"You take the red pill and you stay in Wonderland and I show you how deep the rabbit-hole goes. Remember: all I am offering is the truth, nothing more ."
-- Morpheus, *The Matrix*

ITI 1120
Module 7: Arrays

General Concepts:

1. *Arrays and 2D lists*
2. *Algorithms on arrays*

General Objectif: *Solve problems with arrays.*

Theme 1: Arrays and lists 2D

Sub-theme: Arrays

- An array L x C has L lines and C colonnes.
- Example. An array 4 x 6 of integers (0-100)

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 88 & 70 & 72 \\ 87 & 0 & 0 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 74 & 74 \end{bmatrix}$$

M[r][c] represents the input at the intersection of row r and the colonne c. This notion can extended to 3, 4, … *n* dimensions.
(Note: indexes start at 0).

Arrays

$$M = \begin{bmatrix} 71 & 62 & 33 & 89 & 85 & 74 \\ 68 & 65 & 75 & 88 & 70 & 72 \\ 87 & 0 & 0 & 90 & 92 & 88 \\ 58 & 72 & 66 & 57 & 74 & 74 \end{bmatrix}$$

- An array is represented in our algorithms by a two dimensions list (a list of lists). Each inside list must have the same size. Otherwise, we have a 2D list but not an array.

- Exercise: the array M is a list of 4 lists, each having 6 elements. Cnnsequently:

  M[1][2] contains ?

  M[2][5] contains ?

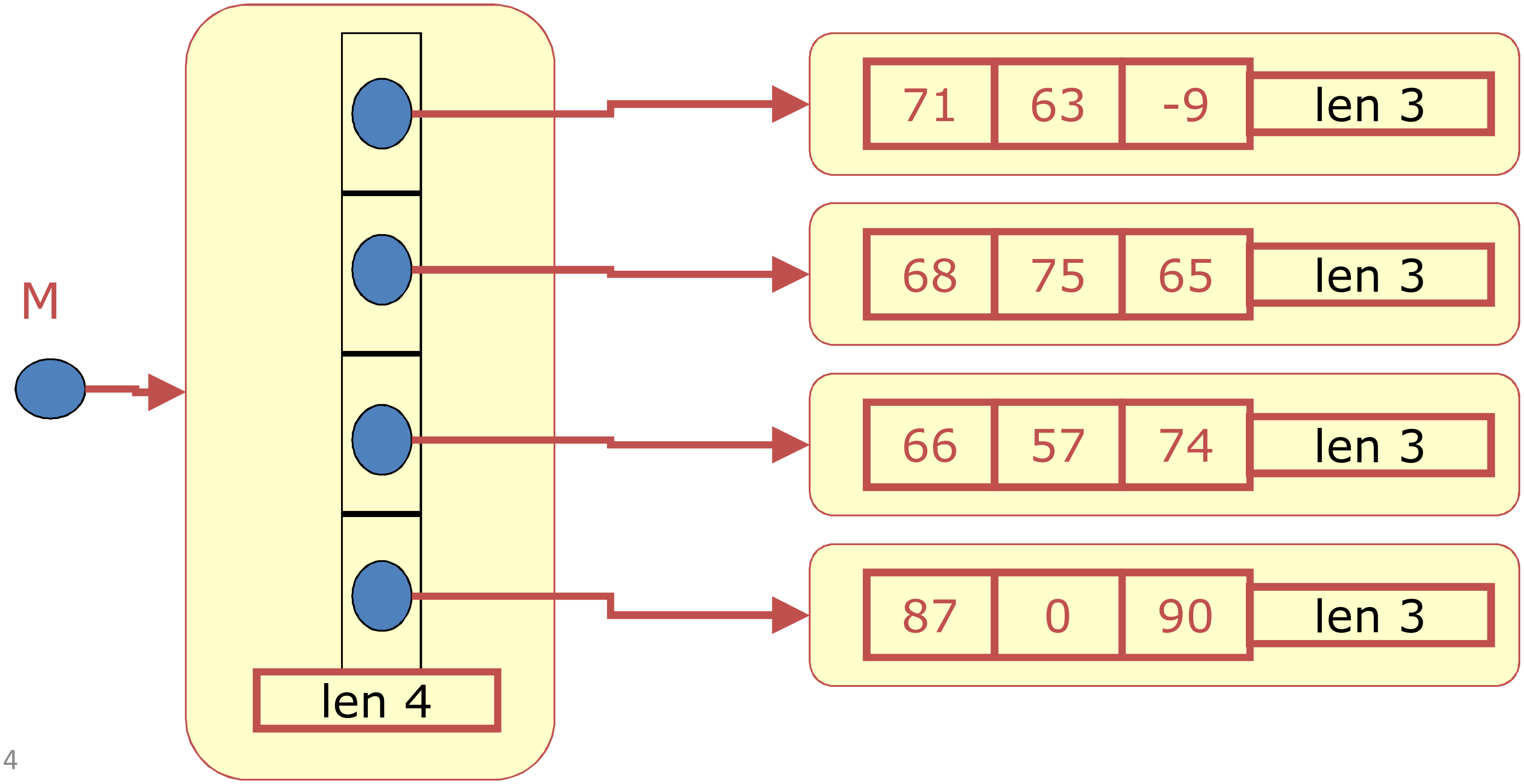
  M[4][1] contains ?

