ITI 1520
Module 5: Listes

@2015 Diana Inkpen, University of Ottawa, All rights reserved

General Concepts:

1. Lists and loops
2. Chain of characters and loops

Click to edit Master subtitle style

General Objectif: Developp programs in Python using lists and character chains.

Learning target:

1. Solve problems in Python with lists and loops.
2. Solve problems in Python with chains of characters and loops.

Theme 1. Lists and loops

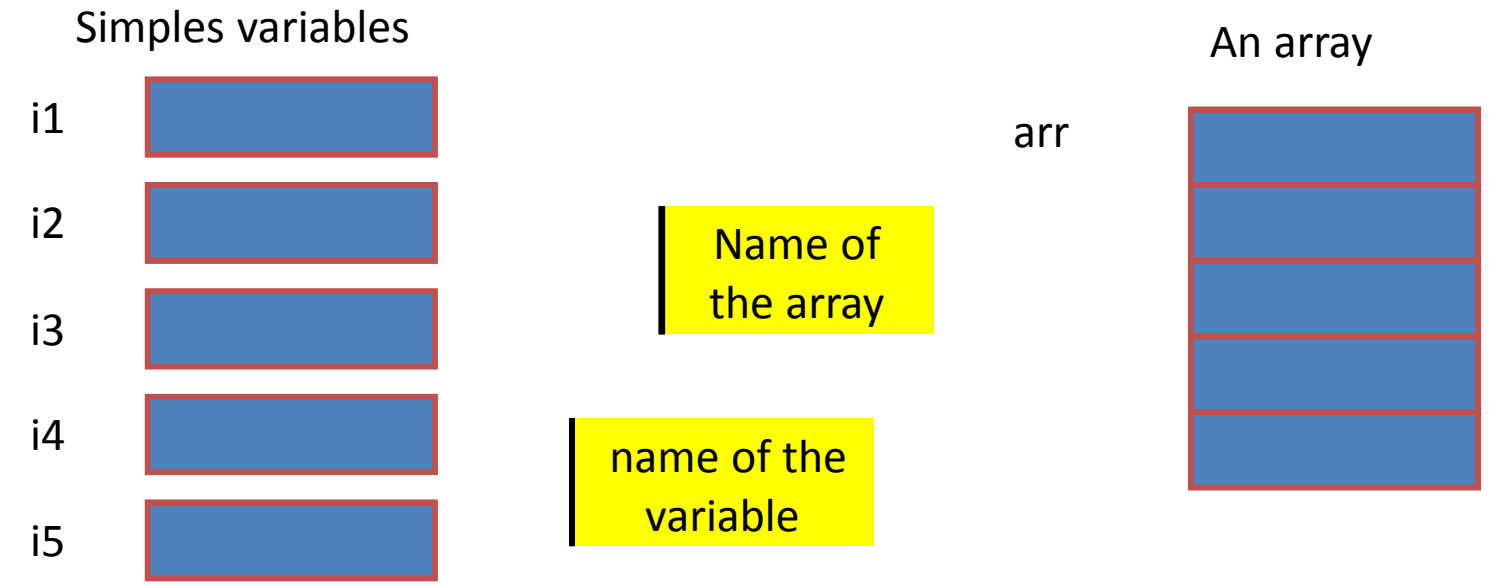
Subs-theme: Probleme with simple variables

- Assume the module of an algorithm reads 5 integers and display them in reverse order:

```
Module:
i1 ☐ ReadInteger()
i2 ☐ ReadInteger()
i3 ☐ ReadInteger()
i4 ☐ ReadInteger()
i5 ☐ ReadInteger()
DisplayLine(i5)
DisplayLine(i4)
DisplayLine(i3)
```

Sub-theme: Computer Computer Arrays

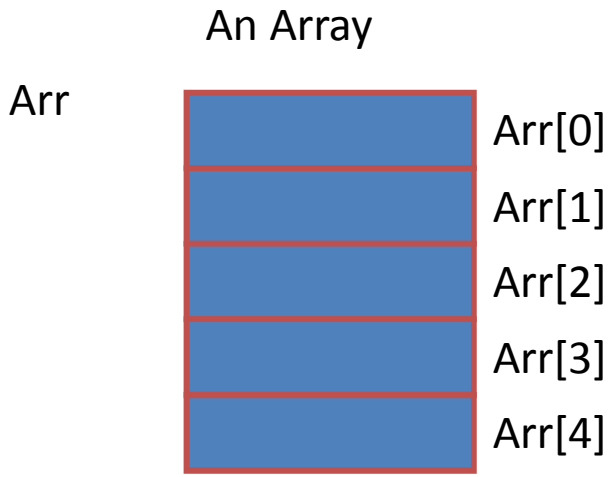
- Simple variables simples contain only one value.
- A computer array is made of several positions, each capable of containing a value.
- An array is essentielly a collection of variables of the same type.



Computer arrays (suite)

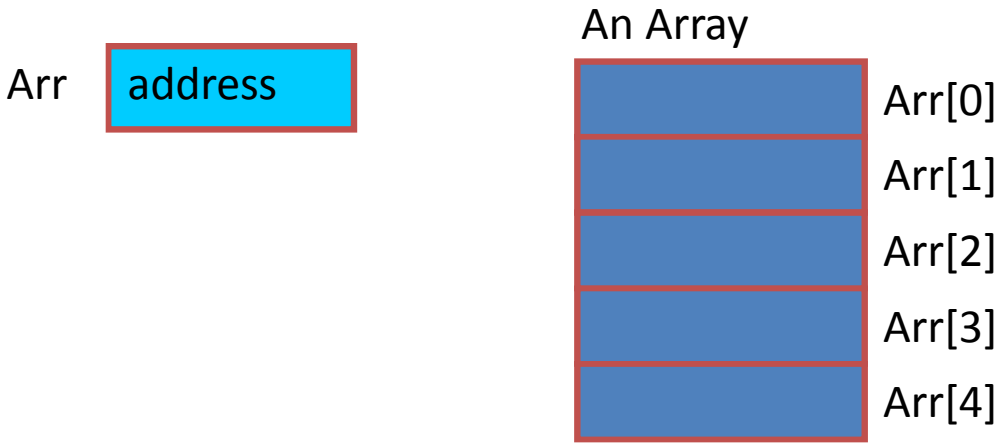
If an array arr has 5 positions, we can access them using indexes.

- Ex: Arr[2] est la troisième position avec index 2.
- Note that Arr[2] is equivalent to a variable name and can be used anywhere a variable name is.



