ITI 1120
Module 2: Branches

General Concepts:
1. Software models with branches.
2. Branch instructions in Python.
3. Boolean expressions.

General Objective: You will be able to use branch instruction in developing codes in Python.

Learning Objectives:
1. Identify and derive software model diagrams that describe visually algorithms.
2. Find an algorithm and derive a Python program for each given problem.
3. Develop an Boolean expression and evaluate complex Boolean expressions.

Theme 1. Branches

Sub-theme: The branch instruction

- Up to now, our algorithm modules have contained:
  - a simple instruction
  - a sequence of simple instructions
- We often need more complex structures in our solutions, when conditions need to be tested to move forward.

Sub-theme:

Problem: Largest number

- Derive an algorithm that return the largest value between two given data.

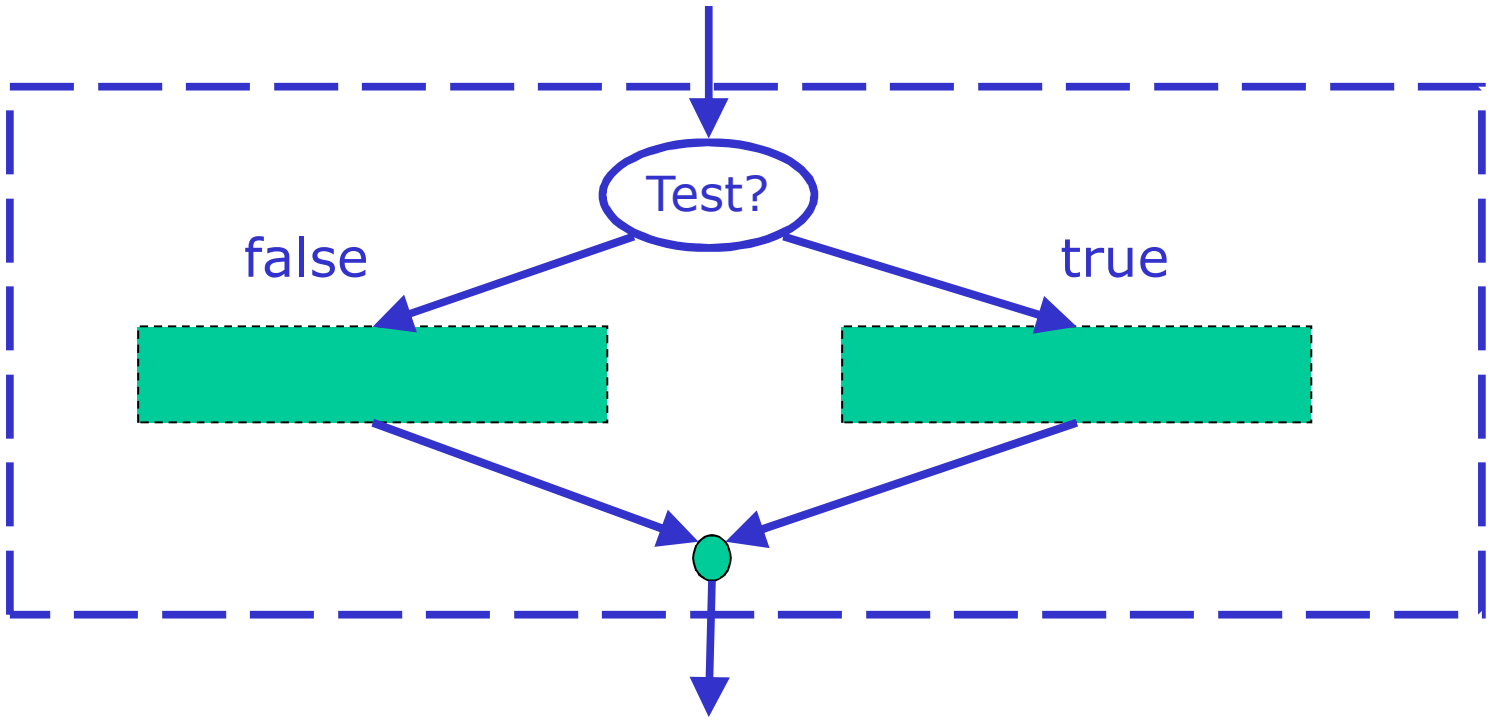
DATAS: X, Y *(2 numbers)*

RÉSULT: M *(maximum between X and Y)*

HEADER: M = Max2(X, Y)

- M = X or M = Y ?
- A branch instruction is required for the solution.

Sous-thème: ranch instruction model



Dashed boxes are pointillées INSTRUCTIONS BLOCS .

Note that the branch instrction is complex et contain several other instructions in the d'instruction blocs

It is a graphical representationof software models!

Explanations Software model diagrams

- A visual algorithms descriptions.
- Composed of nodes connected by arrows.
- Test node:
  - verify the condition (Boolean expression with interrogation point).
- Instructions bloc node:
  - Indicate where another instructions bloc can be introduced. This bloc can contain simple or complex instructions (even no instruction)

Explanations

Instructions bloc content

- Simple instruction (invocation, assignment)
- Empty instruction (Ø = “nothing to do”)
- Branch instruction
- Repetition /loop instruction (to come…)

Important: Each bloc has exactly one input (incoming arrow) and an output (outgoing arrow).

