**FACULDADE DE COMPUTAÇÃO E INFORMÁTICA**

**CIÊNCIA DA COMPUTAÇÃO E SISTEMAS DE INFORMAÇÃO**

**Linguagem de Programação I**

**AULA 08: MATRIZES**

|  |  |
| --- | --- |
| http://images.flatworldknowledge.com/ketchen/ketchen-fig05_x001.jpg | Nossos **objetivos** nesta semana são:   * Conhecer o conceito de listas encadeadas e como defini-las em Python. * Utilizar listas encadeadas para representar matrizes. * Resolver problemas com listas encadeadas. |
|  | O **Capítulo 4** do livro:  DIERBACH, C. *Introduction to Computer Science Using Python: A Computational Problem Solving Focus.* 1st Edition, New York: Wiley, 2012. |
|  |  |

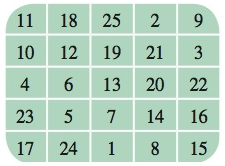
**MATRIZES**

* Até o momento, conhecemos o conceito de lista. Essencialmente, uma lista armazena informações com indexação em uma única dimensão (índice). Assim, uma lista também é conhecida pelo nome de ***array* unidimensional**.

Por exemplo:

|  |  |
| --- | --- |
| **data = [None] \* 10** | **data[2] = 29.95** |

* Podemos pensar numa extensão para duas dimensões. Esta estrutura bidimensional é chamada ***array* bidimensional** ou, comumente, **matriz**. Arrays bidimensionais formam um arranjo bidimensional tabular. Você acessa os elementos com um par de índices A[i][j]. Abaixo temos um exemplo de matriz:



* Em Python, pode-se implementar este arranjo com listas encadeadas (ou seja, listas dentro de outra lista).

Exemplo:

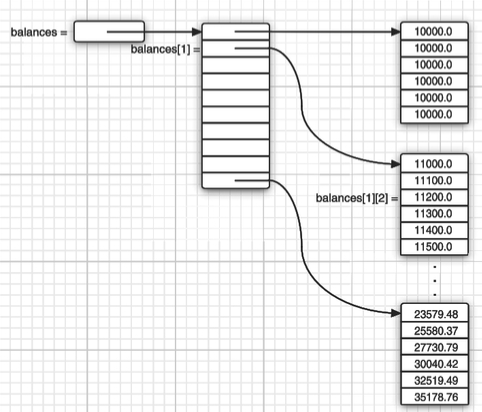
**A = [ [11,18,25,2,9], [10,12,19,21,3], [4,6,13,20,22], [23,5,7,14,16], [17,24,1,8,15] ]**

* O acesso aos elementos de uma matriz é feito através de dois índices: um para a linha e, outro, para a coluna. Por exemplo, considerando-se a matriz acessada por A, o acesso

**A[0][1]**

recuperará o valor 18 (linha 0, coluna 1).

* De forma similar, podemos ter arrays de qualquer dimensão: A[ ][ ][ ], A[ ][ ][ ][ ],... . Para cada dimensão, utilizamos um índice de acesso.
* Na verdade, uma array bidimensional em Python é uma lista de listas.



**EXERCÍCIO COM DISCUSSÃO EM DUPLAS**

Construa uma função que receba a quantidade de **linhas (lin)** e de **colunas (col)** e devolva uma matriz de dimensão **lin x col** contendo números inteiros aleatórios entre 0 e 100. Teste sua função.

def **generate\_matrix** (lin, col):

**EXERCÍCIO COM DISCUSSÃO EM DUPLAS**

Construa um procedimento que receba uma matriz de números inteiros e exiba o seu conteúdo na formatação gráfica de uma matriz. Teste seu procedimento.

def **print\_matrix**(m):

**EXERCÍCIO COM DISCUSSÃO EM DUPLAS**

Construa uma função que receba uma matriz de números inteiros e devolva o **maior valor** presente nesta matriz. Teste sua função.

def **max**(m):

**EXERCÍCIO 1**

Em um jogo de **batalha naval**, os jogadores posicionam suas embarcações em um grid de 10 linhas por 10 colunas.