The Array name

- What is in an Array name?
- It can represent a memory address where the Array is located.
  - Similar to a variable name,\ but treated differently.
  - Similar use in C, C++, Java, Python
  - Previous illustration reflect this representation.
- An Array name can also bea reference variable name – and thus our illustration becomes:



Sub-theme: Lists in Python

- In Python, arrays are called lists.
- A list can be described as a collection of elements separed by commas, regrouped between brackets.
- Elements need not to be of the same type.
- Examples:

```
>>> jours = ['lundi', 'mardi', 'mercredi', 'jeudi']

>>> a = [1, 2,3,4,5,6,7,8,9,10]

>>> b = ['abc', 1, 'cde', 10.65, -3]
```

>>> len(jours)

Sub-theme: Accessing element of a list

- Lists are *sequences of ordered collection objects*.
- Each element can be accessed through its list *index*.
- He index number start at zero (not one), the last element is at index len-1

```
>>> jours[0]
'lundi'
>>> jours[2]
'mercredi‘
>>> jours[len(jours)-1]
‘jeudi'
```

Sub-theme: How to modify a list

```
>>> jours = ['lundi', 'mardi', 'mercredi', 'jeudi']

>>> jours.append('vendredi') # added at the end

>>> print(jours)

['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi']

>>> jours.append('samedi') # added at the end

>>> print(jours)

['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi',
'samedi']

>>> del(jours[3])

>>> print(jours)

['lundi', 'mardi', 'mercredi', 'vendredi', 'samedi']
```

Sub-theme: Loops in lists

- Lists are being searched using while or for loops.
- If a result comes early during the search, the loop is stopped otherwiseit will loop until the last element.

Exercises on loops (I)

1 Find the sum of a list values.

2 Let v be a value and a list. Check the sum values does not exceed V.

a) Use the algorithm of exercise 1.

b) improve the version by getting out once the sum exceeds v .

3 Count how many times K shows up in the list or not.

4 Given a list of values and a number K, check if K is in the list or not.

a) Use the algorithm of example 3.

b) Improve it by exiting the loop as soon as K is found.

Exercises on loops (II)

5 Given a list of values and a number K, find the position of the first K occurrence. (if K is not in the list, returns –1 as the position.)

6 Find the maximum value in a list.

7 Find the position of the first occurrence of the maximum value in a list.

a) Use algorithm of example 6.

b) Use algorithm of any examples.

c) Version with a loop without any other algorithm.

8 Check if a list of values contain or not duplicats.

- Strategy?

Exercise 1: *Solution*
Sum of a list values

DATA: L *(ist of numbers)*

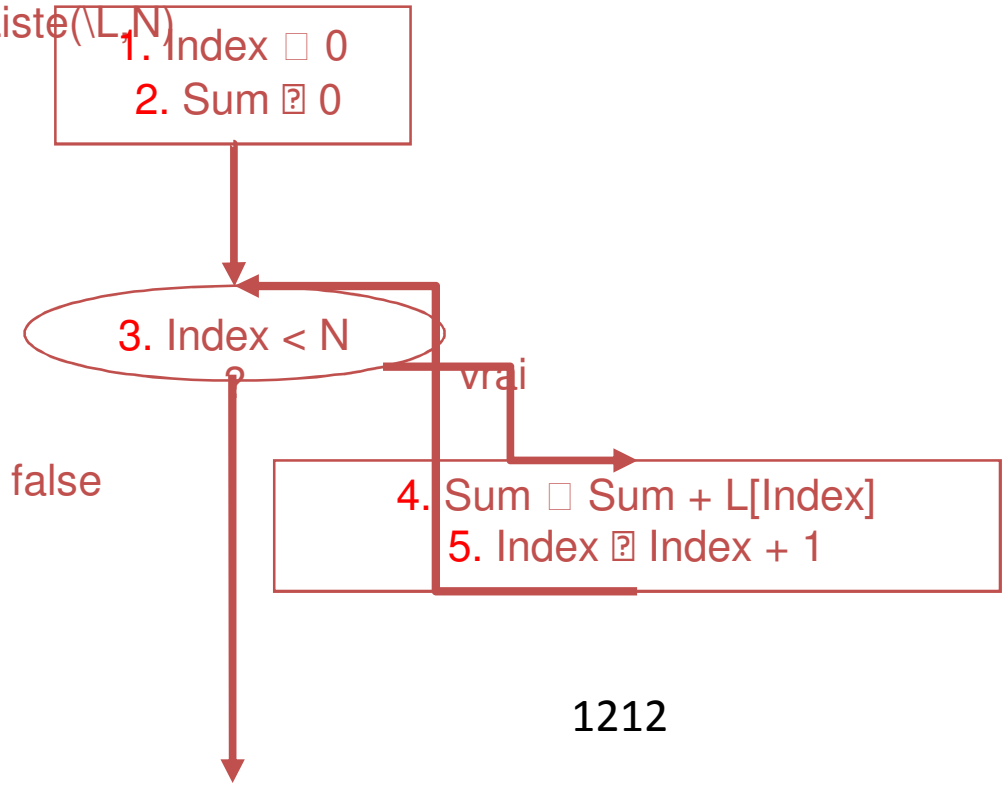
N *(number of list element L)*

INTERMEDIARY: Index *(0 to N-1)*

RESULT: Sum *(sum of list values)*

HEADER: Sum □ SumListe(\L,N)

MODULE:



Exercise 1: Trace de SumListe({2,8,5},3)

Instructions	L	N	Index	Somme
Init.	{2,8,5}	3	?	?
1. Index □ 0			0	
2. Sum □ 0				0
3. Index < N ? True				
4. Sum □ Sum + L[Index]				2
5. Index □ Index + 1			1	
3. Index < N ? True				
4. Sum □ Sum + L[Index]				10
5. Index □ Index + 1			2	
3. Index < N ? True				
4. Sum □ Sum + L[Index				15

Exercise 1: A variance

DATA: L (list of numbers))