Exercise 1 – Back to the problem of the largest of the 2 numbers

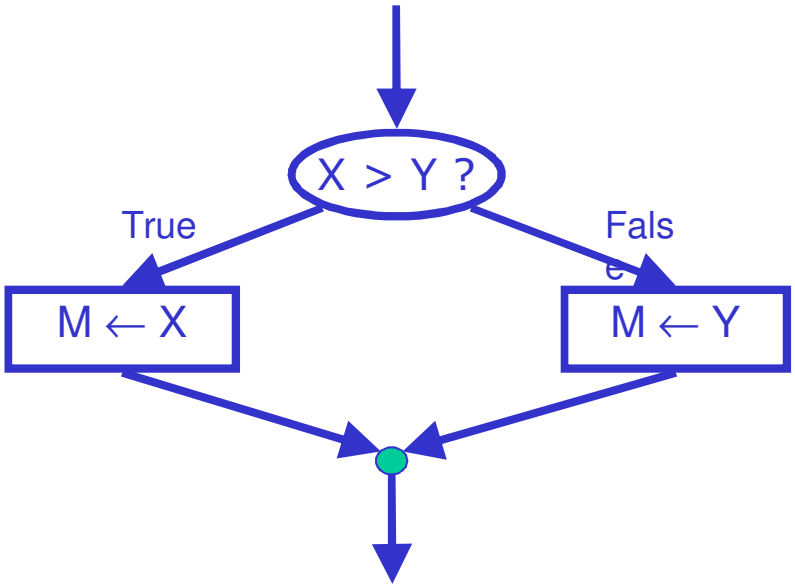
- Derive an algorithm that picks up the maximum (M) between 2 values (X and Y) in our software model?

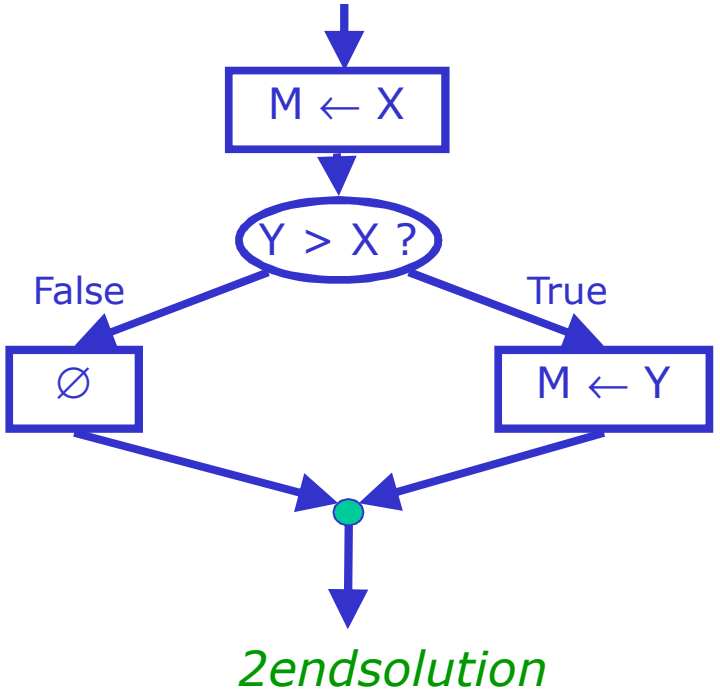
See the solution in the following page

Exercise 1 – *Solution*: Largest of 2 numbers

How to derive an algorithm to find the maximum (M) between 2 numbers (X and Y) in our software model?

DATA: X, Y *(2 numbers)*
RESULT: M *(maximum of X and Y)*
HEADER: M ← Max2(X, Y)
MODULE:





2endsolution

Exercise 2: Largest of 3 numbers

- Find the maximum of 3 given numbers X, Y, and Z.

DATAS X, Y, Z *(3 numbers)*

RESULT: M *(maximum of X , Y and Z)*

HEADER: M = Max3(X, Y, Z)

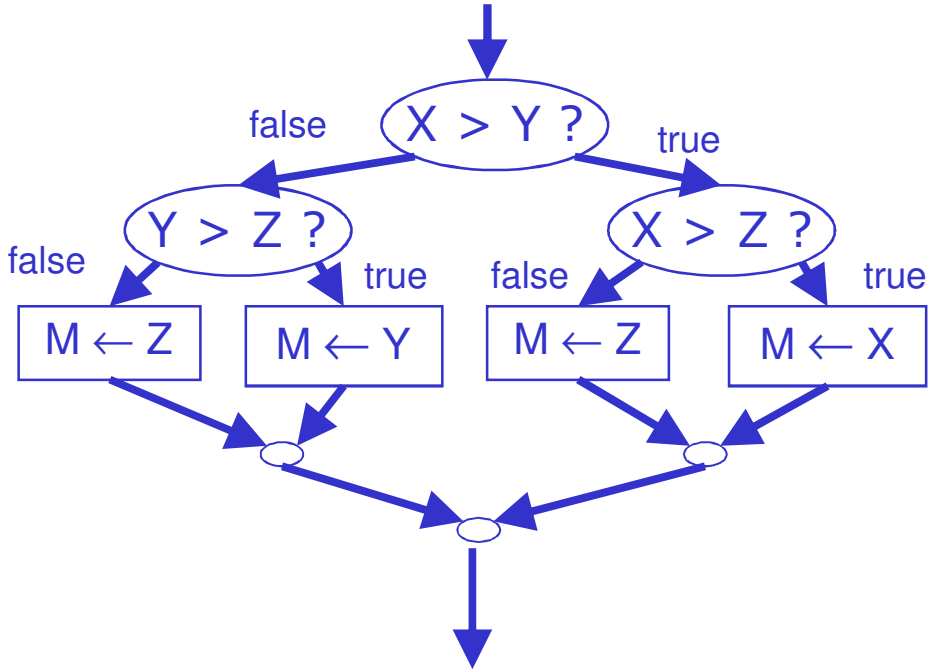
  - Version 1: with interwoven tests
  - Version 2: sequence of tests

