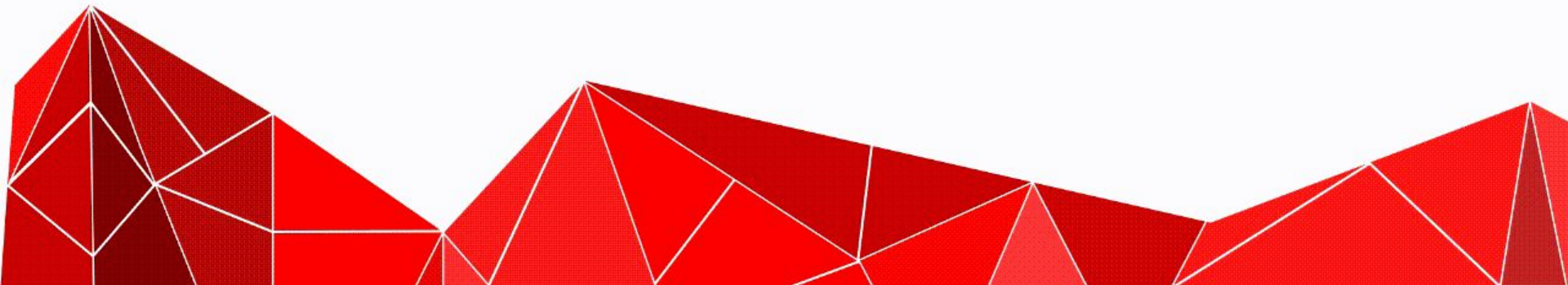




# **CODING MATRIZES REVOLUTION**

**Press -> to continue**

**Powered by PET computação UFC**

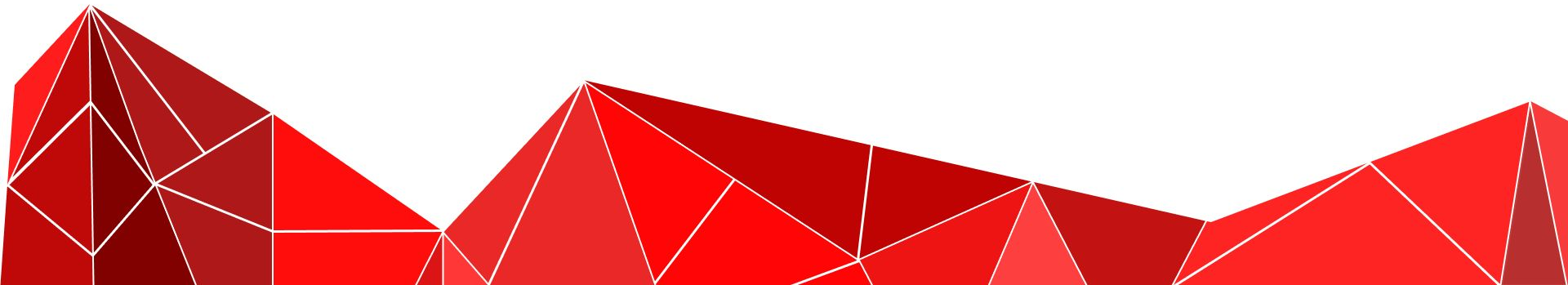


O'Que vamos ver hoje ?

Trabalhando com arrays

list comprehension

Matrizes





# Trabalhando com arrays



# Um looping pela array

- Quando você quiser fazer a mesma coisa com vários elementos
- Suponha que vamos printar uma lista de clientes

- Código

```
clientes = ['rafael', 'gates', 'wesley safadão']
```

```
for cliente in clientes:
```

```
    print(cliente)
```

saída :

rafael

gates

wesley safadão



# Um looping pela array

- Você pode fazer o que quiser com um looping
- É possível colocar quantas linhas de código quiser
- Podemos por exemplo, concatenar strings
- exemplo:

```
for cliente in clientes:  
    print(cliente.title() + ", obrigado pela preferência.")
```

Saída:  
Rafael, obrigado pela preferência.  
Gates, obrigado pela preferência.  
Wesley Safadão, obrigado pela preferência.