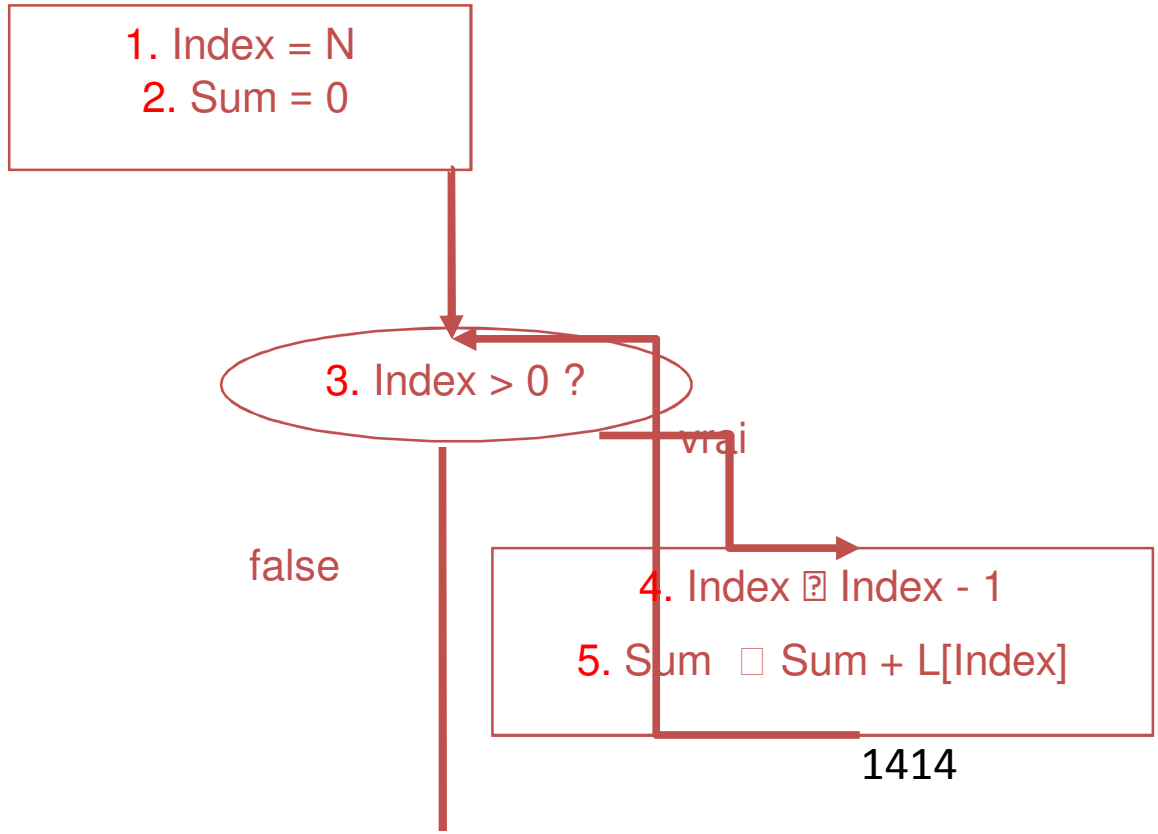
N *(number of a list elements)*

INTERMEDIAIRY: Index *(N-1 to 0)*

RESULT: Sum *(sum of list values)*

HEADER: Sum □ SumListeL,N)

MODULE:



Attention:
Index used in L
Must not be < 0
or supérior or équal to N

Solution: Exercise 1 in Python

```
def sumList(l):
 sum = 0
 index = 0
 while index < len(l):
   sum = sum + l[index]
   index = index + 1
 return sum
a = [10, 28, -5, 6, 31, 25, -7, 20]
```



```
print(sommeListe(a))
```

Exercise 2 - *Solution*

a): Sum of a list values exceeds V?

DATA: X *(list of numbers)*

N (number of a list ele*ments*)

V *(A limit value)*

INTERMEDIAIRY: Sum *(Sum of values)*

RESULT: Exceed *(Boolean: True if Sum> V and false else)*

HEADER: Exceed □ SumExceedV(X,N,V)

MODULE: Sum □ SumList(X,N)

Exceed □ (Sum > V)

Exercice 2 - *Solution*
b): SumExeedV, more efficient version

DATA: X *(list of numbers)*

N (number of a list elements)

V *(a limit value)*

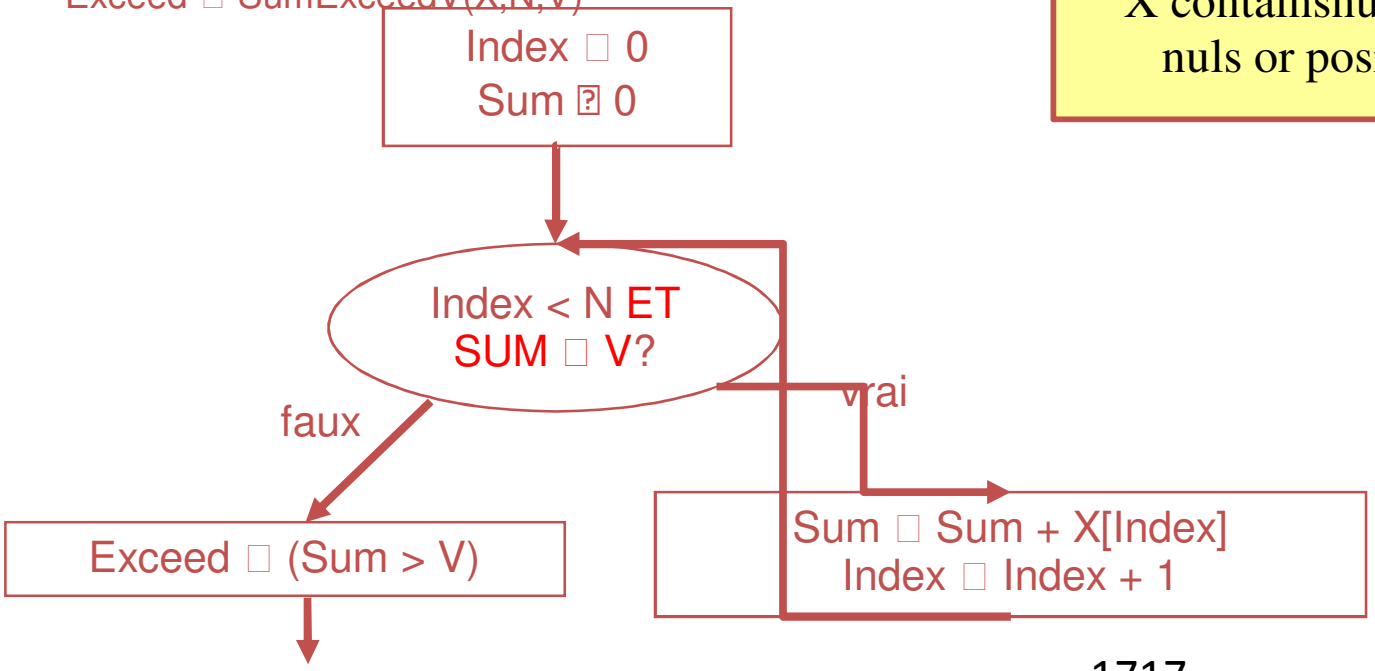
INTERMEDIAIRY: Index (*0 to N-1*)

Sum (Sum *of a list values*)

RESULT: Exceed *(Boolean: True if Sum > V and false else)*

HEADER: Exceed □ SumExceedV(X,N,V)

MODULE:



Hypothèsis:
X containsnumbers
nuls or positive

Solution: Exercise 2a in Python

```
def exceed(x, v):
 "Returns True if the sum of the list elements exceede v"
 return sumList(x) > v

a = [10, 28, -5, 6, 31, 25, -7]
print(exceed(a, 100))
```

Note: Compute more than necessary.