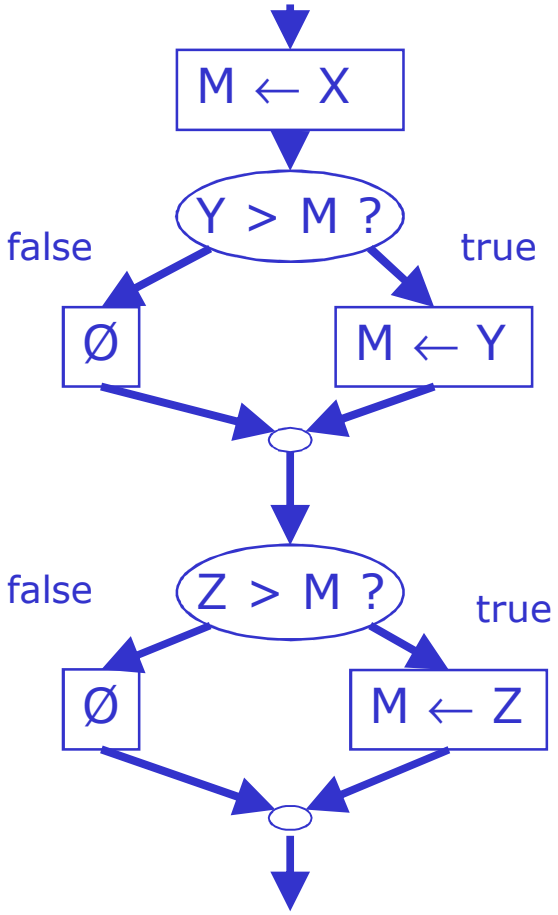
Exercice 2 – *Solution*: Largest of 3 numbers

Find the maximum of 3 given numbers X, Y, et Z.

Version 1: with interwoven tests

Version 2: with a sequence of tests





Sub-theme: Algorithm tracing with branches

- During algorithm tracing with tests (branches or loops):
  - Number the tests and instructions.
  - Add a line to evaluated conditions and to the evaluated results (*true or false*).
  - Indicate only executed instructions in the tracing.

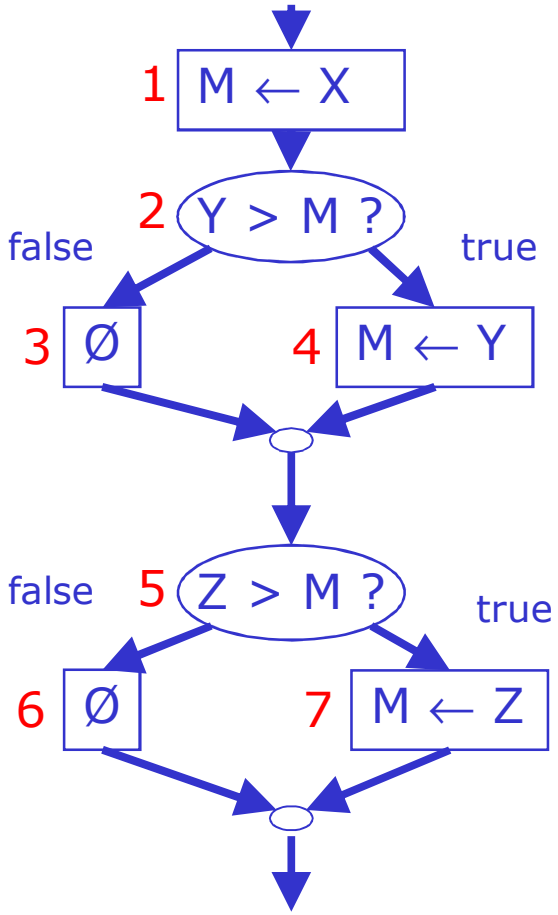
Trace du problème maximum de trois nombres, version 2

- Trace: MAX3(5, 11, 8)

Voir la solution sur la page suivante

Solution: Max3 trace, version 2

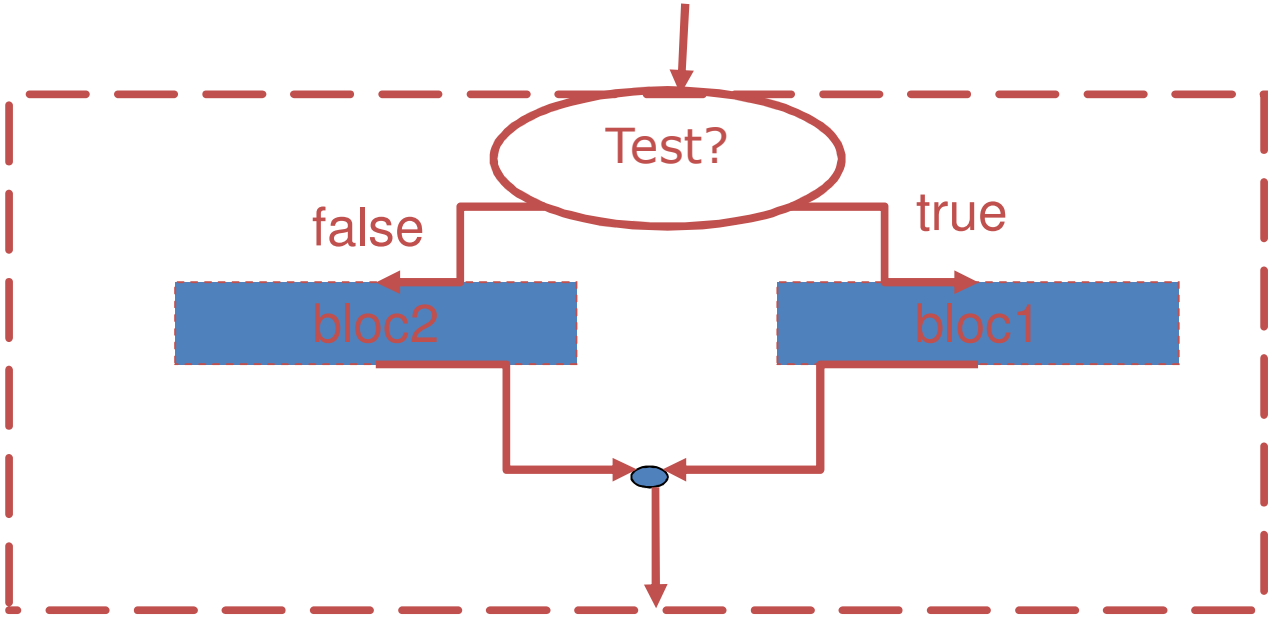
Trace: MAX3(5, 11, 8)



