Processamento da Informação
– Teoria –

Listas e Matrizes

Semana 07
Prof. Jesús P. Mena-Chalco

05/06/2013

Função range (cria uma lista)

```
>>> range(1,5)
[1, 2, 3, 4]

>>> range(5,10)
[5, 6, 7, 8, 9]
```

Uma lista que não contém nenhum elemento é chamada de lista vazia.

```
>>> range(10,10)
[]
```

Concatenação de listas

```
>>> a = [ 1 , 3 , 5]
>>> b = [ 2 , 4 , 6]

>>> c = a+b

>>> print c
[1, 3, 5, 2, 4, 6]
```

Método **append**

O Python fornece também métodos que operam sobre listas.

Por exemplo, adicionar um novo elemento no final de uma lista

```
>>> t = ['a', 'b', 'c']
>>> t.append('d')
>>> print t

['a', 'b', 'c', 'd']
```

Método **del**

Existem várias formas de eliminar elementos de uma lista. Usaremos apenas a função **'del'** para essa tarefa.

```
>>> t = ['a', 'b', 'c']
>>> del(t[1])
>>> print t
['a', 'c']
```

Exercício 00

Dadas uma lista numérica **A**, crie uma função que permita imprimir todos seus elementos.

```
def imprimir_lista(A):
    for i in range(0,len(A)):
        print A[i]
```

Exercício 00

Dadas uma lista numérica **A**, crie uma função que permita imprimir todos seus elementos.

```
def imprimir_lista(A):
    for i in range(0,len(A)):
        print A[i]
```

```
>>> imprimir_lista([5,6,7])
5
6
7
```

Exercício 01

Dadas uma lista numérica, **A** e um escalar **x**, crie uma função que permita determinar o produto **Y = x*A**.

A = [1,3,5,7]
X = 3
x*A = [3, 9, 15, 21]

Cabeçalho: def multiplica(A,x):

Exercício 01

```
def multiplica(A,x):
    B = [ ]
    for i in range(0,len(A)):
        B.append(A[i]*x)
    return B
```

Exercício 01

```
def multiplica(A,x):
    B = [ ]
    for i in range(0,len(A)):
        B.append(A[i]*x)
    return B
```

```
>>> multiplica([1,3,5,7], 3)
[3, 9, 15, 21]
```

Exercício 02

Dadas duas listas numéricas, **A** e **B**, crie uma função que permita determinar o produto interno dessas listas.

A = [2,1,7]
B = [4,5,2]
A.B = 27 = 8+5+14

Cabeçalho: def produto_interno(A,B):

Exercício 02

```
def produto_interno(A,B):
    soma = 0
    if len(A)==len(B):
        for i in range(0,len(A)):
            soma = soma + A[i]*B[i]
        return soma
    print 'Listas de comprimento diferente'
```

Exercício 02

```
def produto_interno(A,B):
    soma = 0
    if len(A)==len(B):
        for i in range(0,len(A)):
            soma = soma + A[i]*B[i]
        return soma
    print 'Listas de comprimento diferente'
```

```
>>> produto_interno([2,1,7], [4,5,2])
27
```

Listas aninhadas

[[12], ['a', 'e', 'i', 'o', 'u'], [3.1, 2.7]]

Lista de listas

Listas aninhadas

```
>>> M = [ [1, 0, 0], [0, 1, 0], [0, 0, 1] ]

>>> print M
[[1, 0, 0], [0, 1, 0], [0, 0, 1]]
```

A representação seria a seguinte:

```
[ [1, 0, 0],
  [0, 1, 0],
  [0, 0, 1] ]
```

← Lista de linhas

Matrizes

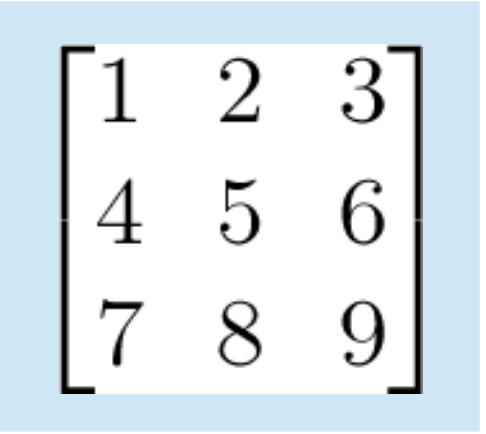
As listas aninhada são comumente utilizadas para **representar matrizes.**

A matriz $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

pode ser representada por:

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Matrizes

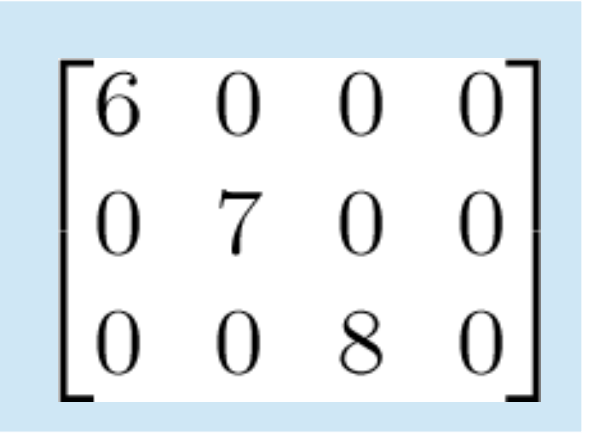


```
>>> M = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]

>>> M[1]                ← Seleção de uma linha
[4, 5, 6]

>>> M[1][1]             ← Seleção de um elemento
5
```

Matrizes

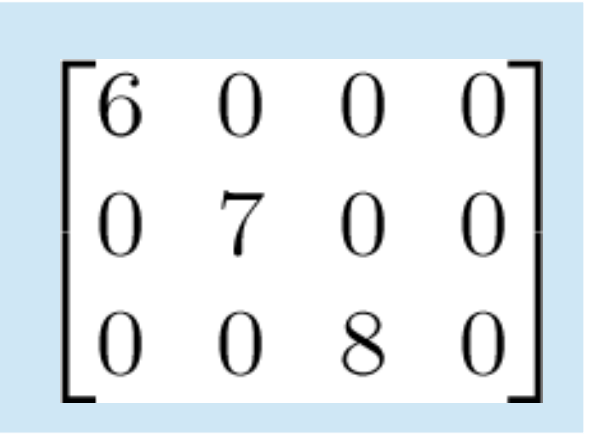



```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]

>>> P[2]                    ← Seleção de uma linha
[0, 0, 8, 0]

>>> P[2][2]                 ← Seleção de um elemento
8
```

Matrizes




```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]

>>> len(P)
3

>>> len(P[0])
4
```

← Número de linhas

← Número de colunas

Exercício 00

Dada uma matriz **A**, crie uma função que permita verificar se a matriz é quadrada.

Cabeçalho: **def matriz_quadrada(A):**

Exercício 00

```
def matriz_quadrada(A):
    if len(A)==len(A[0]):
        return true
    else:
        return false
```

Exercício 00

```
def matriz_quadrada(A):
    if len(A)==len(A[0]):
        return true
    else:
        return false
```

```
>>> M = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
>>> matriz_quadrada(M)
1
```

Exercício 00

```
def matriz_quadrada(A):
    if len(A)==len(A[0]):
        return true
    else:
        return false
```

```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]
>>> matriz_quadrada(P)
0
```

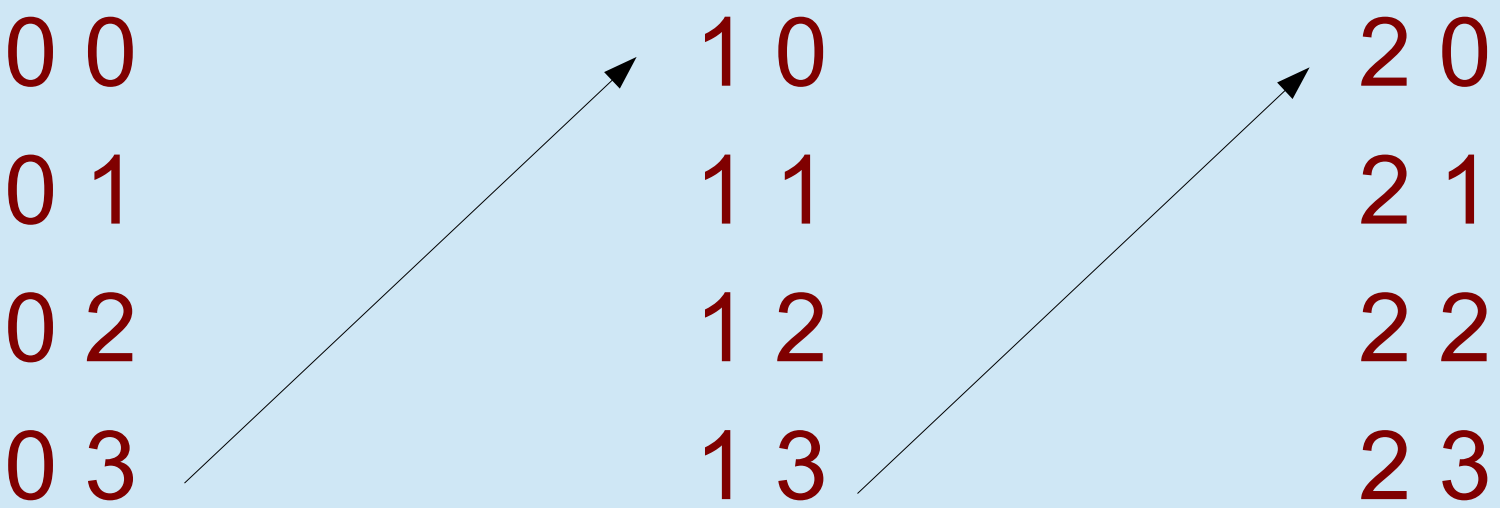
O que faz o seguinte algoritmo?

```
for i in range(0,3):
        for j in range(0,4):
            print i, j
```