Solution: Exercise 2b in Python

```
def exceed(x, v):
 "Returns True if the sum of the list elements exceed v"
 sum = 0
 index = 0
 while (index < len(x)) and (sum <= v):
   sum = sum + x[index]
   index = index + 1
  return sum > v
a = [10, 28, -5, 6, 31, 25, -7]
print(exceed(a, 100))
```

Essayer avec boucle for aussi.

DATA: X (*list of numbers*)

N (*numberof the list elements*)

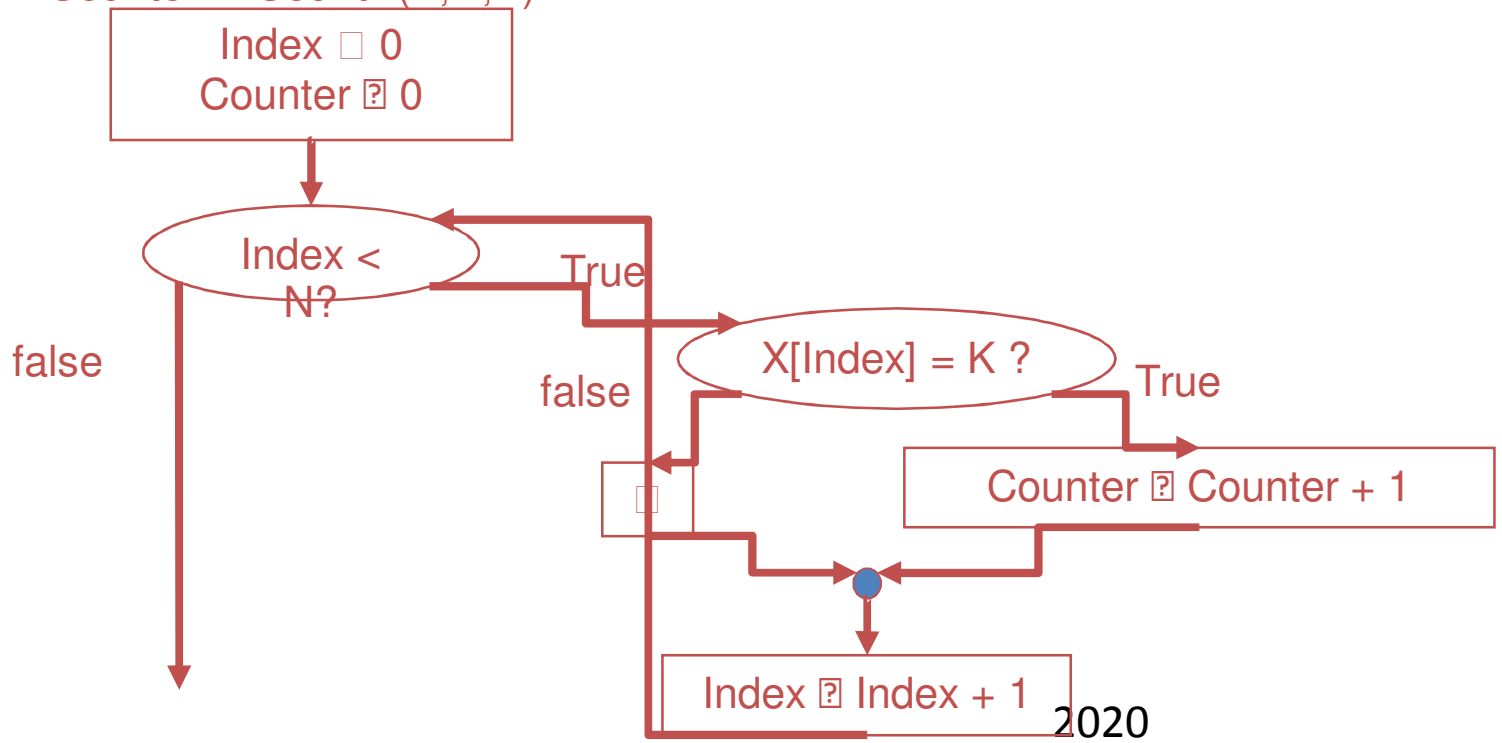
K *(value whose instances are counted)*

INTERMEDIARY: Index (*0 to N-1*)

RESULT: Counter *(number of instances of K in X)*

HEADER: Counter □ CountK(X,N,K)

MODULE:



Solution: Exercise 3 in Python

```
def countK(x, k):
 "count the number of k in the list"
 count = 0
 for val in x:
   if val == k:
     count = count + 1
 return count
a = [10, 28, -5, 6, 31, 25, 10, -7, 10]
print(countK(a, 10))
```

DATA: X (*list of numbers*)

N (*number of a list elements*)

K *(target value)*

INTERMEDIAIRY: Counter *(number of K instances in X, example 3)*

RESULT: Found *(Boolean: true if K is in X, else false)*

HEADER: Found □ FoundK(X,N,K)

MODULE:

Counter □ CountK(X,N,K)

Found □ (Counter > 0)

DATA: X (*list of numbers*)

N (*number of a list elements*)

