Introdução ao Python e ao Pandas

1 - O interpretador Python mantém o estado de uma célula para outra

In [1]:
```
x = 4
y =1
x + y
```

Out[1]:
```
5
```

In [2]:
```
x
```

Out[2]:
```
4
```

2 - Função que pega dois inteiros e faz a adição.

In [3]:
```
def add_number_1(x,y):
    return x + y

add_number_1(1,2)
```

Out[3]:
```
3
```

3 - Função atualizada para obter um terceiro parâmetro opcional. A função print permite imprimir várias expressões dentro de uma única célula.

In [4]:
```
def add_number_2(x,y, z=None):
    if (z==None):
        return x+y
    else:
        return x + y +z

print(add_number_2('a',str(1)))
print(add_number_2(1,2))
```

```
a1
3
```

4 - Função atualizada para obter um quarto parâmetro True/False

In [5]:
```
def add_number_3(x,y, z=None, flag=False):
    if (flag == True):
        print('Flag!!!')
    if (z==None):
        return x+y
    else:
        return x + y +z

add_number_3(1,2, True)
```

Out[5]:
```
4
```

5 - Atribuir a função à uma variável

In [6]:
```
def add_number_4(x,y):
```

```
    return x + y

a = add_number_4

a(1,2)
```

Out[6]: 3

Tipos e Sequências

6 - Usa-se type para saber o tipo do objeto

In [7]:
```
type('Marina Ramalhete')
```

Out[7]: str

In [8]:
```
type(None)
```

Out[8]: NoneType

In [9]:
```
type(1)
```

Out[9]: int

In [10]:
```
type(add_number)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/var/folders/01/_r7b02r11p15j0s54gb9x0040000gn/T/ipykernel_1978/2699688316.py in <module>
----> 1 type(add_number)

NameError: name 'add_number' is not defined
```

In [13]:
```
type (1.0)
```

Out[13]: float

7 - Tuplas são imutáveis, não podem ser alteradas

In [14]:
```
x = ("a", 1, 'b', 2)
type(x)
```

Out[14]: tuple

8 - Listas são mutáveis

In [15]:
```
x = ["a", 1, 'b', 2]
x
type(x)
```

Out[15]: list

9 - Utiliza-se append para se acrescentar um objeto à lista

In [16]:
```
x.append(3.3)
print(x)
```

```
['a', 1, 'b', 2, 3.3]
```

10 - Loop iterando cada objeto de uma lista

In [17]:
```
for item in x:
    print (item)
```

```
a
1
b
2
3.3
```

11 - Loop usando o operador de indexação

In [18]:
```
i = 0
while (i!=len(x)):
    print(x[i])
    i+=1
```

```
a
1
b
2
3.3
```

12 - Para concatenar listas podemos usar o operador +

In [19]:
```
[1,2] + [3,4]
```

Out[19]:
```
[1, 2, 3, 4]
```

13 - Para repetir listas podemos usar o operador *

In [20]:
```
[1,3]*3
```

Out[20]:
```
[1, 3, 1, 3, 1, 3]
```

14 - Para saber se um objeto faz parte de uma lista temos o operador in

In [21]:
```
1 in [1,2,3]
```

Out[21]:
```
True
```

15 - Notações entre colchetes para fatiar (slice) strings

In [22]:
```
x = 'Isso eh uma string'
print(x[0:2])
```

```
Is
```

16 - Assim retornamos o último caracter da string

In [23]:
```
x[-1]
```

Out[23]:
```
'g'
```

17 - Abaixo, fatiamos do quarto elemento do fim e paramos no penúltimo elemento.