  M[3] contains ?

Responses: 75, 88, error, [58 72 66 57 74 74]

Sub-theme: Arrays in Python

- A 2D list in Python is a list of lists; each element of the first list is a reference to a list. If the inside lists have the same size, the 2D list is an array.



Variables of 2D list type

- To create and initialize a 2D list (array of 2x3)

```
>>> m = [[1, 2, 3], [4, 5, 6]]
>>> print(m)
>>> [[1, 2, 3], [4, 5, 6]]
```

- The function **len** returns tha size (number of lines/rows):

```
>>> len(m)
>>> 2
>>> len(m[0])    # number og columns?
>>> 3
```

What is the value of `len(m[0][0])`?

- 2D List, but not array

```
>>> list1 = [[1,2], [3,4,5]]
```

- 3D List 3D (2x2x2)

```
>>> m3 = [[[1,2],[3,4]],[[5,6],[7,8]]]
>>> m3[0][0][0]
>>> 1
```

Sub-theme: The display of an array

```
for i in matrix:    # visit each row
  for j in i:       # visit each element of that row
    print(j, end=" ")
  print()

# alternative
i = 0
while i < len(matrix):
  j = 0
  while j < len(matrix[i]):
    print(matrix[i][j], end=" ")
    j = j + 1
  i = i + 1
  print()
```

Sub-theme: Reading an array from the keyboard

```
m = int(input("Enter the number of rows: "))
n = int(input("Enter the number of columns: "))
matrix = []
i = 0
while (i < m):
  j = 0
  matrix.append([])
  while j < n:
     v = int(input("matrix["+str(i)+","+str(j) +"]="")
     matrix[i].append(v)
     j = j + 1
  i = i + 1
```

Reading an array from the keyboard (version 2)

```
m = int(input("Enter the number of rows: "))
matrix = []
i = 0
while (i < m):
     print("Enter the row", i,
           "(integers separated by spaces)")
     line = [int(val) for val in input().split()]
     matrix.append(line)
     i = i + 1
```

Reading an array from the keyboard (version 3)

```
print("Enter the number of columns, with spaces.")
print(« A row per line, and an empty line at the end.")
matrix = []
while True:
      line = input()
      if not line: break
      values = ligne.split()
      row = [int(val) for val in values]
      matrix.append(row)
```

Question

What does Python display?

```
>>> m = [['a','b','c'], ['d','e'], ['f']]
>>> print(len(m), len(m[0]), len(m[2]), m[0][2])
```

a) 3, 2, 1, a, b, c

b) 3, 3, 2, 1, c

c) 3, 3, 3, a

d) 3, 3, 1, c

Question – *Solution:*

What does Python display?

```
>>> m = [['a','b','c'], ['d','e'], ['f']]
>>> print(len(m), len(m[0]), len(m[2]), m[0][2])
```

a) 3, 2, 1, a, b, c

b) 3, 3, 2, 1, c

c) 3, 3, 3, a

d) 3, 3, 1, c

Correcte response: d)

Explaination: m is a list with 3 elements that are lists, thus len(m) is 3. m[0] is a list with 3 elements. m[2] is a list with with 1 element. m[0][2] is ‘c’.