	X	Y	Z	M
Valeurs initiales				

Theme 2: Branch Instructions in Python

Sub-them: Branch translation



Instruction bloc 1

Instruction bloc2

```
If Test:
    # instructions
else:
    # instructions
```

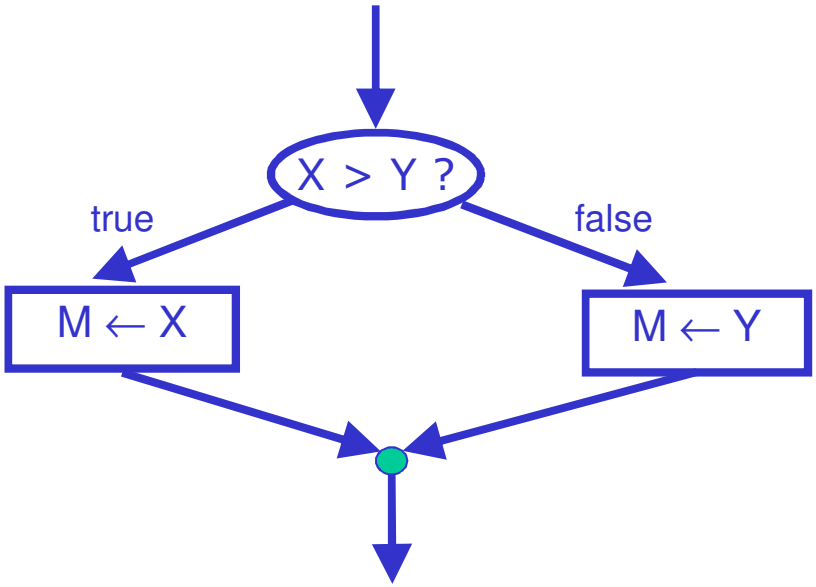
Exercise 3 – Branch Max2 translation

Data: X, Y *(2 numbers)*

RESULT: M (*largest data*)

HEADER: M←Max2(X, Y)

In Python:



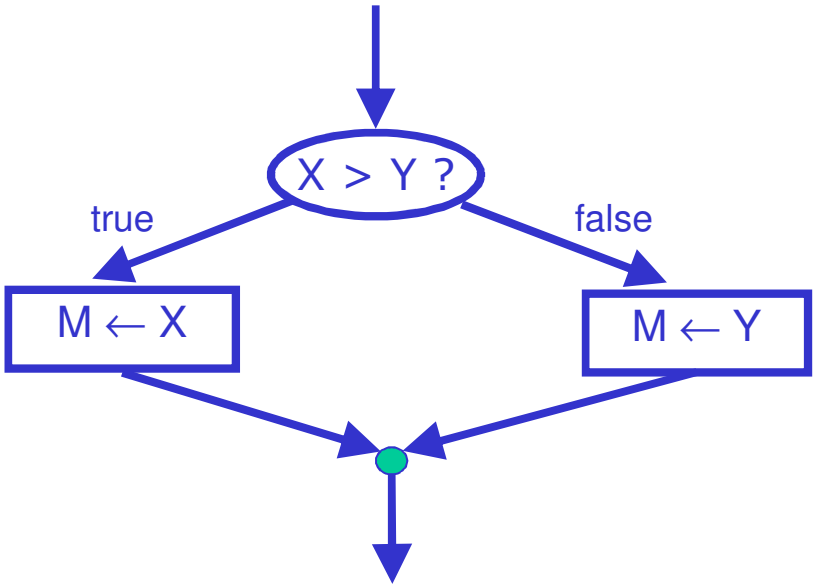
Exercise 3 – *Solution*: Branch Max2 translation

Data: X, Y *(2 numbers)*

Result: M (*largest data*)

Header: M←Max2(X, Y)

In Python:




```
# Solution1
x = 7
y = 10

if x > y :
    m = x
else:
    m = y
print(m)
```

```
# Solution2
def max2(x,y):
 if x > y :
     m = x
 else:
     m = y
 return m

x = 7
y = 10
r = max2(x,y)
print(r)
```