# Cuidado com a indentação

- Python usa indentação para saber quando um linha pertence ao bloco de cima
- Não esqueça de indentar o código após o for
- Exemplo:

```
for cliente in clientes:  
    print(cliente)
```



Saída:

```
    print(cliente)  
    ^
```

IndentationError: expected an indented block



# Cuidado com a indentação

- Não se esqueça de indentar todas as linhas

```
for cliente in clientes:
```

```
    print(cliente.title() + ", obrigado pela preferência.")  
    print("tenha um ótimo dia.")
```

saída :

Rafael, obrigado pela preferência.

Gates, obrigado pela preferência.

Wesley Safadão, obrigado pela preferência.

Tenha um ótimo dia.

- Apenas wesley recebeu um ótimo dia



# Cuidado com a indentação

- Cuidado também com indentações desnecessárias

for cliente in clientes:

```
    print(cliente.title() + ", obrigado pela preferência.")  
    print("Um bom dia a todos vocês")
```

saída :

```
Rafael, obrigado pela preferência.  
Um bom dia a todos vocês  
Gates, obrigado pela preferência.  
Um bom dia a todos vocês  
Wesley Safadão, obrigado pela preferência.  
Um bom dia a todos vocês
```





# Fazendo listas numéricas

- Usando range()
- Junto com a função list()
- A função list() converte o intervalo do range em uma lista
- Exemplo:  
    Lista = list(range(1,6))  
    print(Lista)  
    [ 1, 2, 3, 4, 5]



# Fazendo listas numéricas

- Também podemos fazer com o for

- Exemplo:

```
lista = []  
for valor in range(1,11)  
    lista.append(valor)  
print(lista)  
[1,2,3,4,5,6,7,8,9,10]
```



# Trabalhando com uma parte da lista

- “Fatiar” uma lista
  - Como se estivessemos acessando um elemento
  - Porém colocamos o “:”
    - Ex.: lista[0:3]
  - Indicamos o índice do primeiro e do último elemento que queremos
  - Retira até o índice anterior do índice indicado
    - Exemplo:  
Lista = [0,1,2,3,4]  
print(lista[0:3])  
Saída:  
[0,1,2]



# Trabalhando com uma parte da lista

- Também podemos fazer um loop
- Basta colocarmos a mesma sintaxe no for
- Exemplo :

```
Lista = [0,1,2,3,4]
```

```
for x in lista[:3]:
```

```
    print(x)
```

Saída:

0

1

2



# Trabalhando com uma parte da lista

- Copiar um lista ou uma parte dela
- Podemos usar o método de “fatiar”
- Para copiar a lista inteira:
  - `copiaDaLista = lista[:]`
  - Ao omitir o índice, o python entende que queremos copiar a lista inteira
- Para copiar uma parte
  - Basta indicar o indice

```
lista = [1,2,3,4]
Copia = lista[0:3]
print(copia)
[1,2,3]
```



# Métodos

- `array.count(x)`
  - Retorna a quantidade ocorrências de x no vetor.
  - Vamos ver um exemplo:



```
firstProgram.py
~/Documents
Open ▼ [icon] Save [menu] [window] [close]

1 posicaoTiros = [ 1,2,3,4,5,5,5,5,8,1,9,7,6,5,10]
2 qualPosicaoProcucar = int(input("qual posição você quer checar ?"))
3 numeroOcorrencias = posicaoTiros.count(qualPosicaoProcucar)
4 print(numeroOcorrencias)
```



# Métodos

- O resultado é:

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
rafael@rafael-Lenovo-ideapad-330-15IKB:~/Documents$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exec(open('firstProgram.py').read())
qual posição você quer checar ?5
5
>>> □
```



# Métodos

- `array.insert(i, x)`
  - Insere um novo item com o `x` no vetor antes da posição `i`. Valores negativos são tratados como sendo em relação ao fim do vetor.