Theme 2: Algorithms on arrays

.

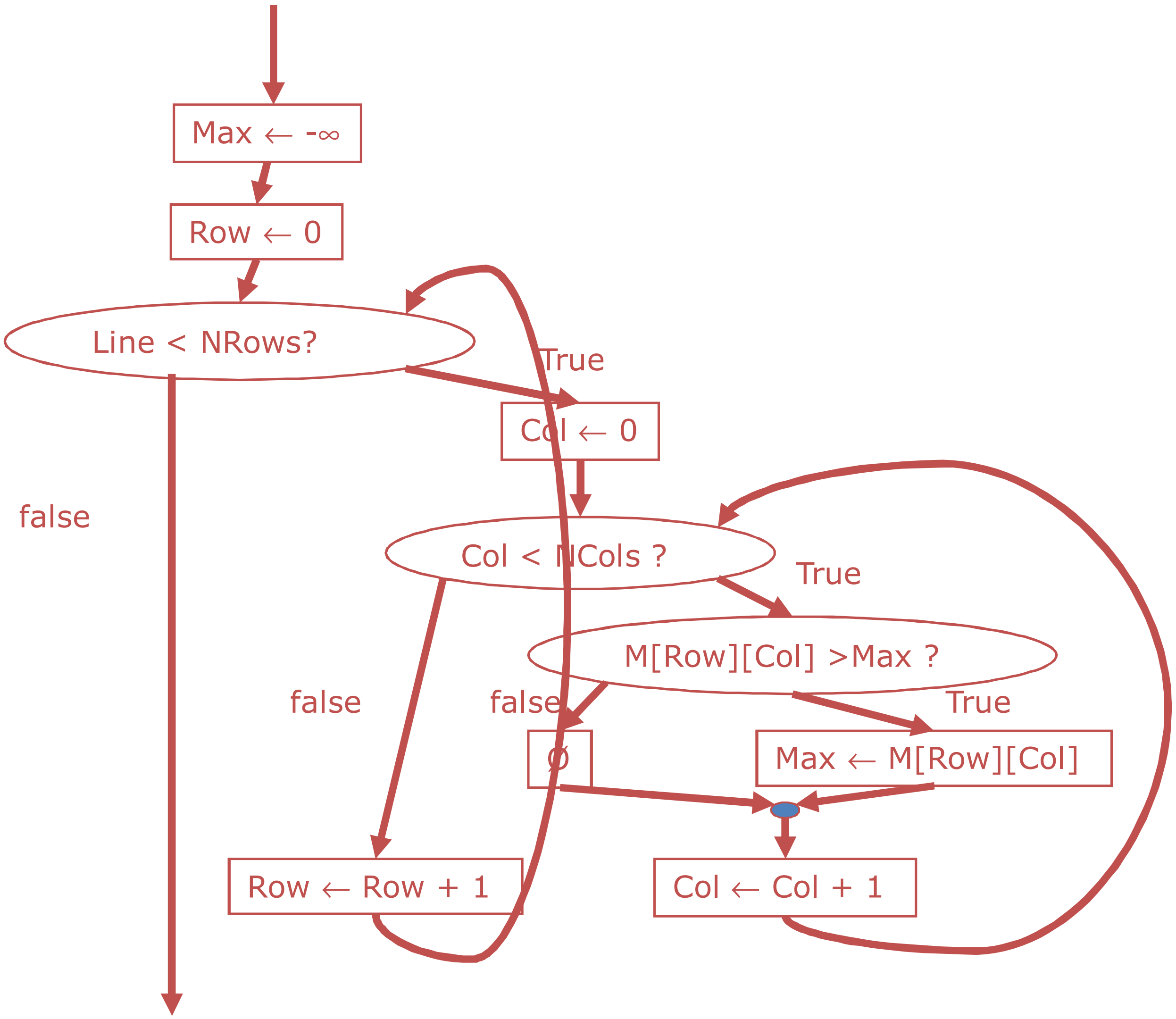
Subs-theme: Maximum value in an array

Derive an algorithm that finds the maximum value in an array:

DATA:	M	*(reference to a matrix)*
	NRows	*(number of rowa in M)*
	NCols	*(number of columns in M)*
INTERMEDIARIES:	Row	*(index of the courant row)*
	Col	*(index of the courant column)*
RESULT:	Max	*(maximum value)*
HEADER:	Max ← FindMaxArray(M, NRows, NCols)	