Na nossa versão do jogo, as seguintes embarcações devem ser distribuídas no grid:

• 1 porta-aviões (ocupa 5 células lineares do grid)

• 1 encouraçado (ocupa 4 células lineares do grid)

• 1 cruzador (ocupa 3 células lineares do grid)

• 1 submarino (ocupa 2 células lineares do grid)

Cada embarcação pode ser posicionada na direção vertical ou na horizontal em relação ao grid. As embarcações não podem se sobrepor, e elas devem caber integralmente dentro do grid.

Para desenvolver um jogo de batalha naval, vamos utilizar uma matriz de caracteres de dimensão 10x10. Cada posição da matriz será preenchida usando a seguinte notação:

* O caractere ponto (.) indica que a célula não é ocupada por nenhuma embarcação.
* O caractere P indica que a célula está sendo ocupada por uma célula do porta-aviões.
* O caractere E indica que a célula está sendo ocupada por uma célula do encouraçado.
* O caractere C indica que a célula está sendo ocupada por uma célula do cruzador.
* O caractere S indica que a célula está sendo ocupada por uma célula do submarino.

A tabela abaixo apresenta um exemplo de preenchimento válido do grid com as embarcações:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | . | . | . | . | . | . | . | . | . | . |
| 1 | . | P | P | P | P | P | . | . | . | . |
| 2 | . | . | . | . | . | . | . | S | . | . |
| 3 | . | . | . | E | . | . | . | S | . | . |
| 4 | . | . | . | E | . | . | . | . | . | . |
| 5 | . | . | . | E | . | . | . | . | . | . |
| 6 | . | . | . | E | . | . | C | C | C | . |
| 7 | . | . | . | . | . | . | . | . | . | . |
| 8 | . | . | . | . | . | . | . | . | . | . |
| 9 | . | . | . | . | . | . | . | . | . | . |

a) Desenvolva uma função que retorne uma matriz 10x10 preenchida em todas as suas posições com o caractere ponto.

def **inicializarGrid():**

b) Desenvolva uma função para posicionar um porta-aviões no grid de batalha.

A função receberá como parâmetros:

* a referência para a matriz do **grid**;
* os índices da **linha** e da **coluna** da primeira posição a ser preenchida;
* a informação indicando se o porta-aviões deverá ser posicionado na **vertical** (o valor *true* indica orientação vertical e *false* indica orientação horizontal).

Se a orientação for horizontal, as demais partes do porta-aviões deverão ocupar as células à direita da célula identificada pelos parâmetros linha e coluna. Se a orientação for vertical, deverão ocupar as células abaixo da célula identificada.

A função retornará um valor booleano indicando se foi ou não possível posicionar o porta-aviões no grid.

O porta-aviões só poderá ser posicionado se:

* Nenhuma de suas partes for ficar fora do grid.
* Não houver nenhuma outra embarcação nas células que o porta-aviões iria ocupar.

def **posicionar\_porta\_avioes**(grid, linha, coluna, vertical):

c) Desenvolva uma função para imprimir o conteúdo do grid na tela do computador.

def **imprimir**(grid):

d) Escreva o **programa principal do módulo** de forma a:

* Inicializar o grid da batalha.
* Posicionar um porta-aviões em uma posição válida.
* Tentar posicionar um porta-aviões em uma posição que faria ele ficar fora do grid.
* Tentar posicionar um porta-aviões em uma posição que ocasionaria sobreposição com o primeiro porta-aviões.
* Imprimir o conteúdo do grid.

Execute e verifique se o resultado apresentado corresponde ao esperado.

e) Utilizando como modelo a função **posicionar\_porta\_avioes**, desenvolva as funções para posicionar encouraçado, cruzador e submarino.

f) No **programa principal**, acrescente chamadas às funções criados no item (e) e verifique se estão funcionando conforme o esperado.