```
*firstProgram.py
~/Documents

1 k = 0
2 x = 0
3 posicao = 0
4 classificacaoDasPatinadoras = []
5 while (k < 5):
6     nota = int(input("Digite a nota da patinadora: "))
7     x = 0
8     if len(classificacaoDasPatinadoras) == 0:
9         classificacaoDasPatinadoras.append(nota)
10    else:
11        while x < len(classificacaoDasPatinadoras) and classificacaoDasPatinadoras[x] > nota :
12            x += 1
13
14        if x < len(classificacaoDasPatinadoras):
15            posicao = x;
16            classificacaoDasPatinadoras.insert(posicao, nota)
17        else:
18            classificacaoDasPatinadoras.append(nota)
19
20    k += 1
21
22 for patinadora in classificacaoDasPatinadoras:
23     print(patinadora)
```





# Métodos

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
Digite a nota da patinadora: 8
Digite a nota da patinadora: 3
Digite a nota da patinadora: 9
Digite a nota da patinadora: 5
Digite a nota da patinadora: 4
9
8
5
4
3
>>> 
```



# Métodos

- `array.pop([i])`
  - Remove o item com o índice `i` do vetor e retorna este item. O valor padrão do argumento é `-1`, assim por padrão o último item é removido e retornado

```
firstProgram.py
~/Documents

1 itemParaRemover = ""
2 itemRemovido = ""
3 aux = 0
4 listaDeItemsNoEstoque = ["Goiaba", "Melão", "Kiwi", "Banana", "Laranja", "Tangerina"]
5 itemParaRemover = input("Digite o nome do item: ").upper()
6 while aux < len(listaDeItemsNoEstoque) and listaDeItemsNoEstoque[aux].upper() != itemParaRemover:
7     aux += 1;
8
9 if aux < len(listaDeItemsNoEstoque):
10     itemRemovido = listaDeItemsNoEstoque.pop(aux).lower()
11     print(f"o item removido é {itemRemovido}\n")
12     print("a lista de items agora é igual a: ")
13     print(listaDeItemsNoEstoque)
14 else:
15     print("Item não encontrado")
```



# Métodos

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
Digite o nome do item: banana
o item removido é banana

a lista de items agora é igual a:
['Goiaba', 'Melão', 'Kiwi', 'Laranja', 'Tangerina']
>>> 
```



# Métodos

- `array.remove(x)`
  - Remove a primeira ocorrência de `x` do vetor.
  - Podemos resolver o problema anterior de outra maneira com o `remove()`

```
firstProgram.py
~/Documents

Open  Save  [Menu]  [Window]  [Close]