Valeur Maximum value in an array (suite)

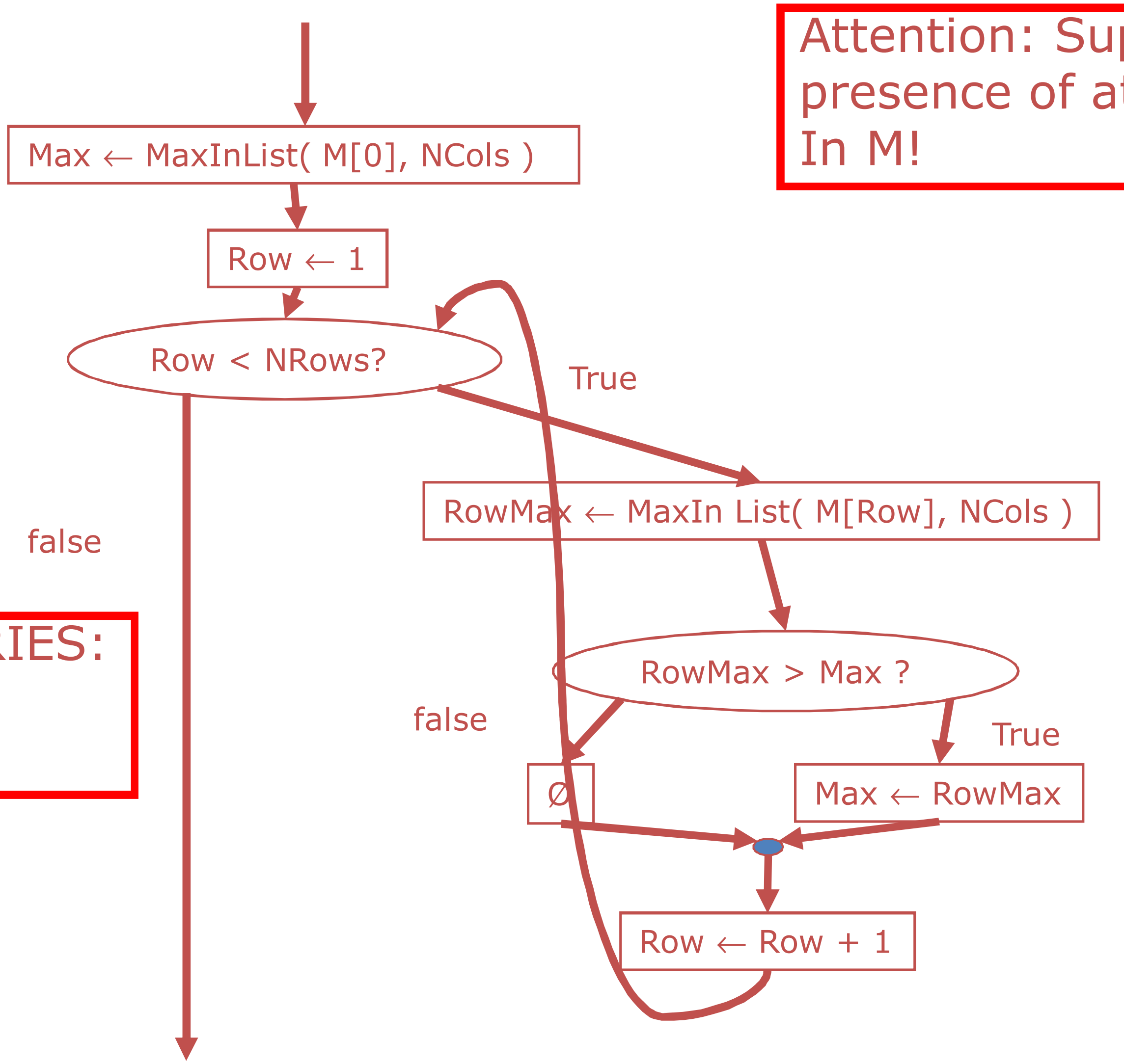
MODULE:



Alternative Algorithm
(uing MaxInList, from module 5 or max from Python)

MODULE:

Attention: Suppose the presence of at least a row In M!