O que faz o seguinte algoritmo?

```
for i in range(0,3):
        for j in range(0,4):
            print i, j
```

Imprime a seguinte sequência:



Exercício 01

Dada uma matriz quadrada **A**, crie uma função que permita **contar o número de zeros** contidos na matriz.

Cabeçalho: **def conta_zeros(A):**

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 8 & 0 \end{bmatrix}$$

Número de zeros = 9

Exercício 01

```
def conta_zeros(A):
    contador = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if A[i][j]==0:
                contador = contador+1
    return contador
```

Exercício 01

```
def conta_zeros(A):
    contador = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if A[i][j]==0:
                contador = contador+1
    return contador
```

```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]
>>> conta_zeros(P)
9
```

Exercício 02

Dada uma matriz quadrada **A**, crie uma função que permita **contar o número de elementos não nulos** contidos na matriz.

Cabeçalho: **def conta_nao_nulos(A):**

$$\begin{bmatrix} 6 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 8 & 0 \end{bmatrix}$$

Número de não nulos = 3

Exercício 02

```
def conta_nao_nulos(A):
    contador = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if A[i][j]!=0:
                contador = contador+1
    return contador
```

Exercício 02

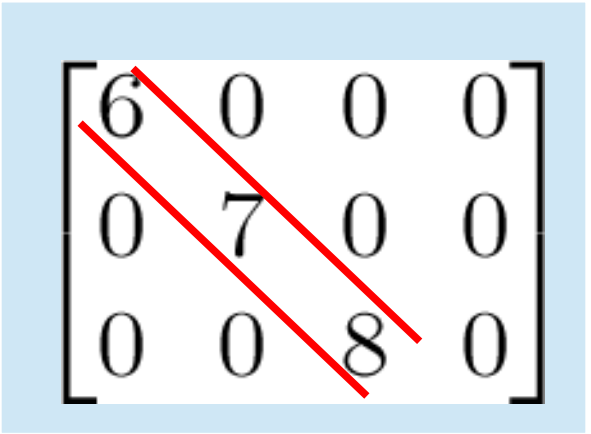
```
def conta_nao_nulos(A):
    contador = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if A[i][j]!=0:
                contador = contador+1
    return contador
```

```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]
>>> conta_nao_nulos(P)
3
```

Exercício 03

Dada uma matriz **A**, crie uma função que determine a **somatória de todos os números presentes na diagonal principal** da matriz.

Cabeçalho: **def soma_diagonal(A):**



Somatória da diagonal = 21

Exercício 03

```
def soma_diagonal(A):
    soma = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j:
                soma = soma + A[i][j]
    return soma
```

Exercício 03

```
def soma_diagonal(A):
    soma = 0
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j:
                soma = soma + A[i][j]
    return soma
```

```
>>> P = [ [6, 0, 0, 0], [0, 7, 0, 0], [0, 0, 8, 0] ]
>>> soma_diagonal(P)
21
```

Exercício 04

Dada uma matriz quadrada **A**, crie uma função que permita verificar se a matriz é **identidade**.

Cabeçalho: **def matriz_identidade(A):**

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Exercício 04

```
def matriz_identidade(A):
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j and A[i][j]!=1:
                return False
            if i!=j and A[i][j]!=0:
                return False
    return True
```

Exercício 04

```
def matriz_identidade(A):
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j and A[i][j]!=1:
                return False
            if i!=j and A[i][j]!=0:
                return False
    return True
```

```
>>> matriz_identidade([[1,0], [0,1]])
1
```

Exercício 04

```
def matriz_identidade(A):
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j and A[i][j]!=1:
                return False
            if i!=j and A[i][j]!=0:
                return False
    return True
```

```
>>> matriz_identidade([[1,0,0], [0,1,0], [0,0,1]])
1
```

Exercício 04

```
def matriz_identidade(A):
    for i in range(0,len(A)):
        for j in range(0,len(A[0])):
            if i==j and A[i][j]!=1:
                return False
            if i!=j and A[i][j]!=0:
                return False
    return True
```

```
>>> matriz_identidade([[1,0,0], [0,1,0], [0,0,0]])
0
```

Exercício 05 (casa)

Dada uma matriz quadrada **A**, crie uma função que permita verificar se a matriz é **simétrica**.

Cabeçalho: **def matriz_simétrica(A):**