Exercise 4 – Max3 branch translation

Data: X, Y, Z *(3 numbers)*

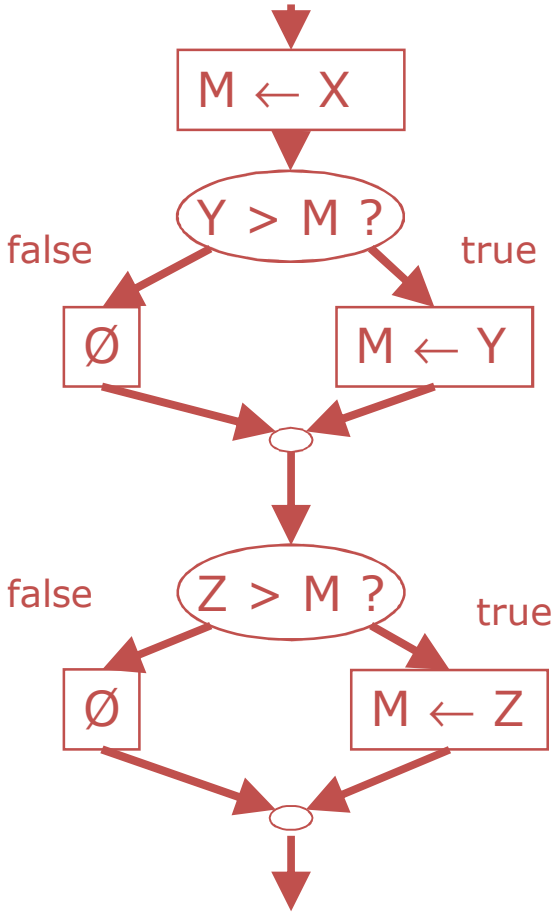
Result: M (*largest data*)

Header: M=Max3(X, Y, Z)

- Two solutions:
  - sequence of branch instructions
  - Interwoven branch instructions
- Python translation:

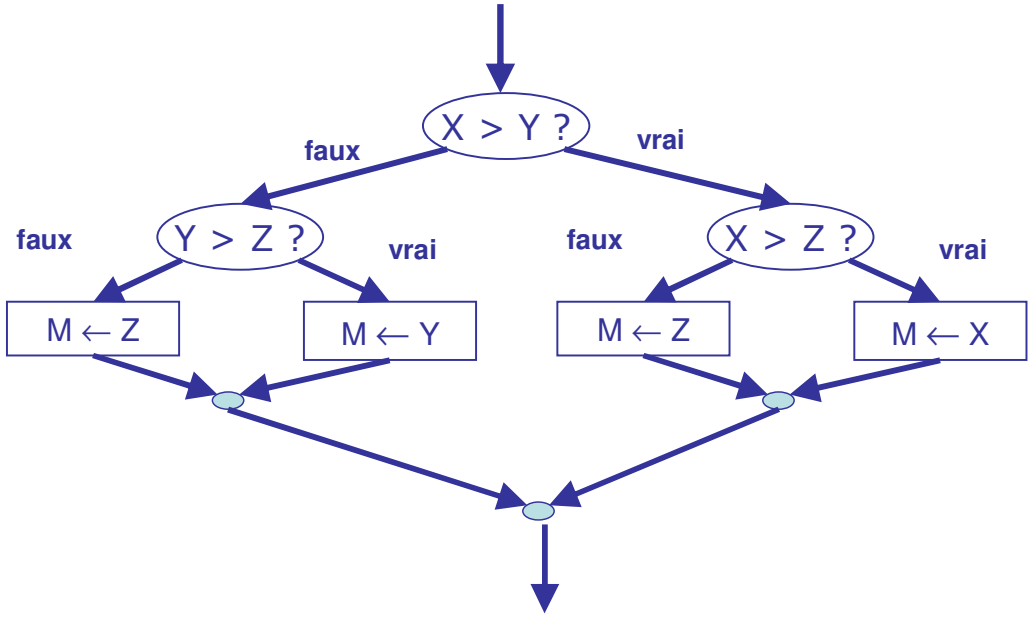
Exercise 4 – **Solution:** Max3 version 1, sequence of branch instructions translation

Data: X, Y, Z *(3 numbers)*
Result: M (*largest data*)
Header: M←Max3(X, Y, Z)