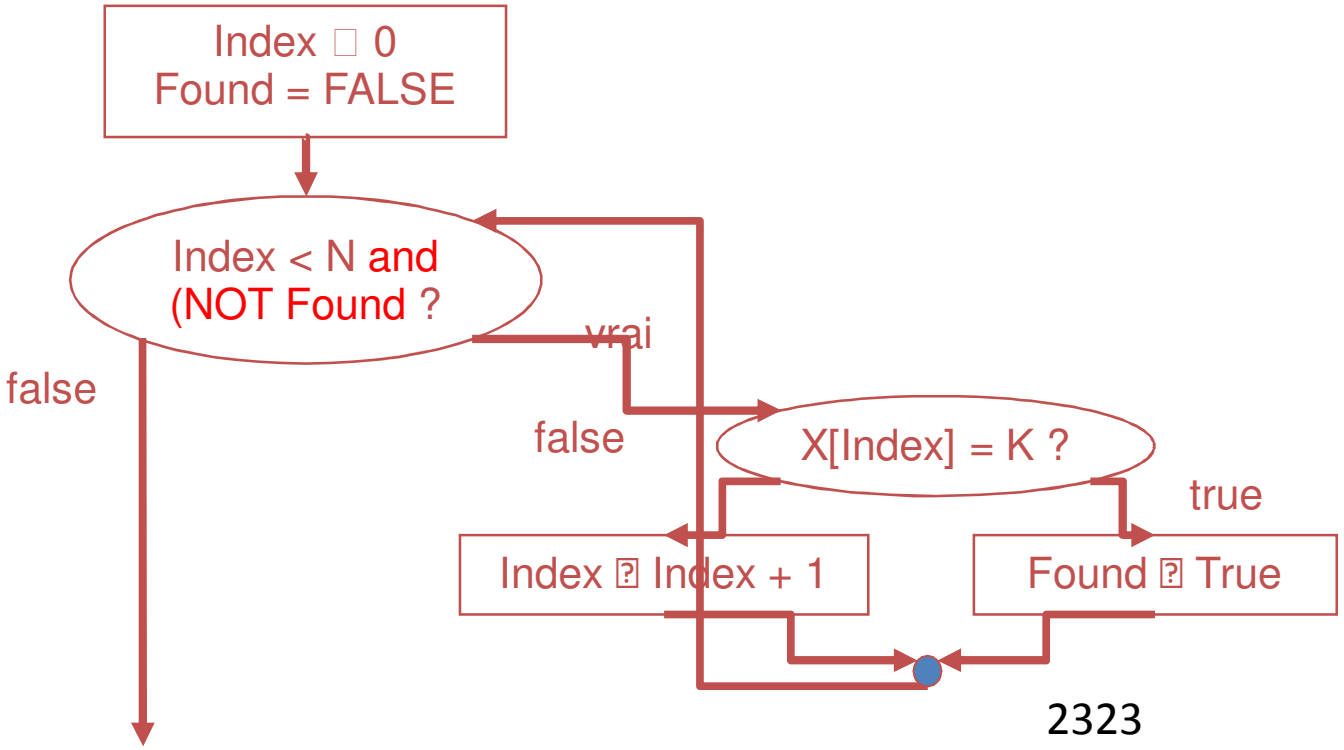
K *(target value)*

INTERMEDIAIRY: Index (*0 to N-1*)

RESULT: Found *(Boolean: tru if K is in X and false else)*

HEADER: Found □FoundK(X,N,K)

MODULE:



Solution: Exercise 4a in Python

```
def found(x, k):
 "Returns True if k is in the list, else it is False"
 found = countK(x,k) > 0
 return found


a = [10, 28, -5, 6, 31, 25, 10, -7, 10]
print(found(a, 10))
print(found, 26))
```

Note: More work than necessarily.

Solution: Exercise 4b in Python

```
def found(x, k):
 "Returns True if k is in the list, False else"
 found = False
 for val in x:
   if val == k:
     found = True
     break
 return found

a = [10, 28, -5, 6, 31, 25, 10, -7, 10]

print(found(a, 10))
print(found(a, 26))
```

DATA: X (*list of numbers*)

N (*number of a list elements*)

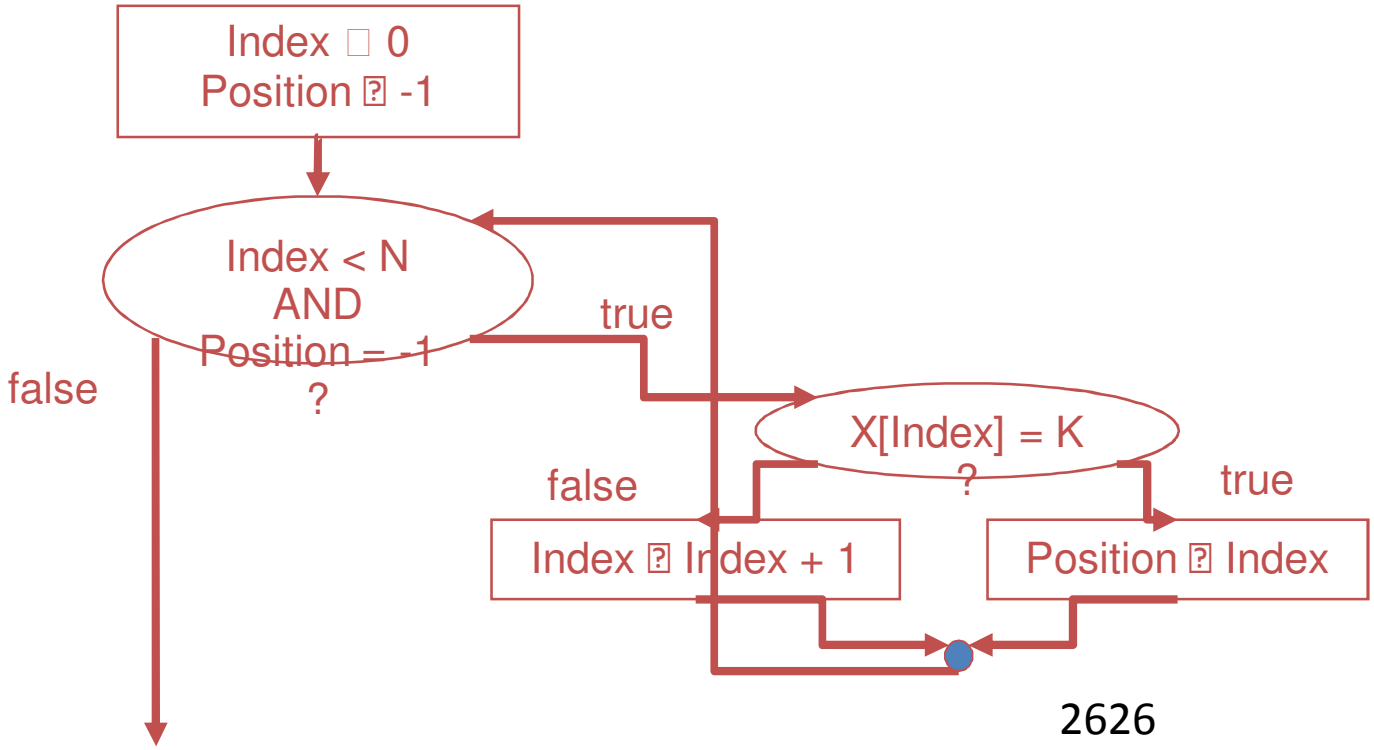
K *(target value)*

INTERMEDIARY: Index (*0 to N-1*)

RESULT: Position *(position of K in X, or -1 if K is not in the list)*

HEADER: Position □ WhereIsK(X,N,K)

MODULE:



Solution: Exercise 5 in Python

```
def whereIsK(x, k):
 "Returns the position of k in the list x, eand -1 if k
is not found"
 position = -1
 index = 0
 while index < len(x) and (position == -1):
   if x[index] == k:
     position = index
      # break
   index = index + 1
 return position
```


x = [10, 28, 5, 6, 31, 25, 10, 7, 10]

Exercise 6: Find the maximum element in the list

- Problem:
  - Givent a list of numbers, how to find the maximum value from the list?

