteiro lin,col
6   real notas[35][3]
7   para (lin=0;lin<=34;lin++)
8   {
9    escreva("*** Notas do ",lin+1,"o. aluno ***\n")
10    para (col=0;col<=2;col++)
11   {
12    escreva("Informe a nota ",col+1,": ")
13    leia(notas[lin][col])
14   }
15  }
16  para (lin=0;lin<=34;lin++)
17  {
18   escreva("*** Notas do ",lin+1,"o. aluno *** \n")
19   para (col=0;col<=2;col++)
20   {
21    escreva("A nota ",col+1," é: ", notas[lin][col], "\n")
22    }
23   }
24  }
25 }
```

Vamos às explicações do que fazem as linhas do algoritmo:

Clique nos botões para ver as informações.

Linha 5

É responsável por criar as variáveis **lin** e **col**, que irão controlar a posição da matriz a ser acessada.

Linha 6

É criada a matriz **notas**, com capacidade para armazenar 105 elementos dispostos em 35 linhas e 3 colunas.

Linha 7

Inicia-se a estrutura repetitiva para preenchimento da matriz. Nesta linha, temos o controle das linhas, que variam de 0 a 34.

Linha 10

Inicia-se a estrutura repetitiva para controle das colunas da matriz. Como temos 3 colunas, a repetição varia de 0 a 2. Veja que precisamos de duas estruturas de repetição porque controlamos uma variável com duas dimensões.

Assim, cada laço controla uma dimensão da matriz.

Linha 13

São armazenadas na matriz as notas informadas pelo usuário via teclado. Veja que o nome do vetor está acompanhado das variáveis de linha e coluna, e não de um valor numérico fixo.

Laços iniciados nas linhas 16 e 19

São responsáveis pela exibição de todos os valores armazenados na matriz.

Agora, imagine que você deseja receber uma matriz quadrada com 3 linhas e 3 colunas para calcular a soma dos elementos armazenados em posições onde a linha e a coluna tenham o mesmo valor. Veja como ficaria o programa em C++ para resolver essa situação:

*** Programa em C++**

```
1 #include < iostream >
2 using namespace std;
3 int main()
4 {
5  int Mat_Num[3][3];
6  int lin, col, soma=0;
7  for (lin=0;lin<=2;lin++)
8  {
9   for (col=0;col<=2;col++)
10   {
11    cout <<"Qual o elemento ["<<lin+1<<","<<col+1<<"]? ";
12     cin <<Mat_Num[lin][col];
13    if (lin==col)
14    {
15     soma+=Mat_Num[lin][col];
16    }
17   }
18  }
19  cout <<"A soma da diagonal principal é "<<soma;
20 }
```

Agora que você sabe como funcionam as matrizes e já viu alguns exemplos de sua utilização nos algoritmos e nos programas em C++, vamos fazer alguns exercícios para praticar.

Esta é a nossa última aula. Parabéns por ter chegado até aqui. Esperamos que a disciplina tenha contribuído para o aprimoramento do seu raciocínio lógico e torcemos para que as atividades tenham despertado em você o gosto pela programação de computadores.

Atividade

3 - Escreva um algoritmo e um programa em C++ que receba duas matrizes quadradas 2x2 denominada M1 e M2. Em seguida, troque os elementos da diagonal principal da primeira matriz com os elementos da diagonal principal da segunda matriz. Por fim, exiba as diagonais das duas matrizes.

Observação: Os elementos da diagonal principal são aqueles armazenados nas posições onde o número da linha e da coluna são iguais.

Atenção! Aqui existe uma videoaula, acesso pelo conteúdo online

Notas

comércio dos outros [1]

Assim, o governante mantém seu grupo familiar ampliado e seu pessoal militar com os proventos de seu próprio comércio e da exploração do comércio dos outros. Sua posição especial com respeito à propriedade da terra é, frequentemente, resultado e não causa da dominação política pela qual pode explorar oportunidades econômicas disponíveis.

Entretanto, o governo patrimonial com base nos recursos pessoais do governante e na administração do grupo familiar não pode normalmente dar conta dos problemas que surgem quando vastos territórios extrapatrimoniais ficam sujeitos a esse governo.

deveres dos dependentes [2]

Embora, em determinados casos, possa ser difícil distinguir entre a obediência pessoal de um dependente e os deveres públicos de um súdito político, está claro que a expansão do governo patrimonial tende a retirar o dependente e o súdito político do controle direto do governante.

aventureiro [3]

Essa característica aventureira é responsável pela extraordinária “plasticidade” do português, fazendo com que os primeiros colonos se fizessem instrumentos passivos, sobretudo se aclimatando facilmente, aceitando o que lhes sugeria o ambiente, sem cuidar de impor normas fixas e indeléveis.

Esse dominar ajustando-se às condições de nova terra acaba por deixar suas marcas na história da sociedade brasileira, e isto fica representado, talvez de forma mais intensa, na constituição das cidades, as quais, segundo a bela expressão do autor, foram como que semeadas, e não construídas por “ladrilhadores” com seus planejamentos e critérios rígidos, como as cidades da América hispânica.

Sergio Buarque [4]

Esse é, em linhas gerais, o raciocínio seguido dor Sergio Buarque na construção do que se pode considerar um dos eixos de argumentação de *Raízes do Brasil*.

Desde a apresentação do legado ibérico, associado à cultura da personalidade, até a construção da ideia de cordialidade, passando pelo ruralismo e pela aventura como elemento orquestrador da colonização, Sergio Buarque está descrevendo o “tradicionalismo” peculiar à sociedade brasileira, noção que toma a base do conceito de “patrimonialismo” de Max Weber, como podemos ver nas descrições anteriores.

Referências

MANZANO, J. A. N. G., OLIVEIRA, J. F. **Algoritmos**: lógica para desenvolvimento de programação de computadores. 28.ed. São Paulo: Érica, 2016.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados com aplicações em Java**. 2.ed. São Paulo: Prentice Hall, 2005.

Próxima aula

Explore mais

Você se lembra do exercício na aula anterior, que foi recomendado para que aprendesse o conceito de funções? Que tal tentar um desafio mais complexo?

Para compreender melhor os temas estudados nesta aula, sugerimos um desafio sobre Funções com parâmetros <https://studio.code.org/s/course4/stage/14/puzzle/1 > . Veja se consegue resolver os problemas propostos enquanto aplica a

noção de **funções**.