In [24]:
```
x[-4:-2]
```

Out[24]:
```
'ri'
```

18 - Começa com o primeiro caractere e vai até a posição três

In [25]:
```
x[:3]
```

Out[25]:
```
'Iss'
```

19 - Começa no quarto caractere, porque a indexação sempre começa com zero, e vai até o final da lista

In [26]:
```
x[3:-1]
```

Out[26]:
```
'o eh uma strin'
```

20 - As operações que você pode fazer em uma lista, você pode fazer numa string

In [27]:
```
firstname = 'Marina'
lastname = 'Ramalhete'

print(firstname + ' ' + lastname)
print(firstname*3)
print('Mari' in firstname)
```

```
Marina Ramalhete
MarinaMarinaMarina
True
```

21 - Split retorna uma lista contendo todas as palavras de uma string, conforme o caractere de separação

In [28]:
```
firstname = 'Marina Ramalhete'.split(' ')[0]
print(firstname)
```

```
Marina
```

22 - Precisamos ter certeza de convertar objetos antes de utilizá-los

In [29]:
```
'Marina' + 2
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/var/folders/01/_r7b02r11p15j0s54gb9x0040000gn/T/ipykernel_1978/1938104367.py in <module>
----> 1 'Marina' + 2

TypeError: can only concatenate str (not "int") to str
```

In [30]:
```
'Marina' + str(300)
```

Out[30]:
```
'Marina300'
```

23 - Dicionários associam chaves a valores

In [31]:
```
x = {'Vitor Rolla': 'vitorgr@impa.br', 'Artur Avila': 'aavila@impa.br'}
print(x)
```

```
{'Vitor Rolla': 'vitorgr@impa.br', 'Artur Avila': 'aavila@impa.br'}
```

24 - Adicionar novos itens ao dicionário usando o operador de indexação

In [32]:
```
x['Pedro Arthur'] = None
type(x['Pedro Arthur'])
print(x)
```

```
{'Vitor Rolla': 'vitorgr@impa.br', 'Artur Avila': 'aavila@impa.br', 'Pedro Arthur': None}
```

25 - Iterando por todas as chaves

In [33]:
```
for key in x:
    print(key)
```

```
Vitor Rolla
Artur Avila
Pedro Arthur
```

26 - Iterando pelos valores

In [34]:
```
for email in x.values():
    print(email)
```

```
vitorgr@impa.br
aavila@impa.br
None
```

27 - Podemos iterar por chaves e valores simultâneamente

In [35]:
```
for name, email in x.items():
    print(name)
    print(email)
```

```
Vitor Rolla
vitorgr@impa.br
Artur Avila
aavila@impa.br
Pedro Arthur
None
```

28 - Podemos descompactar uma sequencia em diferentes variáveis

In [36]:
```
x = ('Marina', 'Ramalhete', 'Souza')
fname, sname, lname = x
```

In [37]:
```
fname
```

Out[37]:
```
'Marina'
```

In [38]:
```
sname
```

Out[38]:
```
'Ramalhete'
```

29 - É preciso ter certeza que o número de valores sendo descompactados casa com o número de variáveis recebendo os valores

In [39]:
```
x = ('Marina', 'Ramalhete', 'Souza', '123')
fname, sname, lname = x
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/01/_r7b02r11p15j0s54gb9x0040000gn/T/ipykernel_1978/4074702228.py in <module>
      1 x = ('Marina', 'Ramalhete', 'Souza', '123')
----> 2 fname, sname, lname = x

ValueError: too many values to unpack (expected 3)
```

Lendo Arquivos .CSV

30 - Vamos importar um arquivo CSV (mpg.csv), que contém dados sobre o consumo de combustíves de 234 carros

- mpg : miles per gallon
- class : car classification
- cty : city mpg
- cyl : # of cylinders
- displ : engine displacement in liters
- drv : f = front-wheel drive, r = rear wheel drive, 4 = 4wd
- fl : fuel (e = ethanol E85, d = diesel, r = regular, p = premium, c = CNG)
- hwy : highway mpg
- manufacturer : automobile manufacturer
- model : model of car
- trans : type of transmission
- year : model year

In [40]:
```
import csv

%precision 2

with open('./Data/mpg.csv') as csvfile:
    mpg = list(csv.DictReader(csvfile))

mpg[:3]
```