**EXERCÍCIO 2**

O programa desenvolvido no exercício anterior só permite inicializar o grid e posicionar as embarcações. Incremente o programa de forma a gerar um jogo de batalha naval.

a) Desenvolva a função atirar. A função receberá como parâmetros:

* a referência para a matriz do **grid**;
* os índices da **linha** e da **coluna** onde o tiro está sendo realizado.

A função retornará um caractere com o valor que estava armazenado naquela posição do grid.

A execução da função deverá alterar o conteúdo do grid:

* Se a posição não tinha nenhuma embarcação (“água”), o novo valor deverá ser ‘x’ (minúsculo).
* Se a posição estava ocupada por uma embarcação, o novo valor deverá ser ‘X’ (maiúsculo).

def **atirar**(grid, linha, coluna):

b) Implemente o fluxo principal do jogo no **programa principal** de forma a:

* Inicializar o grid da batalha.
* Posicionar de forma aleatória, em posições válidas, um porta-aviões, um encouraçado, um cruzador e um submarino.
* Dar ao jogador 20 chances para atirar. A cada tiro o programa deverá informar ao jogador o resultado do tiro (água ou parte de uma das embarcações). Caso o jogador tente atirar pela segunda vez em uma mesma posição, o programa deverá informá-lo disto e dar uma nova chance sem descontar esta tentativa do total de tiros.
* Imprimir o conteúdo do grid ao final do jogo.

**EXERCÍCIOS EXTRA-CLASSE**

1. Dizemos que uma matriz quadrada A é **simétrica** se e somente se A[i][j]=A[j][i]. Implemente uma função para verificar se uma matriz de números inteiros é simétrica. Teste a sua função.
2. Dizemos que uma matriz quadrada A é **anti-simétrica** se e somente se A[i][j]=**−**A[j][i]. Implemente uma função para verificar se uma matriz de números inteiros é anti-simétrica. Teste a sua função.
3. A **matriz identidade (I)** é uma matriz quadrada onde os elementos da diagonal principal são todos iguais a 1 e, os demais elementos, todos iguais a 0. Construa uma função que receba um número inteiro n e devolva uma matriz identidade de dimensão nxn. Teste a sua função.
4. Implemente uma função que verifique se duas matrizes de números inteiros são **iguais**. Teste a sua função.
5. Uma matriz B é chamada de **inversa** de uma matriz A se e somente se AxB=I, ou seja, se o produto entre elas for a matriz identidade. Implemente uma função que receba duas matrizes A e B e verifique se B é a matriz inversa de A. Teste a sua função.
6. O **traço de uma matriz** é a soma dos elementos de sua diagonal principal. Implemente uma função que receba uma matriz quadrada (número de linhas = número de colunas) e devolva o seu **traço**. Teste a sua função.
7. Construa uma função que receba duas matrizes - A, de dimensão nxm e, b, de dimensão mxp – e devolva uma matriz de dimensão nxp, resultante do **produto de A por B.** Teste a sua função.
8. A **transposta** de uma matriz A, de dimensão n por m, é a matriz AT, de dimensão m por n, onde o elemento AT[j][i] da transposta corresponde ao elemento A[i][j] da matriz original. Implemente uma função que receba uma matriz de números inteiros A e devolva a sua transposta.
9. Dizemos que uma matriz A é **hermitiana** se e somente se A=AT. Implemente uma função que receba uma matriz de números inteiros A e devolva **true**, caso ela seja hermitiana e **false**, caso contrário. Teste a sua função.
10. O **Sr. Krika** deseja fazer vários orçamentos de custo de uma compra de materiais para construção. Os três produtos P1, P2 e P3 são necessários nas quantidades Q1, Q2 e Q3, respectivamente. Ao pesquisar os preços em 5 estabelecimentos comerciais, o Sr. Krika encontrou a seguinte tabela de preço:



Escreva um programa que **calcula o custo da compra dos três produtos em cada um dos estabelecimentos**. Considere que as informações a respeito dos estabelecimentos são fornecidas pelo usuário através do console.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Q1 | Q2 | Q3 |
| Quantidades | 10 | 8 | 17 |