Max ← MaxInList(M[0], NCols)

Row ← 1

Row < NRows?

True

false

RowMax ← MaxIn List(M[Row], NCols)

RowMax > Max ?

false

True

∅

Max ← RowMax

Row ← Row + 1

INTERMÉDIARIES:
RowMax
Instead of Col

Exercise: Convert in Python the first algorithm that finds the maximum value in an array

```
def maxArray(m):
   max = - float('Inf') # minus the infinity
                        # m[0][0] could be another option
                        # if the array is not empty!
   row = 0
   while row < len(m):
      col = 0
      while col < len(m[row]):
         if m[row][col] > max:
            max = m[row][col]
         col = col + 1
      row = row + 1
   return max
```

Exercise: Find the maximum value in an array using *for* loops

```
def maxArray(m):
    max = - float('Inf')      # minus the infinity
              # m[0][0] could be another option
              # if the array mis not empty!
    for row_val in m:
      for col_val in row_val:
         if col_val > max:
            max = col_val
    return max
```

Exercise: Convert in Python the second algorithm that finds the maximum value in an array.

```
def maxArray(m):
   max = max(m[0])  # max in the first row using
                    # Python function max
   row = 1          # start from row 1

   while row < len(m):
      rowMax = max(m[row])
      if rowMax > max:
         max = rowMax
      row = row + 1
   return max
```

Sub-theme: Diagonal matrix

- A square matrix has the same number of rows and columns. If the values in the two triangles surrounding the diagonal are 0, then it is a diagonal matrix. Forr example:

$$M1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad M2 = \begin{bmatrix} 2 & 4 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

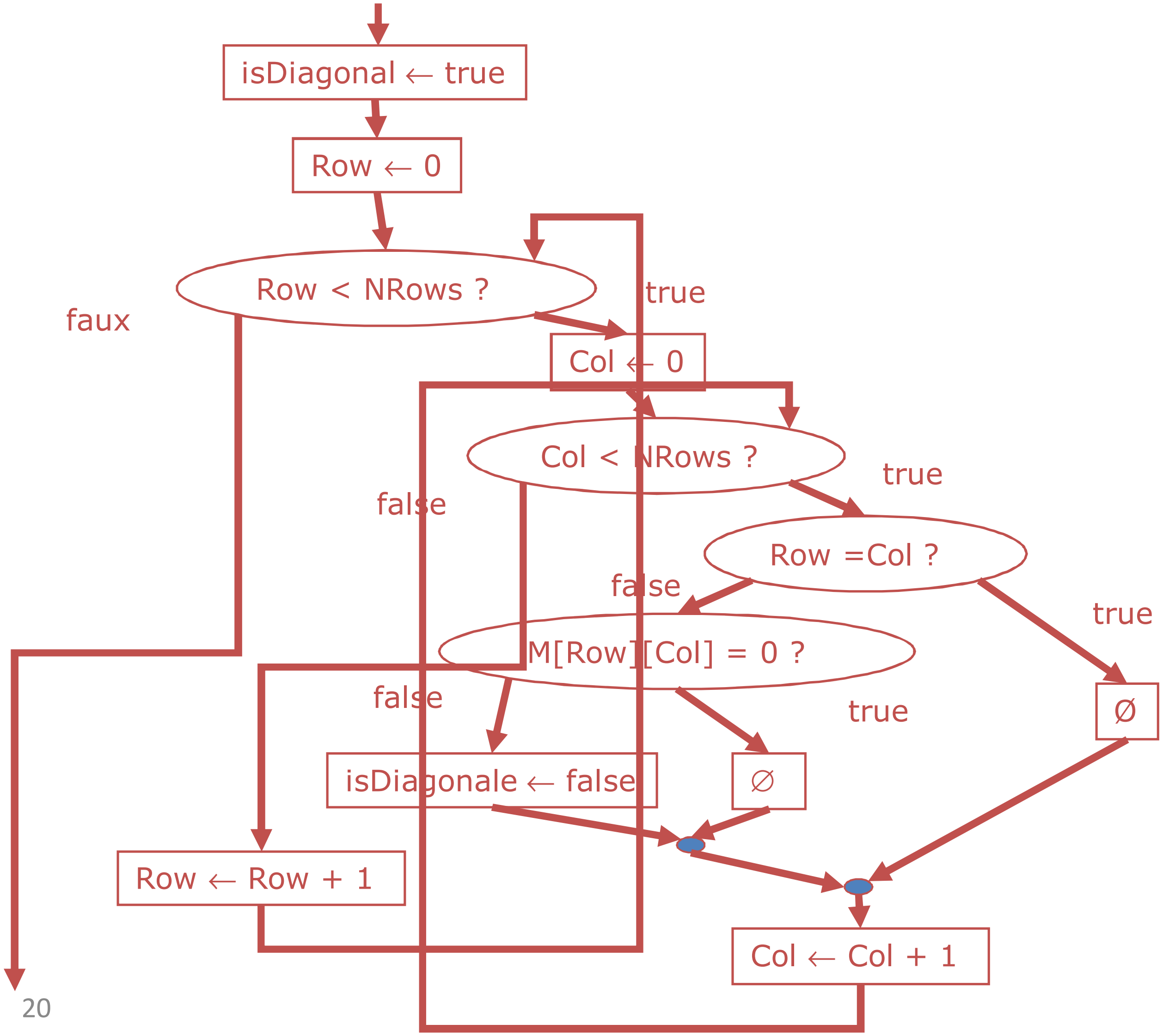
- *M1* is a diagonal matrix while *M2* is not.
- Derive an algorithm that checks if a square matrix is diagonal.