Idea:

  - Strategy « explore ad update ». Start by using the first element as a candidate to the max. value. Then, check the other elements of the list to possibly update the max. when a bigger number is met.

- We use a loop to check the list elements.

Exercise 6: Find the maximum element in the list

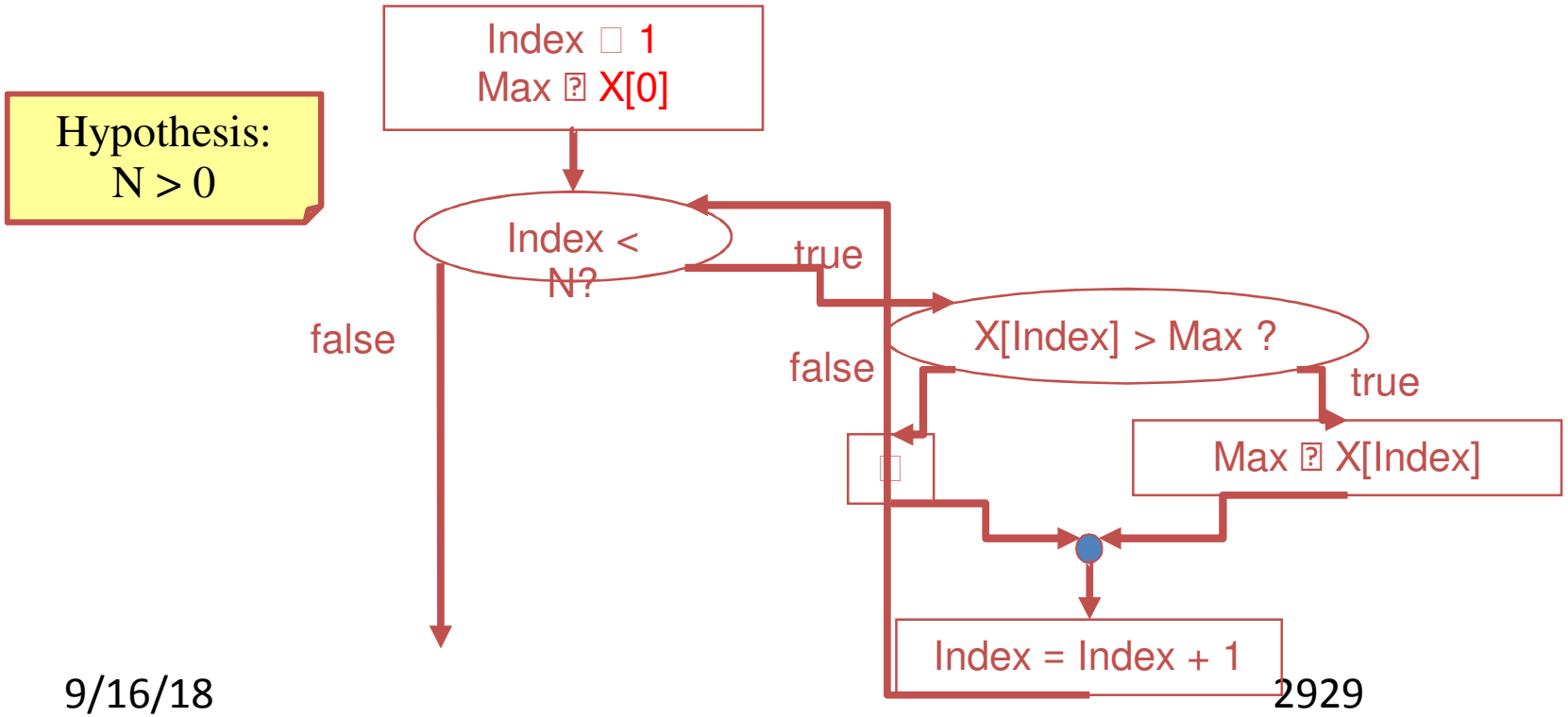
(point to a list of numbers)

er of a list elements)

RY: Index (*0 to N-1*)

Max *(maximum value in the list)*

Max MaxInLlist(T,N)



Solution: Exercise 6 in Python

```
def maxInList(l):
 "Retuns the maximum value in a list"
   max = l[0]   # - float('Inf')
   for val in l:
      if val > max:
         max = val
 return max

a = [10, 28, -5, 6, 31, 25, -7, 20]

print(maxInList(a))
```

Exercice 7 a): Find the position of the list max. value

DATA: X (*list of numbers*)

N (*number of a list elements*)

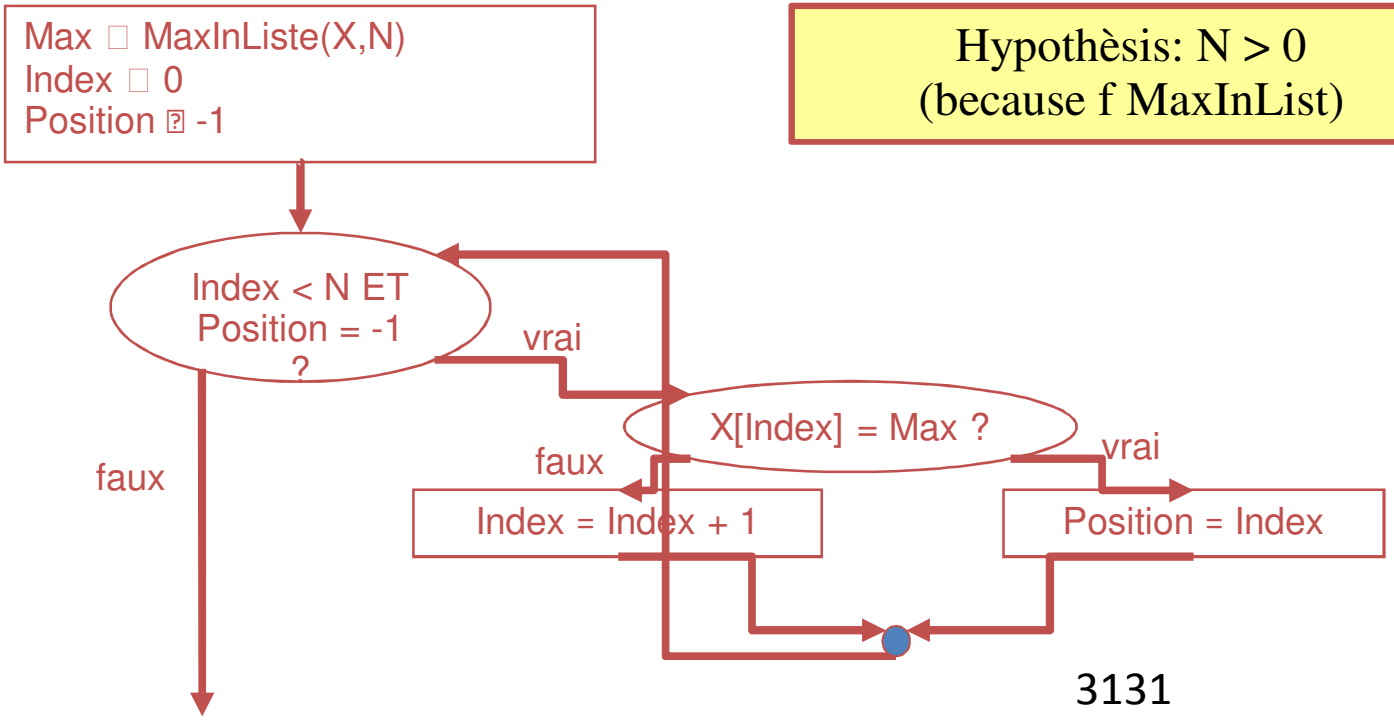
INTERMEDIAIRY: Index (*0 to N-1*)