1 itemParaRemover = ""
2 itemRemovido = ""
3 aux = 0
4 listaDeItemsNoEstoque = ["Goiaba", "Melão", "Kiwi", "Banana", "Laranja", "Tangerina"]
5 itemParaRemover = input("Digite o nome do item: ")
6 listaDeItemsNoEstoque.remove(itemParaRemover)
7 print(f"o item removido é {itemRemovido}\n")
8 print("a lista de items agora é igual a: ")
9 print(listaDeItemsNoEstoque)
10
```



# Métodos

- O resultado é o mesmo:

Mas cuidado, é preciso fazer o tratamento do erro, pois caso o item não esteja na lista o método retorna um erro

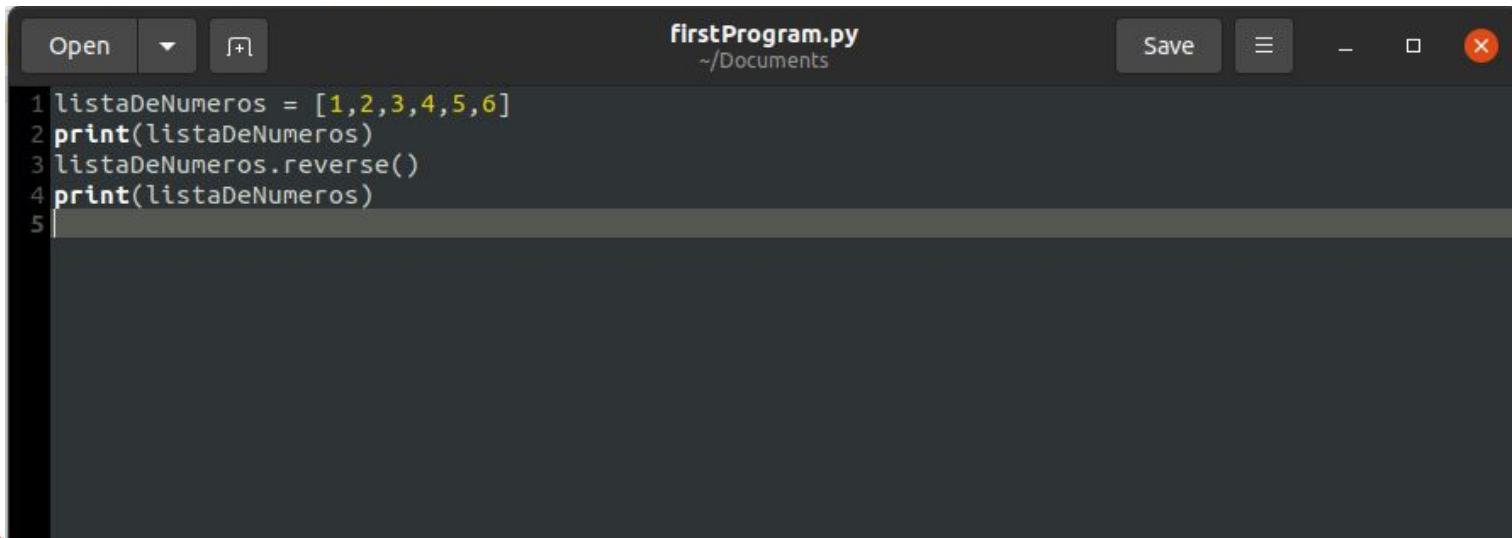
```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
Digite o nome do item: Banana
o item removido é

a lista de itens agora é igual a:
['Goiaba', 'Melão', 'Kiwi', 'Laranja', 'Tangerina']
>>> □
```



# Métodos

- `array.reverse()`
  - Inverte a ordem dos itens no vetor.



```
1 listaDeNumeros = [1,2,3,4,5,6]
2 print(listaDeNumeros)
3 listaDeNumeros.reverse()
4 print(listaDeNumeros)
5
```



# Métodos

- O resultado é:

```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
[1, 2, 3, 4, 5, 6]
[6, 5, 4, 3, 2, 1]
>>> []
```



# list comprehension

- Uma forma de criar e manipular listas.
- Sua sintaxe básica é:
  - `[expr for item in lista]`
- Se fossemos “traduzir” isso, seria equivalente a:
  - aplique a expressão `expr` em cada `item` da `lista`.
- Vamos ver um exemplo.





# list comprehension

- Veja o seguinte código:

```
1 for item in range(10):  
2     lista.append(item**2)  
3
```

- E o resultado é :

```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents  
>>> exec(open('firstProgram.py').read())  
0  
1  
4  
9  
16  
25  
36  
49  
64  
81  
>>>
```

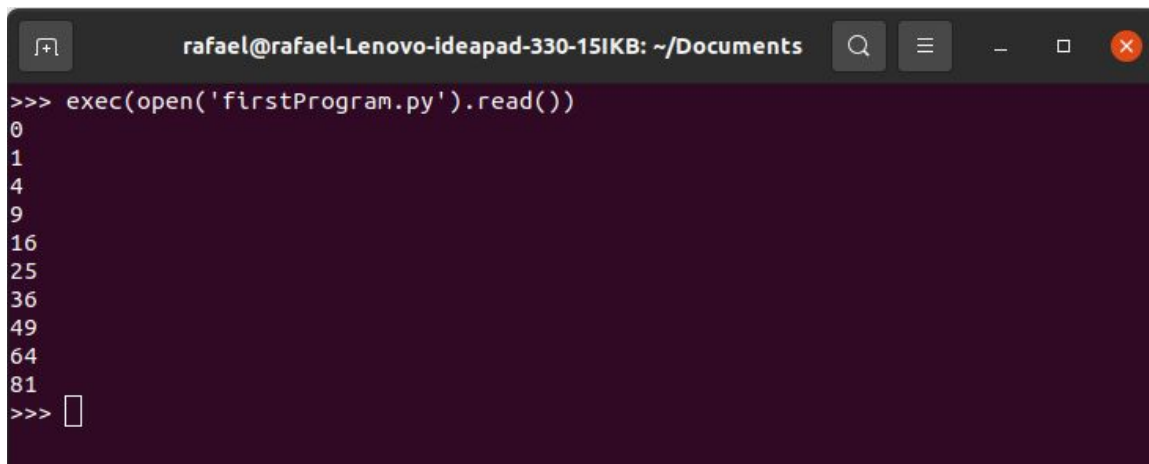


# list comprehension

- Agora vamos ver o mesmo exemplo com list comprehension