Out[40]:
```
[{'': '1',
  'manufacturer': 'audi',
  'model': 'a4',
  'displ': '1.8',
  'year': '1999',
  'cyl': '4',
  'trans': 'auto(l5)',
  'drv': 'f',
  'cty': '18',
  'hwy': '29',
  'fl': 'p',
  'class': 'compact'},
 {'': '2',
  'manufacturer': 'audi',
  'model': 'a4',
  'displ': '1.8',
```

```
 'year': '1999',
 'cyl': '4',
 'trans': 'manual(m5)',
 'drv': 'f',
 'cty': '21',
 'hwy': '29',
 'fl': 'p',
 'class': 'compact'},
{'': '3',
 'manufacturer': 'audi',
 'model': 'a4',
 'displ': '2',
 'year': '2008',
 'cyl': '4',
 'trans': 'manual(m6)',
 'drv': 'f',
 'cty': '20',
 'hwy': '31',
 'fl': 'p',
 'class': 'compact'}]
```

31 - O comprimento da lista mostra que ela contém 234 dicionários

In [41]:
```
len(mpg)
```

Out[41]:
```
234
```

32 - Podemos usar o método chaves para listar as colunas

In [42]:
```
mpg[0].keys()
```

Out[42]:
```
dict_keys(['', 'manufacturer', 'model', 'displ', 'year', 'cyl', 'trans', 'drv', 'cty', 'hw
y', 'fl', 'class'])
```

33 - Média do gasto de combustível entre todos os carros na cidade

In [43]:
```
sum(float(d['cty']) for d in mpg) / len(mpg)
```

Out[43]:
```
16.86
```

33 - Média do gasto de combustível entre todos os carros na estrada

In [44]:
```
sum(float(d['hwy']) for d in mpg) / len(mpg)
```

Out[44]:
```
23.44
```

34 - Valores únicos do número de cilindros

In [45]:
```
cylinders = set(d['cyl'] for d in mpg)
cylinders
```

Out[45]:
```
{'4', '5', '6', '8'}
```

35 - Exemplo mais complexo: agrupar carros conforme cilindros e encontrar seus gasto de combustível por grupo

In [46]:
```
# iterar sobre todos os níveis de cilindro
```

```
# iterar por todos os dicionários
# se o cilindro combinar,
# adicione no gasto na cidade
# incrementar o contador
# anexar a tupla ('cylinder', 'avg mpg')
```

36 - Quais são as classes de veículos?

In [47]:
```
veiculoClass = set(d['class'] for d in mpg)
veiculoClass
```

Out[47]:
```
{'2seater', 'compact', 'midsize', 'minivan', 'pickup', 'subcompact', 'suv'}
```

37 - Encontrar a média do gasto na estrada por classe de veículos

In [ ]:
```

```

Tempo e Datas

38 - Importar datetime

In [48]:
```
import datetime as dt
import time as tm
```

39 - Hora atual desde 1970

In [49]:
```
tm.time()
```

Out[49]:
```
1637504714.83
```

40 - Converter o timestamp para datetime

In [50]:
```
dtnow = dt.datetime.fromtimestamp(tm.time())
dtnow
```

Out[50]:
```
datetime.datetime(2021, 11, 21, 11, 25, 16, 79876)
```

41 - Atributos do datetime

In [51]:
```
dtnow.hour
```

Out[51]:
```
11
```

42 - Diferença entre datas

In [52]:
```
delta = dt.timedelta(days =100)
delta
```

Out[52]:
```
datetime.timedelta(days=100)
```

43 - Today retorna a data corrente

In [53]:
```
today = dt.date.today()
```

In [54]:
```
today - delta
```

Out[54]:
```
datetime.date(2021, 8, 13)
```

In [55]:
```
today > today - delta
```

Out[55]:
```
True
```

Python Avançado - Orientação a Objetos

44 - Exemplo de classe em Python

In [57]:
```
class Person:
    department = 'UFF'

    def set_name(self, new_name):
        self.name = new_name
    def set_location(self, new_location):
        self.location =new_location
```

45 - Instanciando a classe

In [58]:
```
person = Person()
person.set_name('Marina Ramalhete')
print(person.name)
```

```
Marina Ramalhete
```