Max (*max. value in X)*

RESULT: Position *(position of the first max in X)*

-HEADER: Position = MaxPosInList(X,N)

MODULE:



Exercice 7 b): Find the position of the list max. value

DATA: X (*list of numbers*)

N (*number of a list elements*)

INTERMEDIAIRY: Index (*0 to N-1*)

Max (*max. value in X*)

Hypothesis: N > 0
(because of MaxInList)

RESULT: Position (*position of the first max in X*)

-HEADER: Position = MaxPosInList(X,N)

MODULE:

Max ← MaxInList(X,N)

Position ← whereIsK(X,N,Max)

Exercice 7 c): Trouver la position de la valeur maximale d'une liste

DATA: X (*list of numbers*

N (*number of the list elements)*

INTERMEDIARY: Index (*0 to N-1*)

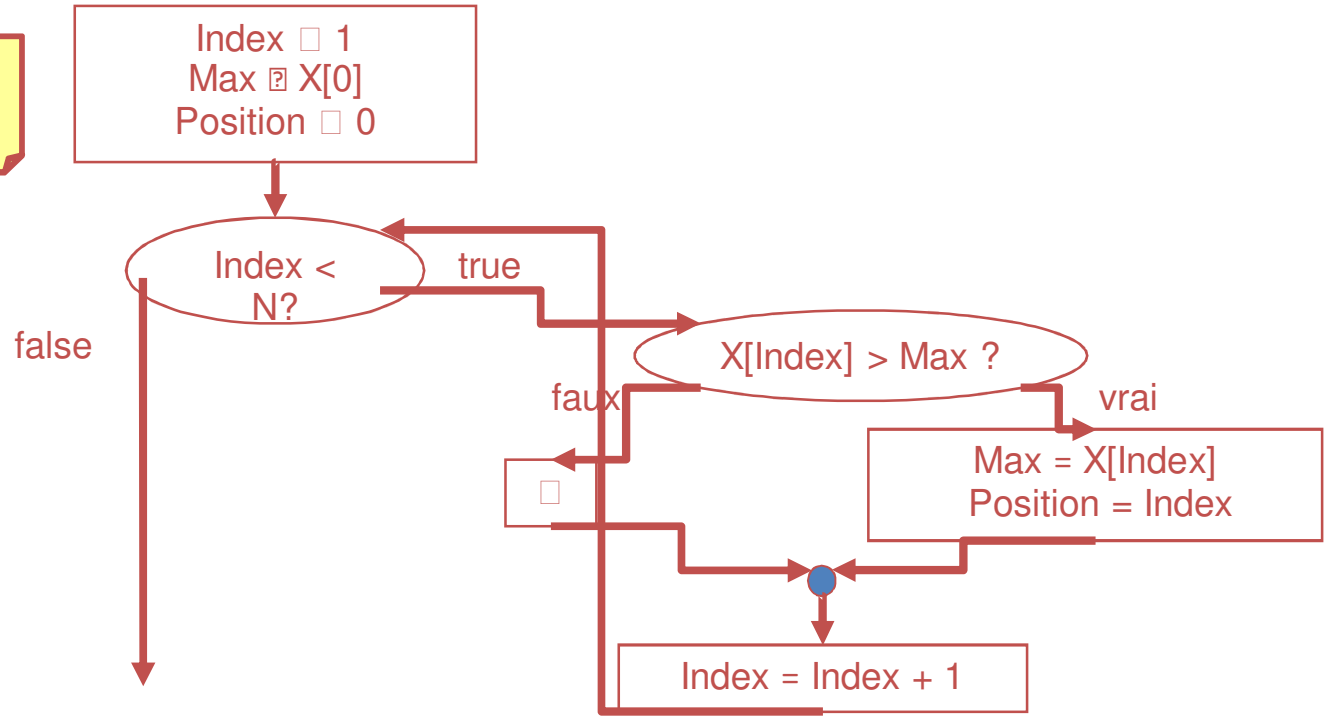
Max *(max. value in X)*

RESULTA Position *(position of first max. value in X)*

HEADER: Position □ MaxPosInList(X,N)

MODULE:

Hypothesis:
N > 0



Solution: Exercise 7a in Python

```
def maxPosInList(x):
 "Returns the position of the max. element in list"
 index = 0
 position = -1
 max = maxInList(x)
 while index < len(x) and position == -1:
  if x[index] == max:
    position = index
  index = index + 1
 return position
```


a = [10, 28, -5, 6, 31, 25, -7, 20]

Solution: Exercise 7b in Python

```
def maxPosInList(x):
 "Returns the position of the max. element in list"
 max = maxInList(x)
 position = whereIsK(x, max)
return position


a = [10, 28, -5, 6, 31, 25, -7, 20]
print(maxPosInList(a))
```

Solution: Exercise 7c in Python

```
def maxPosInList(x):
 "Returns the position of the max. element in list"
 index = 0
 position = -1
 max = x[0]
 while index < len(x):
  if x[index] > max:
    max = x[index]
    position = index
  index = index + 1
 return position
```


a = [10, 28, -5, 6, 31, 25, -7, 20]

Exercise 8: Are there duplicates in the list?