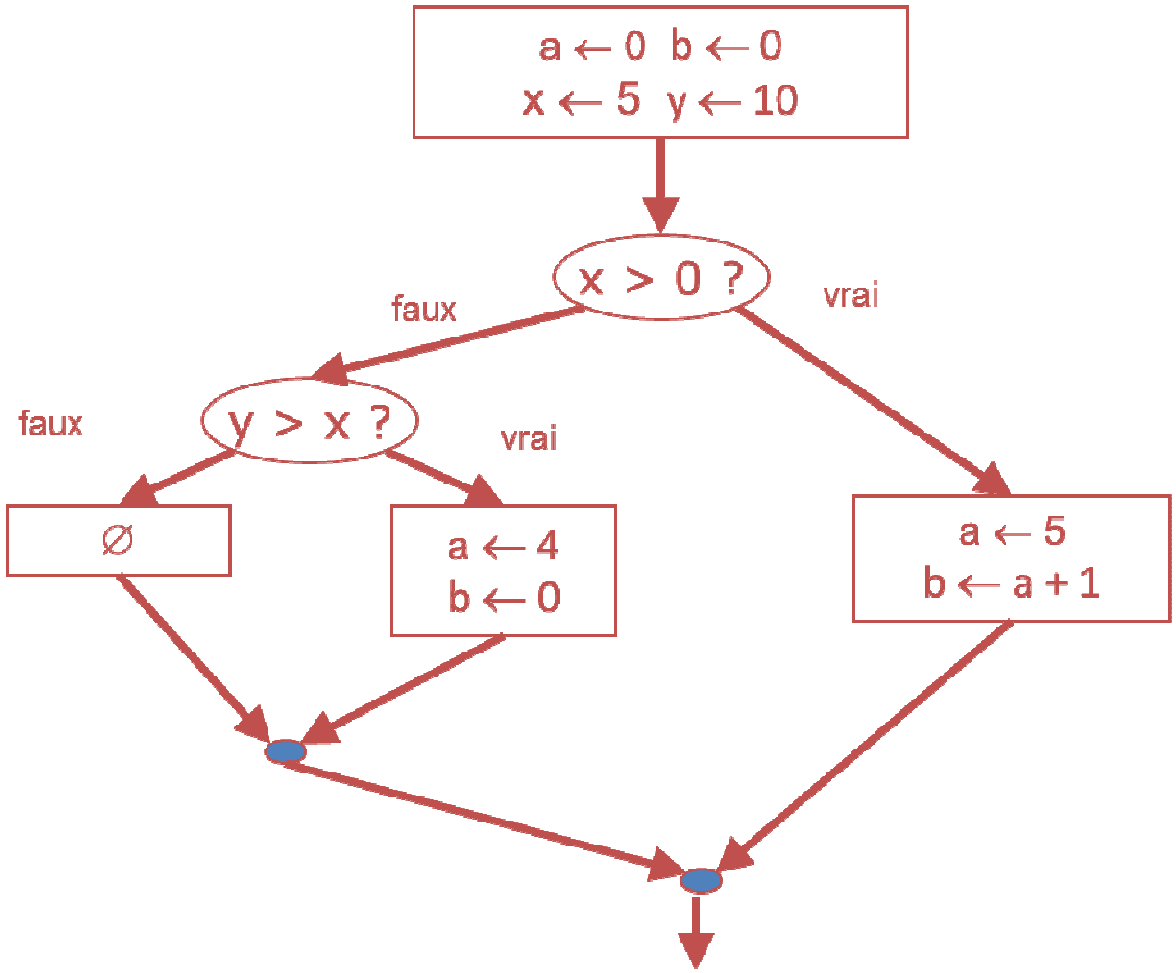

```
def max3(x,y,z):
 m = x
 if y > m :
   m = y
 if z > m:
   m = z
 return m

x = 7
y = 10
z = 5
r = max3(x,y,z)
print r
```

Exercise 4 – **Solution:** Max 3 version 2
Translation of interwoven branches

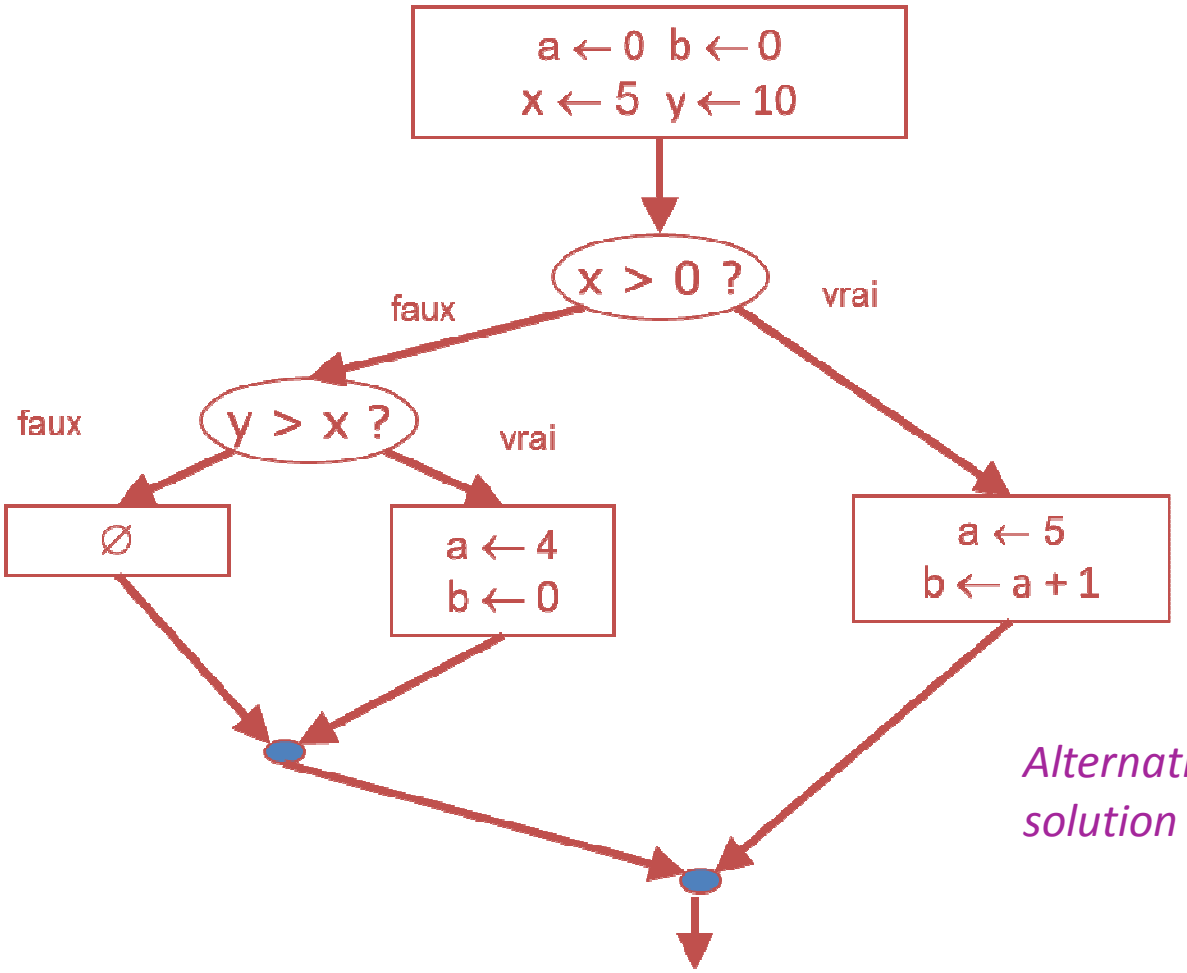



```
def max3(x,y,z):
 if x > y :
  if x > z:
   m = x
  else:
   m = z
 else:
  if y > z:
   m = y
  else:
   m = z
 return m
```