Com essas informações o Sr. Krika multiplica as quantidades dos produtos pelos seus respectivos preços unitários em cada um dos estabelecimentos e realiza o somatório por estabelecimento para saber em qual estabelecimento ele pode comprar todos os produtos gastando menos, um trabalho extremamente cansativo para o Sr Krika. Dado essa complicação, ele resolveu contratar você para desenvolver um programa que o ajudasse determinar em qual estabelecimento é mais barato comprar todos os produtos, para tanto considere:

1. Para representar os **Estabelecimentos** utilize uma matriz com 3 linhas e N colunas, sendo assim, escreva uma função que aloca e lê a cotação dos preços dos produtos para os N estabelecimentos, retornando o estabelecimento com seus produtos.

def **leEstabelecimentos**(){...}

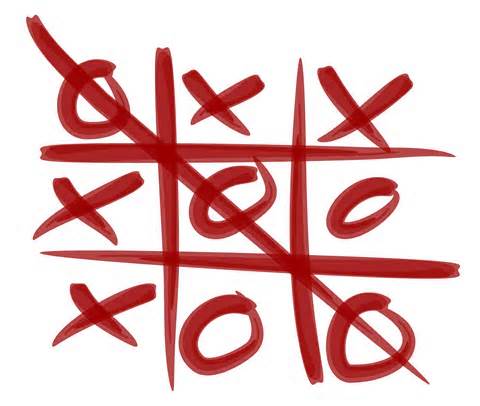
1. As quantidades dos produtos deverão ser armazenadas em um vetor de inteiros, implemente uma função que aloca e lê as quantidades para o vetor.

def **leQuantidades**(){...}

1. Implemente uma função que receba a matriz de estabelecimentos e o vetor de quantidades. A sua função calcula e devolve um vetor de **N** posições onde cada posição tem-se o preço total para se comprar todos os produtos no estabelecimento representado pela aquela posição.

def **somatorioEstabelecimentos**(estabelecimentos,quantidades){...}

1. Por fim, utilize as funções implementadas para implementar um programa principal, a função **Main()**, para ajudar o Sr. Krika a decidir onde é mais barato comprar todos os produtos.
2. O **Jogo da Velha** é um jogo bem conhecido e pode ser implementado com base na teoria que aprendemos nesta semana. No Jogo da Velha, temos dois jogadores, **O** e **X**, que jogam sob um tabuleiro 3x3. Ganha o jogo quem preencher primeiro uma linha, uma coluna ou uma das diagonais.



Seu objetivo neste exercício será implementar este jogo no modo texto. Para tanto:

a) Vamos representar o nosso tabuleiro como uma matriz de caracteres 3x3. Sua primeira função deverá devolver uma matriz de caracteres 3x3, com algum valor preenchido para indicar que uma posição não foi ocupada.

def **initialize**():

...

b) Implemente uma função que receba uma matriz do jogo da velha, uma posição (linha, coluna) e o código do jogador (X ou O) e devolva a matriz com a posição preenchida com o código do jogador, caso esteja livre. Se a posição conseguiu ser preenchida, **step** deve retornar **true** e, caso não puder ser preenchida, **false**.

def **step** (M, lin, col, gamer):

...

c) Implemente uma função que receba uma matriz do jogo da velha e verifique o estado do jogo: alguém venceu, ocorreu um empate ou o jogo deve continuar. Vamos usar o seguinte código: **-1** (o jogo pode continuar), **0** (ocorreu um empate), **1** (O venceu) e **2** (X venceu).

def **status** (M):

...

d) Implemente um procedimento para executar a lógica deste jogo. Suponha que o jogador **O** sempre começa. A cada jogada, a matriz do jogo deverá ser exibida na tela. Ao final do jogo, seu procedimento deve mostrar o estado a que se chegou (vitória ou empate). Teste o seu jogo.

def **game**():

...