46 - Mapeando a função min() entre duas listas

In [59]:
```
store1 = [12.00, 11.00, 12.34, 2.34]
store2 = [9.00, 10.00, 11.34, 2.01]
cheapest = map(min, store1, store2)
cheapest
```

Out[59]:
```
<map at 0x7fbcc3d04670>
```

47 - Iterando sobre o objeto mapeado

In [60]:
```
list(cheapest)
```

Out[60]:
```
[9.00, 10.00, 11.34, 2.01]
```

Python Avançado - Lambdas & LC

48 - Exemplo de função lambda que recebe 3 parâmetros e soma os dois primeiros

In [61]:
```
my_function = lambda a, b, c: a + b
```

49 - Retorna uma referência

In [62]:
```
my_function(1,2,3)
```

Out[62]:
```
3
```

50 - Vamos iterar de 0 a 999 e retornar os números pares

In [1]:
```
my_list = []
for number in range(0,1000):
    if number % 2 == 0:
        my_list.append(number)
# my_list
```

51 - A mesma coisa da anterior, mas com list comprehension

In [2]:
```
my_list = [number for number in range(0,1000) if number % 2==0]
# my_list
```

Python Avançado - Numpy

52 - Importar Numpy

In [65]:
```
import numpy as np
```

53 - Criar lista e trasformar e numpy array

In [66]:
```
mylist = [1, 2, 3]
x = np.array(mylist)
x
```

Out[66]:
```
array([1, 2, 3])
```

54 - Passando a lista diretamente também funciona

In [67]:
```
y = np.array([4, 5, 6])
y
```

Out[67]:
```
array([4, 5, 6])
```

55 - Lista de listas para criar arrays multidimensionais

In [68]:
```
m = np.array([[7, 8, 9], [10, 11, 12]])
m
```

Out[68]:
```
array([[ 7,  8,  9],
       [10, 11, 12]])
```

56 - O método shape retorna as dimensões de um array

In [69]:
```
m.shape
```

Out[69]:
```
(2, 3)
```

57 - Retorna valores espaçados, começa em 0 e vai de 2 a 2 até 30

In [70]:
```
n = np.arange(0, 30, 2) # start at 0 count up by 2, stop before 30
n
```

Out[70]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28])
```

58 - Mesmos dados em uma nova forma

In [71]:
```
n = n.reshape(3, 5) # reshape array to be 3x5
n
```

Out[71]:
```
array([[ 0,  2,  4,  6,  8],
       [10, 12, 14, 16, 18],
       [20, 22, 24, 26, 28]])
```

59 - Retorna 9 valores espaçados entre 0 e 4

In [72]:
```
o = np.linspace(0, 4, 9) # return 9 evenly spaced values from 0 to 4
o
```

Out[72]:
```
array([0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ])
```

60 - Muda a forma e o tamanho do array

In [73]:
```
o.resize(3, 3)
o
```

Out[73]:
```
array([[0. , 0.5, 1. ],
       [1.5, 2. , 2.5],
       [3. , 3.5, 4. ]])
```

61 - Matriz de 1s

In [74]:
```
np.ones((3, 2))
```

Out[74]:
```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

62 - Matriz de 0s

In [75]:
```
np.zeros((2, 3))
```

Out[75]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

63 - Matriz com 1s nas diagonais e 0s no resto

In [76]:
```
np.eye(3)
```

Out[76]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

64 - Constrói uma matriz diagonal

In [77]:
```
np.diag(y)
```

Out[77]:
```
array([[4, 0, 0],
       [0, 5, 0],
```

```
       [0, 0, 6]])
```

65 - Constrói uma matriz repetindo uma lista

In [78]:
```
np.array([1, 2, 3] * 3)
```

Out[78]:
```
array([1, 2, 3, 1, 2, 3, 1, 2, 3])
```

66 - Repetindo os elementos da matriz

In [79]:
```
np.repeat([1, 2, 3], 3)
```

Out[79]:
```
array([1, 1, 1, 2, 2, 2, 3, 3, 3])
```

67 - Matriz de 1s dois por três

In [80]:
```
p = np.ones([2, 3], int)
p
```

Out[80]:
```
array([[1, 1, 1],
       [1, 1, 1]])
```

68 - Empilhamento vertical

In [81]:
```
np.vstack([p, 2*p])
```

Out[81]:
```
array([[1, 1, 1],
       [1, 1, 1],
       [2, 2, 2],
       [2, 2, 2]])
```