Exercise 5 – Instruction bloc translation



Exercise 5 – Instruction bloc translation



Solution:

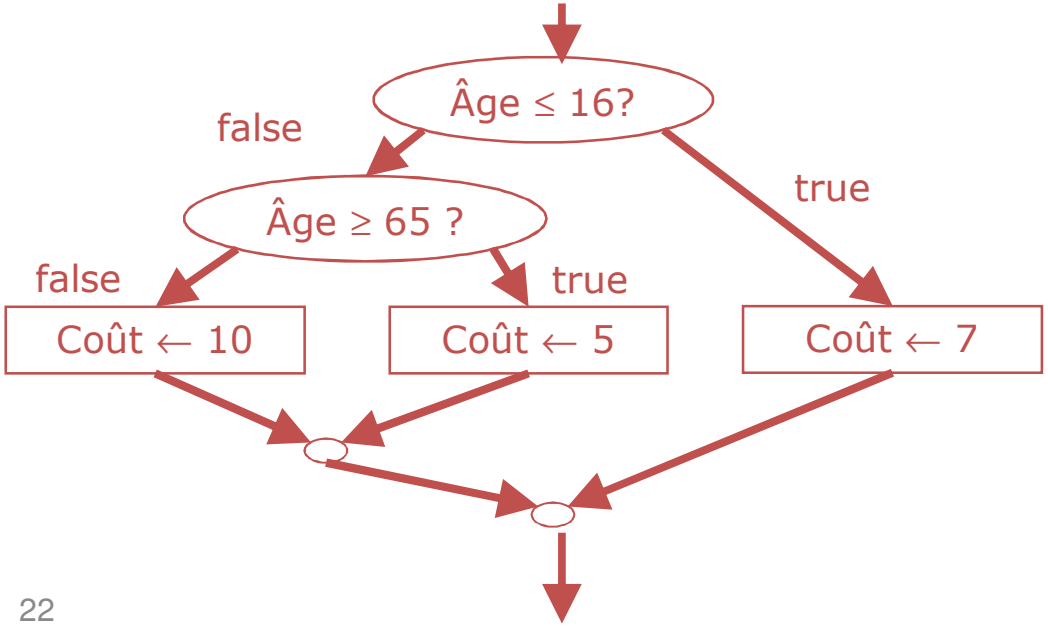
```
a = 0
b = 0
x = 5
y = 10

if x > 0 :
   a = 5
   b = a + 1
else:
   if y > x :
      a = 4
      b = 0
```

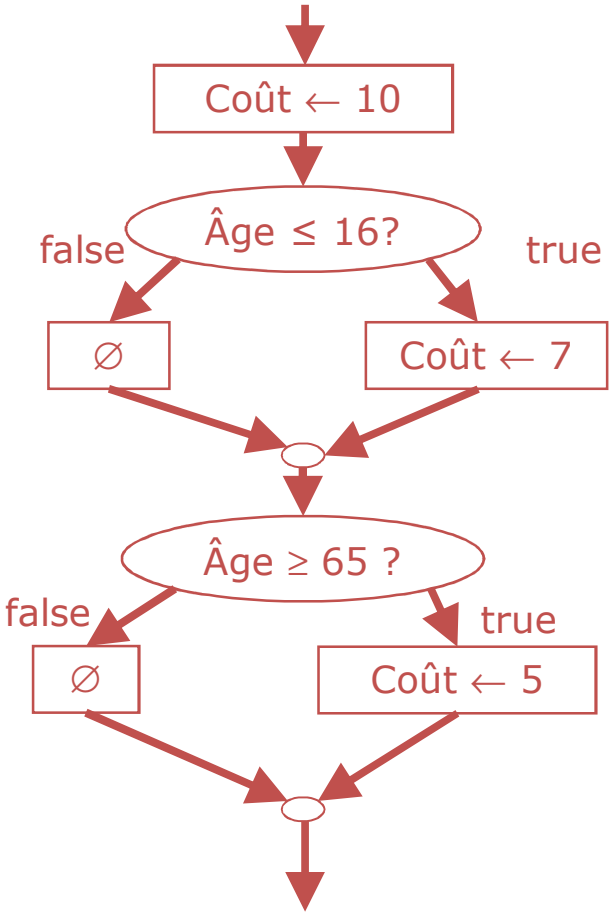
Alternative solution

```
if x > 0 :
   a = 5
   b = a + 1
elif  y > x :
      a = 4
      b = 0
```

Exercise 6 – Movie ticket

- Derive an algorithm that calculate the cost of a ticket when it is 7$ if you are 16 years old or younger, 5$ if you are 65 years or older and 10$ otherwise.
  - Version 1: Interwoven tests
  - Version 2: sequence of tests





Exercise 6 – Movie ticket in Python

- Version 1: interwoven tests

```
def movieTicket(age):
 if age <= 16 :
   cout = 7
 elif age >= 65 :
   cout = 5
 else:
   cout = 10
 return cout


age = 18
print(movieTicket(age))
```