Algorithme checkDiag

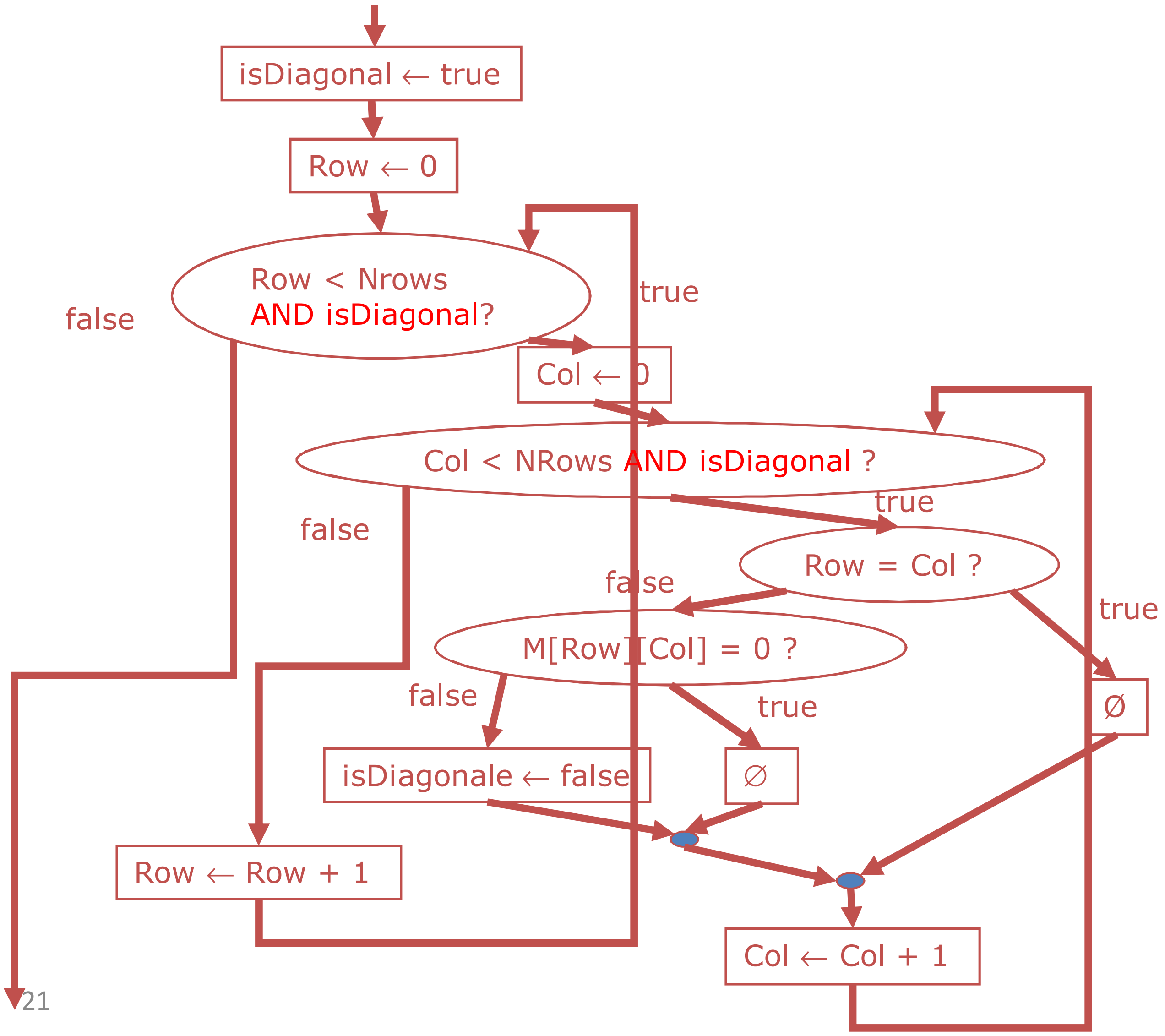
- An algorithm that checks if a square matrix is doagonal:

DATA:	M	*(référence to a matrix)*
	NRows	*(numbers of rows in M)*
		(also the number of columns)
INTERMÉDIARIES:	Row	*(index of the courant row)*
	Col	*(index of the courant column)*
RESULT:	isDiagonal	*(Boolean: true id M is diagonale)*
EN-TÊTE:	isDiagonal ← CheckDiag(M, NRows)	

Algorithme checkDiag (suite)



Algorithm checkDiag (the efficient Version



Exercise: Convert algorithm checkDiag in Python (the efficient version)

```
def checkDiag(m):
   ''' (list) -> bool
   '''
   isDiagonal = True
   row = 0
   while row < len(m) and isDiagonal:
      col = 0
      while col < len(m[row]) and isDiagonal:
        if (rang != col):
          if m[row][col] != 0:
              isDiagonal = False
        col = col + 1
      row = row + 1
   return isDiagonal
```

Exercice: Effacer une rangée d’une matrice

- Écrivez un programme Python pour effacer une rangée d’une matrice.

Solution: Erase one row in a matrix

```
def eraseRow(m, r):
    ''' (list, int) -> None
    Erase the row r
    Precondition: r is from 0 to len(m)-1
    '''
    del(m[r])
```

Exercise: Erase a column in a matrix

- Derive a Python program that erases a column in a matrix.

Solution: Erase a column in a matrix

```
def eraseCol(m, c):
    ''' (list, int) -> None
    Erase the column c
    Precondition: c is a valid index
    '''
    i = 0
    while i < len(m):
        del(m[i][c])
        i = i + 1
```

Question

Which lines in the following codes are equivalentes to the code in the body of that function?

```
def reverse(m, c1, c2):
  '''(list, int, int) -> None
  Exchange columns c1 and c2
  Preconditions: m is a matrix,
  c1 and c2 are valide values for the columns index.
  >>> m = [['a','b','c'], ['d','e','f'], ['g','h','i']]
  >>> reverse(m, 0, 2)
  >>> m
  ['c', 'b', 'a'], ['f', 'e', 'd'], ['i', 'h', 'g']]
  '''
  i = 0
  while i < len(m):
      temp = m[i][c1]
      m[i][c1] = m[i][c2]
      m[i][c2] = temp
      i = i + 1
```

Question (suite)

```
I
while i < len(m):
      m[i][c1], m[i][c2] = m[i][c2], m[i][c1]
      i = i + 1

II
for v in m:
    v[c1], v[c2] = v[c2], v[c1]

III
i = 0
while i < len(m):
   m[c1], m[c2] = m[c2], m[c1]
    i = i + 1
```

Possible responses:

a) I
b) II
c) III
d) I et II
e) I, II, et II

Question – *Solution*:

Correcte response: d)
Explaination: I and II exchange both columns. III exchanges rows.

Conclusion

- We can use 2D, 3D, or even larger dimensions lists.
- A matrix is a 2D list where inside lists have the same size.
- We visit matrixes with a loop for each row and an imbricated loop for each column (to visit each element of the courant row).