```
1 lista = [ item**2 for item in range(10) ]  
2  
3 for item in lista:  
4     print(item)
```

- Veja que o resultado é o mesmo e economizamos algumas linhas:



A terminal window titled 'rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents' showing the execution of a Python script. The command 'exec(open('firstProgram.py').read())' is entered, and the output displays the squares of numbers from 0 to 9: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81. The prompt '>>>' is visible at the bottom.

```
>>> exec(open('firstProgram.py').read())  
0  
1  
4  
9  
16  
25  
36  
49  
64  
81  
>>> █
```



# list comprehension

- Podemos usar condicionais também:
- A sintaxe é:
  - `[expr for item in lista if cond]`
- A tradução seria:
  - Aplique a expressão `expr` em cada `item` da `lista` caso a condição `cond` seja satisfeita.

```
cashier_3 = []  
for item in cart:  
    if item % 2 == 0:  
        cashier_3.append(item)
```

**Non-list comprehension**

```
cashier_3 = [item for item in cart if item % 2 == 0]
```

**List comprehension**



# list comprehension

- Vamos ver um exemplo:

```
1 resultado = [numero for numero in range(20) if numero % 2 == 0]
```

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
>>> exec(open('firstProgram.py').read())
0
2
4
6
8
10
12
14
16
18
>>> 
```

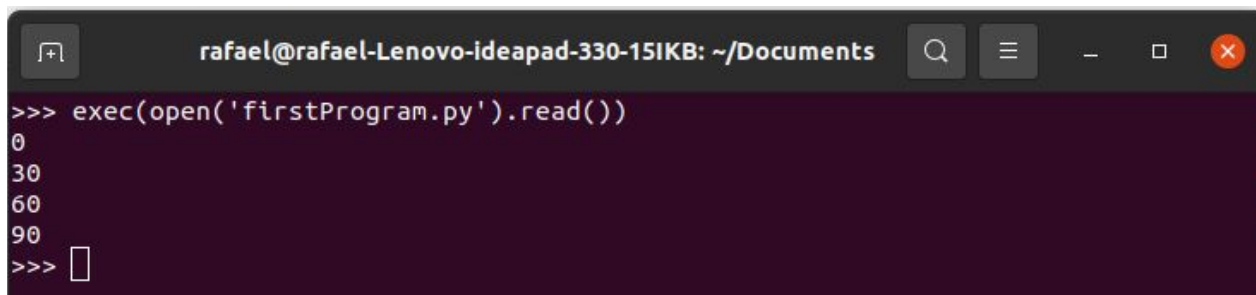


# list comprehension

- Podemos usar vários condicionais:
- Vamos modificar um pouco o código anterior

```
1 resultado = [numero for numero in range(100) if numero % 5 == 0 if numero % 6 == 0]
2
3 for numero in resultado:
4     print(numero)
5
```

- O resultado é :



A terminal window with a dark background. The title bar shows the user 'rafael' on a 'rafael-Lenovo-ideapad-330-15IKB' machine, in the directory '~/Documents'. The prompt is '>>>'. The first command is 'exec(open('firstProgram.py').read())'. The output shows the numbers 0, 30, 60, and 90, each on a new line. The prompt '>>>' is followed by a cursor.

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents
>>> exec(open('firstProgram.py').read())
0
30
60
90
>>> █
```



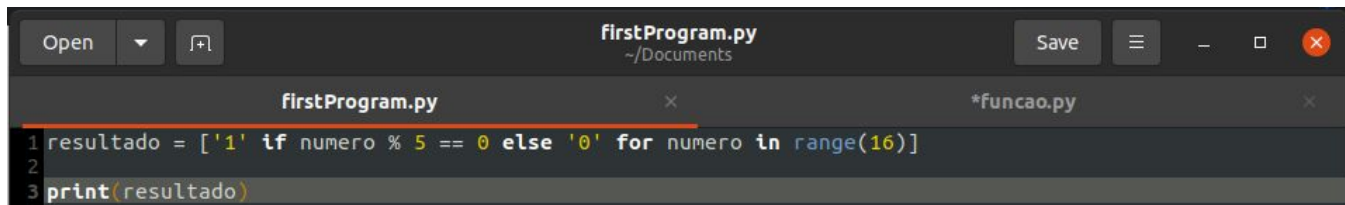
# list comprehension

- Ainda podemos usar if + else:
- A sintaxe básica é :
  - `[resultado_if if expr else resultado_else for item in lista]`
- Em outras palavras: para cada item da lista, aplique o resultado `resultado_if` se a expressão `expr` for verdadeira, caso contrário, aplique `resultado_else`.



# list comprehension

- Vamos ver um exemplo:



A screenshot of a code editor window titled 'firstProgram.py' with a path of '~/.Documents'. The editor shows three lines of Python code: `1 resultado = ['1' if numero % 5 == 0 else '0' for numero in range(16)]`, `2`, and `3 print(resultado)`. The file explorer on the left shows 'firstProgram.py' and '\*funcao.py'.

```
1 resultado = ['1' if numero % 5 == 0 else '0' for numero in range(16)]
2
3 print(resultado)
```

- O resultado é:



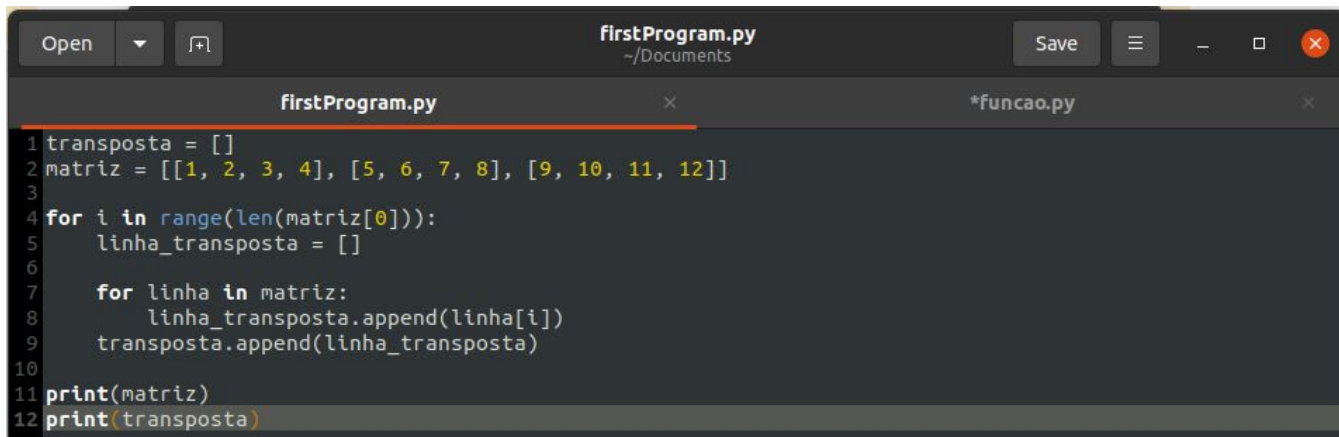
A screenshot of a terminal window with the title 'rafael@rafael-Lenovo-ideapad-330-15IKB: ~/.Documents'. It shows the command `>>> exec(open('firstProgram.py').read())` being executed, followed by the output `['1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1']`. The prompt `>>>` is shown again with a cursor.

```
>>> exec(open('firstProgram.py').read())
['1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1', '0', '0', '0', '0', '1']
>>> █
```



# list comprehension

- Agora vamos deixar um pouco mais elaborado
- Podemos usar múltiplas list comprehension
- Vamos ver um exemplo para achar a transposta de uma matriz



```
firstProgram.py
~/Documents
Save
firstProgram.py
*funcao.py
1 transposta = []
2 matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
3
4 for i in range(len(matriz[0])):
5     linha_transposta = []
6
7     for linha in matriz:
8         linha_transposta.append(linha[i])
9     transposta.append(linha_transposta)
10
11 print(matriz)
12 print(transposta)
```





# list comprehension

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-151KB: ~/Documents
>>> exec(open('firstProgram.py').read())
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
>>> []
```





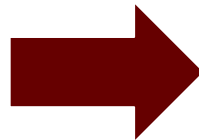
# Matrices



# O que é isso ?

- A matriz é comumente utilizada para a organização de dados
- Pode ser entendida com uma tabela
- Exemplo :

	Pão	Bolo	Salgado
Rafael	1	2	4
gabriel	4	3	2
thiago	4	7	1



$$\begin{pmatrix} 1 & 2 & 4 \\ 4 & 3 & 2 \\ 4 & 7 & 1 \end{pmatrix}$$



# O que é isso ?

- A matriz pode ser indicada como  $A_{m \times n}$ , onde  $m$  e  $n$  são o número de linhas e colunas
- Cada elemento é representado por  $a_{ij}$
- Onde  $i$  é o número da linha e  $j$  o número da coluna

	1	2	3
1	1	2	4
2	4	3	2
3	4	7	1

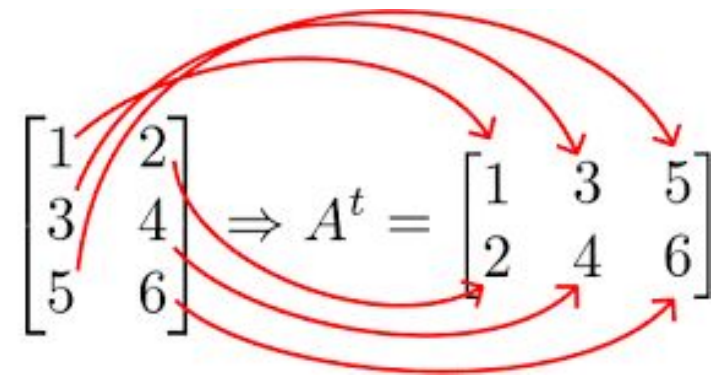
The matrix is enclosed in large blue parentheses. The element 4 in the first row and third column is circled in red, with a red arrow pointing to the label  $a_{13}$ .

Não é “a treze”  
Lemos como “a um três”



# Operações com matrizes

- Assim como fazemos com números, também podemos somar, subtrair e multiplicar matrizes.
- Ainda temos a matriz transposta
  - A transposta de A possui os mesmos elementos de A, mas com a posição das linhas e colunas trocadas, “o que é linha vira coluna e virse versa”
- Vamos ver as outras operações

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow A^t = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$




# Operações com matrizes

- Soma e subtração
  - Para somar ou subtrair duas matrizes, fazemos a operação com cada elemento

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \Rightarrow$$

$$A + B = \begin{bmatrix} 1+1 & 2+2 \\ 3+3 & 4+4 \\ 5+5 & 6+6 \end{bmatrix} \Rightarrow$$

$$C = \begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \end{bmatrix}$$



# Operações com matrizes

- Produto
  - Para multiplicar uma matriz, primeiro precisamos verificar se é possível
    - Para isso verificamos o número de linhas e colunas de cada matriz
    - É preciso que o número de colunas de uma seja igual ao número de linhas da outra

$$A = (a_{ij})_{m \times n} \quad B = (b_{jk})_{n \times p}$$

É possível multiplicar

$$A \cdot B = C = (c_{ik})_{m \times p}$$



# Operações com matrizes

- Produto
  - Após isso, fazemos o produto interno de cada linha de uma matriz com cada coluna da outra matriz

$$B \cdot A = \begin{bmatrix} -1 & 3 \\ 4 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} (-1) \cdot 1 + 3 \cdot 3 & (-1) \cdot 2 + 3 \cdot 4 \\ 4 \cdot 1 + 2 \cdot 3 & 4 \cdot 2 + 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 8 & 10 \\ 10 & 16 \end{bmatrix}$$