DATA: X (*liste of numbers*)

N (*number of elements in the*)

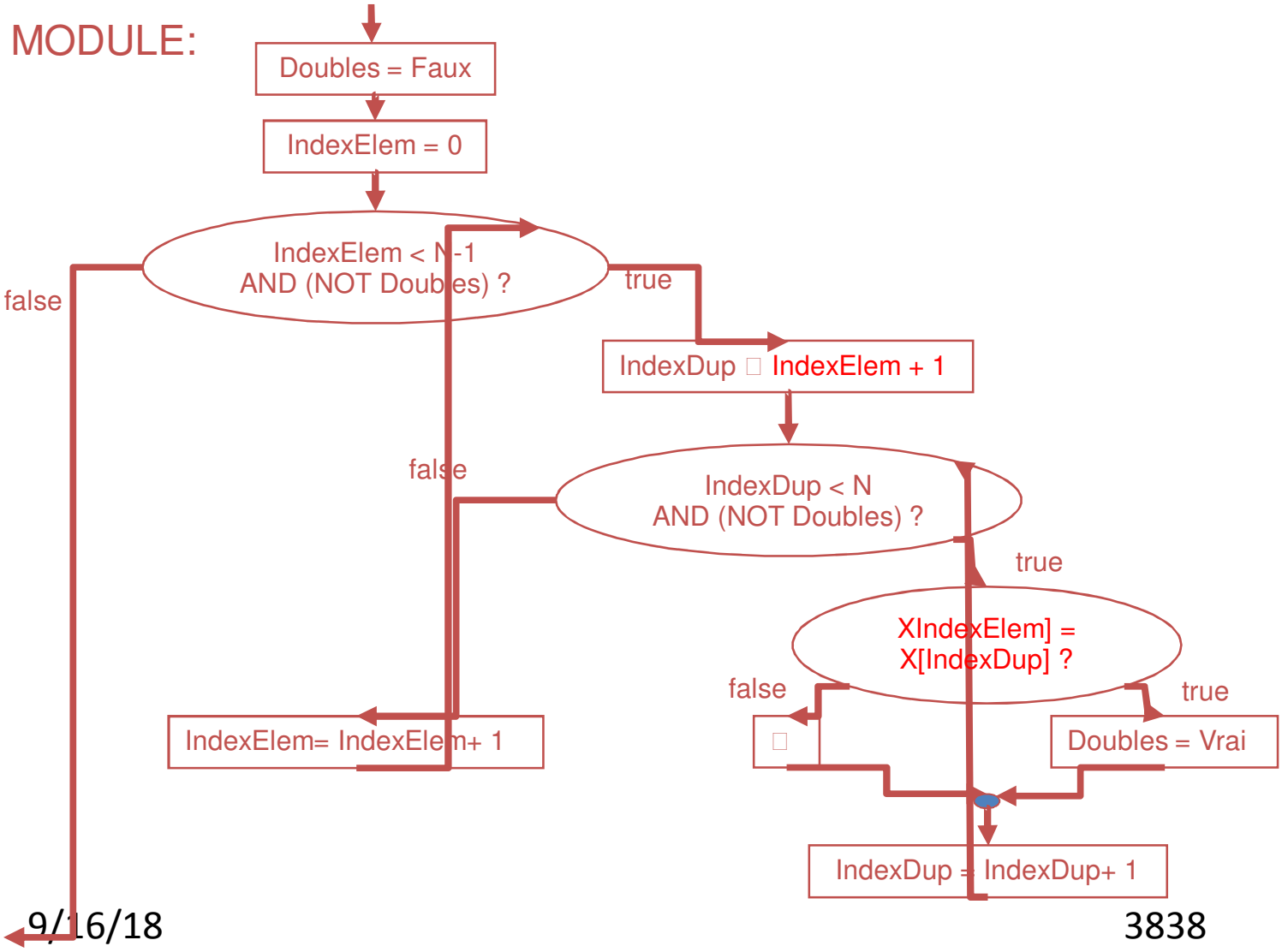
INTERMÉDIARY: IndexElem *(index of courant element)*
IndexDup *(index to look for a duplicate to the courant element)*

RESULT: Doubles *(Boolean: true if list has duplicates, otherwise false)*

HEADER: Doubles HasDoubles(T,N)

MODULE:

Exercise 8:
Are there duplicates in the list?



Solution: Exercise 8 en Python

```
def hasDoubles(x):
 "Returns True if there doubles in the list"
 doubles = False
 indexElem = 0
 while indexElem < len(x) - 1 and not doubles:
  indexDup = indexElem + 1
  while indexDup < len(x) and not doubles:
    if x[indexElem] == x[indexDup]:
       doubles = True
    indexDup = indexDup + 1
```

Question:

What will the following Python code display?

```
def myFunctionList(l):
 index = 0
 m =  l[0]
 while index < len(l):
  if l[index] < m:
    m = l[index]
  index = index + 1
 return m
```

Theme 2. *Character chain and loops*

Sub-theme: Character chains

- Variables whose type is a character chain have always been a challenge in programming languages.
  - They have variable sizes thus the computer would require more informative and predictable data about the amount of memory space necessary to store them.
- As a consequence character chains are often considered as «special cases»

Character chains in Python

- A data of type *string* can be defined as sequence of characters.
- We can group them using simple quotes or double quotes.

Examples :

```
>>> state1= ‘the eggs are hard.'
>>> state2= '“yes", he answered,'
>>> state3 = “I like it"
>>> print(state2, state3, state1)
“Yes", he answered, I like it the eggs are hard.
```

Sub-theme: Accessing characters

- A character chain is a composed data, a sequence, a collection of ordered elements.
- Each character is accessible through its index.

```
Example :

>>> ch = "Christine"

>>> print(ch[0], ch[3], ch[5])

C i t

>>> print (len(ch))

9
```

Exercise: Comparison of characters

- What is the comparison outcome of those examples?
  - Example 1:

```
str1 = "abcde"
str2 = "abcfg"
```

str1 < str2 ?

  - Example 2:

```
str1 = "abcde"
str2 = "ab"
```

str1 < str2 ?

Reponse: True, False

Sub-theme: loops in character chains

- We can use while loops to visit character chains, as it was done to visit lists.
- Print character chain elements one per line:

```
ch = “hello"

index = 0

while (index < len(ch)):

    print (ch[index])

    index = index + 1
```

Exercise: Double the characters in a list

- Develop a function that takes a character chain and returns another chain the double of each.
- Test-la.
- For example, if the chain is ‘hello', the result is: ‘hheellllloo'

Essayez vous-même en Python/IDLE.
Voir la solution à la page suivante.

Exercise - *Solution:*
Double the characters in a list

```
def double(x):
 "Returns a character chain with each element doubled"
 result = ""
 index = 0
 while (index < len(ch)):
    result = result  + ch[index] + ch[index]
    index = index + 1
 return(result)
ch = “hello"
print(double(ch))
```

Question

What will the following Python code display?

```
ch = "bonjour"
index = 0
n = len(ch)-1
while (index < len(ch)):
    print (ch[n-index], end="")
    index = index + 1
```

Conclusion

- The concept of lists and character chains allow us to use composed data.
- We can visit then using loops to solve complex problems.