- Version 2: sequence of tests

```
def movieTicket(age):
 cout = 10;
 if age <= 16 :
   cout = 7
 if age >= 65 :
   cout = 5
 return cout


age = 18
print(movieTicket(age))
```

Question:

What is the output for my_function(350)

```
def my_function(x):
 if x < 100 :
  print ("Small!")
 elif x < 1000:
  print("ok!")
 else:
  return("B!")
```

Theme 3. Boolean expressions

Sub-theme: Boolean variables

- A Boolean variables can only have 2 possible values: TRUE(0) or FALSE (1)

Value assignments can be used

X =TRUE

Y = FALSE

A test result (Boolean expression) can also be assigned to a Boolean variable:

X = (A < 0)

Exercise 7 Positive value

- Derive an algorithm that checks if a given number X is strictly positive.

DATA:
RESULT:

HEADER:

MODULE

Exercise 7 – *Solution:* Positive value

Derive an algorithm that checks if a given number X is strictly positive..

DATA: X *(a real number)*
RESULT: Pos *(a Boolean variable true if x>0 otherwise false)*

HEADER: Pos = Positive (X)

MODULE

Pos = (X > 0)

Sub-theme: composed Boolean expressions

- A Composed Boolean expression (multiple conditions) has two or more simple Boolean connected with logical operators (*AND and OR*).

- Exercise 8 – Derive a Boolean expression that returns TRUE if a given age is between 16 and 65 not including them and FALSE otherwise.

Exercise 8 *Solution*

Derive a Boolean expression that returns TRUE if a given age is between 16 and 65 not including them and FALSE otherwise.

age > 16 AND age < 65

Sub-theme: Truth tables

- A truth table of a composed Boolean expression shows results for multiple possible value combination:

X	Y	X ET Y	X OU Y
VRAI	VRAI		
VRAI	FAUX		
FAUX	VRAI		
FAUX	FAUX		

NOT Operator

X	NOT X
TRUE	FALSE
FALSE	TRUE

- NOT is an operator used to get the complement of a simple value or of a complex Boolean expression:
- Example. If age = 15, then:
  - The expression age > 16 will be evaluated to FALSE, and NOT (age > 16) will be TRUE.
  - The expression age < 65 is TRUE, and NOT (age <65) is FALSE.

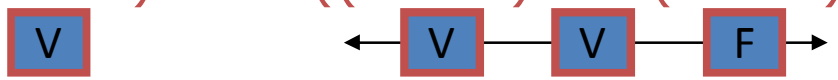
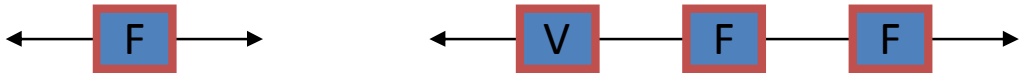
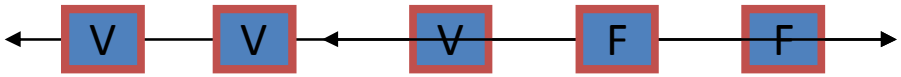
Exercise 9 – Complex Boolean expressions

Assume X = 5 and Y = 10.

Expression	Value
(X > 0) AND (NOT Y = 0)	
(X > 0) AND ((X < Y) OR (Y = 0)	
(NOT X > 0) OR ((X < Y) AND (Y = 0))	
NOT ((X > 0) OR ((X < Y) AND (Y = 0)))	

Exercise 9 – *Solution*: Complex Boolean expressions

Assume X = 5 and Y = 10.

Expression	Value
(X > 0) AND (NOT Y = 0) 	
(X > 0) AND ((X < Y) OR (Y = 0)) 	
(NOT X > 0) OR ((X < Y) AND (Y = 0)) 	
NOT ((X > 0) OR ((X < Y) AND (Y = 0))) 	

Sub-theme: Expressions in the tests

- A TEST in branches or loops can be any Boolean expression:
  - Boolean Variable
  - Negation of a Boolean variable
    - NOT
  - Comparison between 2 values of compatible types
    - Operators: **`==  !=  <  >  <=  >=`**
    - The Boolean compared data are not necessarily **`Boolean`**, but the result of the comparison is **`Boolean`**
  - Composed Boolean epressions
    - **`AND`**
    - **`OR`**

Examples en Python

```
x = 20
if ( x % 2 == 1) AND (x >= 10) :
  print("True!!!")
else:
  print("False!!!")
a, b = 10, 20
if ((a <= 10) and (b > 20)) or (a % 2 == 0) :
  print("True!!!")
else:
  print("False!!!")
```

Question:

What is the output of the following code?

```
a, b = 10, 20
if ((a % 2 == 1) and (b <= 10)) or (a % 2 == 0):
  print("True!!!")
else:
  print("False!!!")
```

Possible responses (**choose one**):

a) True!!!False!!!

b) True!!!

c) False!!!

d) error

Question – *Solution:*

What is the output?

a, b = 5, 10

```
if ((a % 2 == 1) and (b <= 10)) or (a % 2 == 0):
  print("True!!!")
else:
  print("False!!!")
```

Possible responses (**choose one**)::

a) True!!!False!!!

b) True!!!

Correcte response b)

Explanation: a % 2 is 1 and b <= 10 is TRUE, so the AND is TRUE. OR just need one of its argument to be TRUE so eventhough (a % 2 == 0 is false) the condition tested is still TRUE.

Conclusion

- The branching concept is essential to solve problems.
- The execution becomes dynamic depending on the chosen branch.