69 - Empilhamento horizontal

In [82]:
```
np.hstack([p, 2*p])
```

Out[82]:
```
array([[1, 1, 1, 2, 2, 2],
       [1, 1, 1, 2, 2, 2]])
```

70 - Use + , − , ∗ , / e ∗∗ para executar adição, subtração, multiplicação, divisão e potência "element wise"

In [83]:
```
print(x + y) # [1 2 3] + [4 5 6] = [5  7  9]
print(x - y) #   [1 2 3] - [4 5 6] = [-3 -3 -3]
```

```
[5 7 9]
[-3 -3 -3]
```

In [84]:
```
print(x * y) #   [1 2 3] * [4 5 6] = [4  10  18]
print(x / y) #   [1 2 3] / [4 5 6] = [0.25  0.4  0.5]
```

```
[ 4 10 18]
[0.25 0.4  0.5 ]
```

In [85]:
```
print(x**2) #   [1 2 3] ^2 =  [1 4 9]
```

```
[1 4 9]
```

71 - Produto Escalar

$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \cdot \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3$$

In [86]:
```
x.dot(y)  # dot product  1*4 + 2*5 + 3*6
```

Out[86]:
```
32
```

72 - Matriz anterior Y e seus valores quadráticos em uma nova matriz Z

In [87]:
```
z = np.array([y, y**2])
print(len(z)) # number of rows of array
z
```

Out[87]:
```
2
array([[ 4,  5,  6],
       [16, 25, 36]])
```

73 - Antes da transposição

In [88]:
```
z.shape
```

Out[88]:
```
(2, 3)
```

74 - Depois da transposição

In [89]:
```
z.T
```

Out[89]:
```
array([[ 4, 16],
       [ 5, 25],
       [ 6, 36]])
```

75 - Shape depois da transposição

In [90]:
```
z.T.shape
```

Out[90]:
```
(3, 2)
```

76 - Para ver o tipo dos elementos da matriz

In [91]:
```
z.dtype
```

Out[91]:
```
dtype('int64')
```

77 - Para fazer cast de tipos

In [92]:
```
z = z.astype('f')
z.dtype
```

Out[92]:
```
dtype('float32')
```

78 - Métodos para se executar em matrizes

In [93]:
```
a = np.array([-4, -2, 1, 3, 5])
```

In [94]:
```
a.sum()
```

Out[94]:
```
3
```

In [95]:
```
a.max()
```

Out[95]:
```
5
```

In [96]:
```
a.min()
```

Out[96]:
```
-4
```

In [97]:
```
a.mean()
```

Out[97]:
```
0.6
```

In [98]:
```
a.std()
```

Out[98]:
```
3.2619012860600183
```

79 - `argmax` e `argmin` retornam o index do máximo e mínimo valores na matriz

In [99]:
```
a.argmax()
```

Out[99]:
```
4
```

In [100…
```
a.argmin()
```

Out[100…
```
0
```

80 - Vetor com quadrados de 0 a 12

In [101…
```
s = np.arange(13)**2
s
```

Out[101…
```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144])
```

81 - Exemplos de indexação

In [102…
```
s[0], s[4], s[-1]
```

Out[102…
```
(0, 16, 144)
```

82 - Array[start:stop], se houver um deles vazio começa no início ou fim

In [103…
```
s[1:5]
```

Out[103…
```
array([ 1,  4,  9, 16])
```

83 - Números negativos contam pelas costas (começando pelo final)

In [104…
```
s[-4:]
```

Out[104…
```
array([ 81, 100, 121, 144])
```

84 - Um segundo "dois pontos" pode ser usado para indicar o tamanho do passo. Nesse exemplo começamos contando do quinto elemento de trás para frente, dando passos -2 até o início do array