# Operações com matrizes

- Existe também a matriz inversa
  - Uma matriz invertível (que possui inversa) é sempre uma matriz quadrada
    - Ou seja, o número de linhas é igual ao número de colunas
  - Um matriz inversa é uma matriz que ao multiplicarmos uma matriz pela sua inversa o resultado será uma matriz identidade (“equivale ao número 1”)

$$\mathbf{A} \text{ é inversa de } \mathbf{A} = \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix} \text{ é } \mathbf{A}^{-1} = \begin{bmatrix} 3 & -1 \\ -5 & 2 \end{bmatrix}$$

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix} \cdot \begin{bmatrix} 3 & -1 \\ -5 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}_2$$

$$\mathbf{A}^{-1} \cdot \mathbf{A} = \begin{bmatrix} 3 & -1 \\ -5 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 \\ 5 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}_2$$



# Operações com matrizes

- Mas como achar a inversa de uma matriz ?

$$A \cdot A^{-1} = I_n$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} a + 2c & b + 2d \\ 3a + 4c & 3b + 4d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



# No python

- No python usamos arrays
- Fazemos arrays de arrays
- Em teoria a matriz anterior seria assim:
  - Teríamos uma array para conter as linhas
    - `matriz = [ ]` ( ela teria 3 posições, já que são 3 linhas )
  - Cada array dentro de matriz representa uma linha
    - `matriz = [ [1,2,4], [ 4, 3, 2 ], [ 4,7,1 ] ]`
  - Cada índice dentro das arrays de linhas representa uma coluna
- Vamos entender melhor como acessar cada elemento.

1	2	4
4	3	2
4	7	1



# No python

- Como acessar um elemento
- primeiro acessamos a linha que queremos
  - `matriz[0]`
- Depois acessamos o elemento da linha
  - `matriz[0][0]`
- Assim, acessamos o elemento da 1ª linha e 1ª coluna
- Lembre-se !!!
  - No python o índice de uma array começa com 0

	0	1	2
0	1	2	4
1	4	3	2
2	4	7	1

```
matriz = [ [1,2,4], [ 4, 3, 2 ], [ 4,7,1 ] ]
```

```
Matriz = [  
    [1,2,4],  
    [4,3,2],  
    [4,7,1]  
]
```



# Mas como criar uma matriz ?

- Podemos utilizar o for

- Vejamos um exemplo:

```
matriz = []
```

```
for x in range(3):
```

```
    linha = []
```

```
    for y in range(3):
```

```
        linha.append(1)
```

```
    matriz.append(linha)
```

```
print(matriz)
```

O primeiro for cria a array que contém uma linha  
Ele itera 3 vezes (a matriz tem três linhas)

O segundo for itera 3 vezes ( a matriz tem 3 colunas )  
E coloca cada elemento em uma posição da lista

Ao fim, coloca cada linha na matriz

Saída

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



# list comprehension

- Com list comprehension o código fica assim:

```
*firstProgram.py
~/Documents

Open Save

*firstProgram.py *funcao.py


1 matriz = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
2 transposta = [[linha[i] for linha in matriz] for i in range(4)]
3
4 print(matriz)
5 print(transposta)
```

- O resultado é :

```
rafael@rafael-Lenovo-ideapad-330-15IKB: ~/Documents

>>> exec(open('firstProgram.py').read())
[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
>>>
```





**VAMOS PRATICAR !**

The background of the slide features a red grid pattern. The grid is composed of horizontal and vertical lines, with additional diagonal lines extending from the corners towards the center, creating a perspective effect. The lines are thin and red, set against a white background.

**Obrigado!!**  
**Ainda com dúvida ?**  
**Entre em contato**