In [105…
```
s[-5::-2]
```

Out[105…
```
array([64, 36, 16,  4,  0])
```

85 - Arrays multidimensionais

In [106…
```
r = np.arange(36)
r.resize((6, 6))
r
```

Out[106…
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 31, 32, 33, 34, 35]])
```

86 - Indexação multidimensional

In [107…
```
r[2, 2]
```

Out[107…
```
14
```

87 - Fatiamento multidimensional

In [108…
```
r[3, 3:6]
```

Out[108…
```
array([21, 22, 23])
```

88 - Fatiar todas as linhas até a linha 2 (sem incluí-la), e todas as colunas até a última (sem incluí-la)

In [109…
```
r[:2, :-1]
```

Out[109…
```
array([[ 0,  1,  2,  3,  4],
       [ 6,  7,  8,  9, 10]])
```

89 - Fatiamento da última linha, com um passo de 2 na coluna

In [110…
```
r[-1, ::2]
```

Out[110…
```
array([30, 32, 34])
```

90 - Indexação condicional

In [111…
```
r[r > 30]
```

Out[111…
```
array([31, 32, 33, 34, 35])
```

91 - Atribuição de valores com indexação condicional

In [112…
```
r[r > 30] = 30
r
```

Out[112…
```
array([[ 0,  1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10, 11],
       [12, 13, 14, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 30, 30, 30, 30, 30]])
```

92 - Cópia vs. Fatiamento: Atenção `r2` é um fatiamento de `r`

In [113…
```
r2 = r[:3,:3]
r2
```

Out[113…
```
array([[ 0,  1,  2],
       [ 6,  7,  8],
       [12, 13, 14]])
```

93 - Atribuir os elementos da fatia r2 para 0

In [114…
```
r2[:] = 0
r2
```

Out[114…
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

94 - r também mudou!!!

In [115…
```
r
```

Out[115…
```
array([[ 0,  0,  0,  3,  4,  5],
       [ 0,  0,  0,  9, 10, 11],
       [ 0,  0,  0, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 30, 30, 30, 30, 30]])
```

95 - Para copiar sem afetar o original use r.copy()

In [116…
```
r_copy = r.copy()
r_copy
```

Out[116…
```
array([[ 0,  0,  0,  3,  4,  5],
       [ 0,  0,  0,  9, 10, 11],
       [ 0,  0,  0, 15, 16, 17],
       [18, 19, 20, 21, 22, 23],
       [24, 25, 26, 27, 28, 29],
       [30, 30, 30, 30, 30, 30]])
```

96 - Se mudarmos r.copy, não mudamos r

In [117…
```
r_copy[:] = 10
print(r_copy, '\n')
```

```
print(r)
```

```
[[10 10 10 10 10 10]
 [10 10 10 10 10 10]
 [10 10 10 10 10 10]
 [10 10 10 10 10 10]
 [10 10 10 10 10 10]
 [10 10 10 10 10 10]]

[[ 0  0  0  3  4  5]
 [ 0  0  0  9 10 11]
 [ 0  0  0 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 30 30 30 30 30]]
```

97 - Criar uma matriz 4x3 com números aleatórios de 0 a 9

In [118…

```
test = np.random.randint(0, 10, (4,3))
test
```

Out[118…

```
array([[0, 4, 0],
       [6, 3, 5],
       [7, 0, 0],
       [4, 1, 8]])
```

98 - Iterando por linhas

In [119…

```
for row in test:
    print(row)
```

```
[0 4 0]
[6 3 5]
[7 0 0]
[4 1 8]
```

99 - Iterando por index

In [120…

```
for i in range(len(test)):
    print(test[i])
```

```
[0 4 0]
[6 3 5]
[7 0 0]
[4 1 8]
```

100 - Iterando por linhas e index

In [121…

```
for i, row in enumerate(test):
    print('row', i, 'is', row)
```

```
row 0 is [0 4 0]
row 1 is [6 3 5]
row 2 is [7 0 0]
row 3 is [4 1 8]
```

In [ ]: