

# PARADIGMES DE PROGRAMMATION

DIPLOME DE FORMATION D'INGÉNIEURS EN INFORMATIQUE (FI-A1)

## *Module 3*

*Mondher Bouden*



**2025-2026**

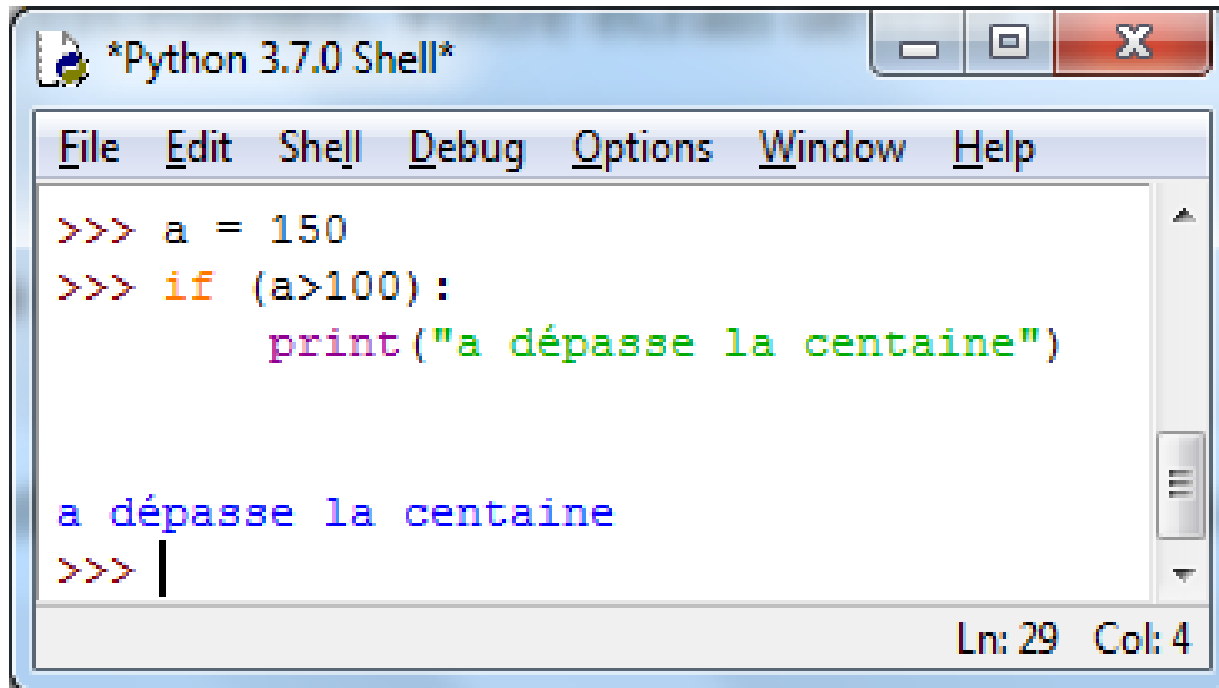
# **Les structures de contrôle**

(Instructions conditionnelles vs répétitives)

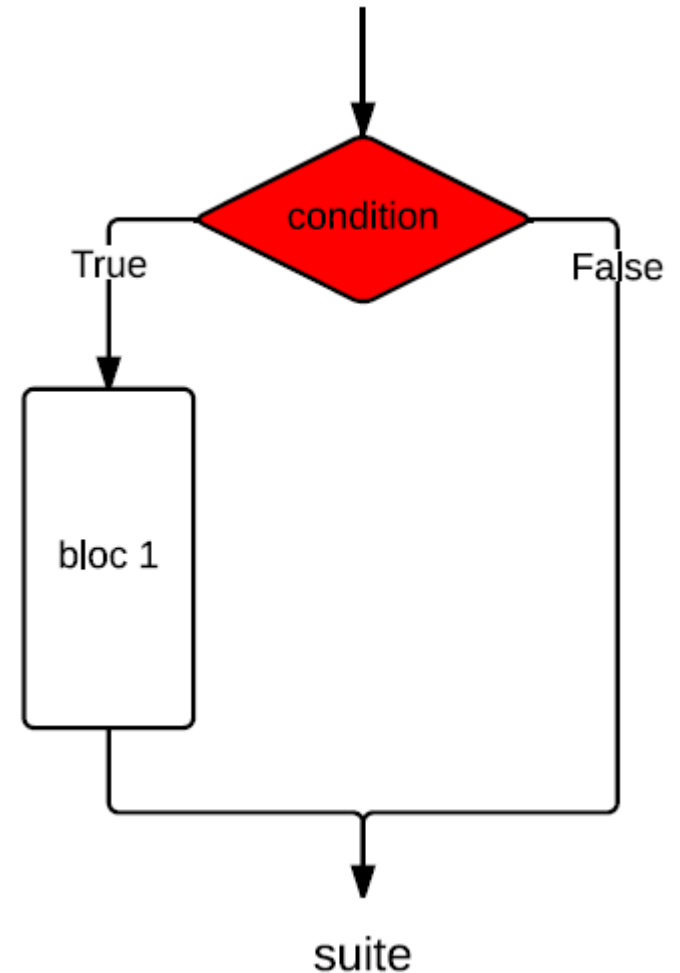
# Les instructions conditionnelles

## Structure *si*

- Si (une **condition** est vraie) alors
  - ▶ On exécute le bloc 1
- Suite



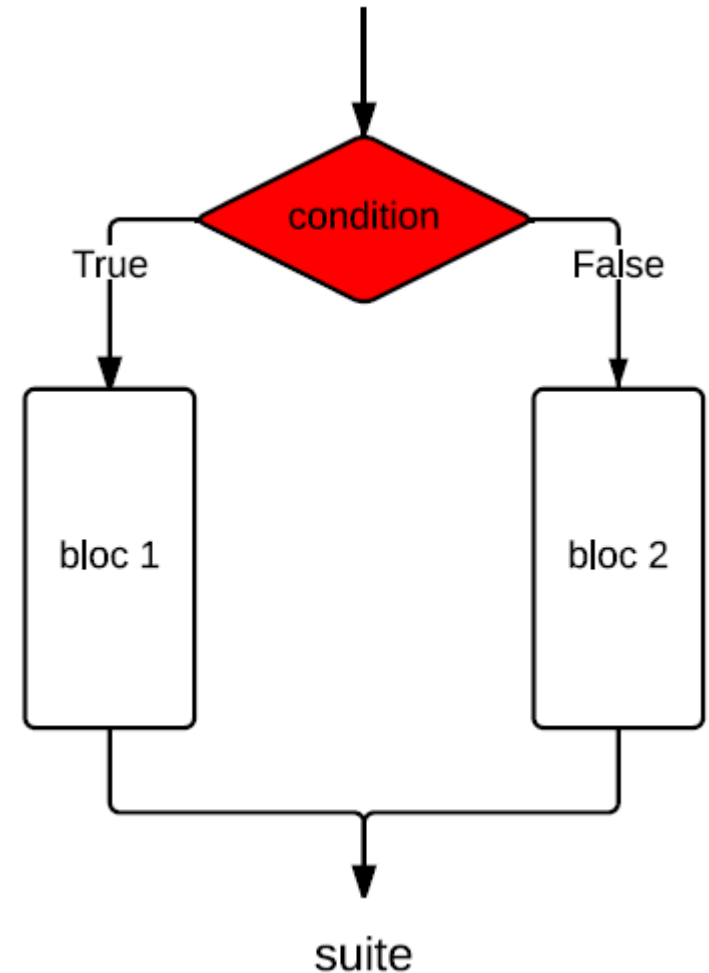
```
*Python 3.7.0 Shell*
File Edit Shell Debug Options Window Help
>>> a = 150
>>> if (a>100):
>>>     print("a dépasse la centaine")
a dépasse la centaine
>>> |
Ln: 29 Col: 4
```



# Les instructions conditionnelles

## Structure *si - sinon*

- Si (une **condition** est vraie) alors
  - ▶ On exécute le bloc 1
- Sinon
  - ▶ On exécute le bloc 2
- Suite



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> a = 20
>>> if (a > 100):
>>>     print("a dépasse la centaine")
else:
>>>     print("a ne dépasse pas cent")

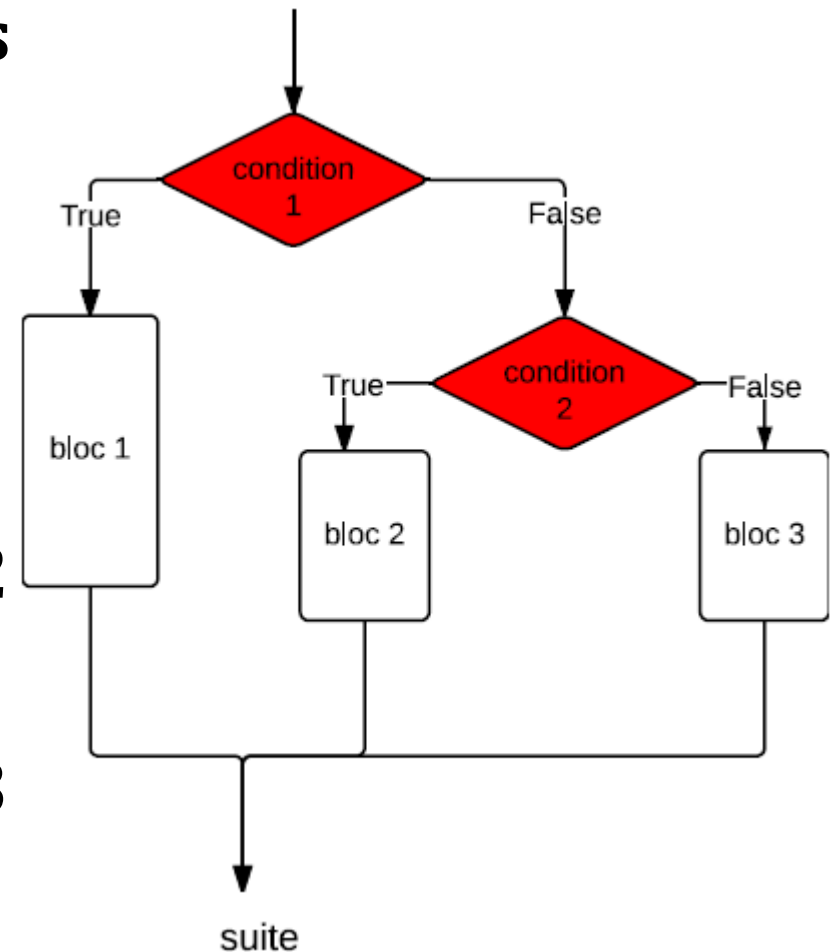
a ne dépasse pas cent
>>> |
```

The screenshot shows a Python 3.7.0 Shell window. The code defines a variable `a` with the value 20 and uses an `if-else` statement to check if `a` is greater than 100. Since the condition is false, the `else` branch is executed, printing "a ne dépasse pas cent". The status bar at the bottom indicates the cursor is at line 95, column 4.

# Les instructions conditionnelles

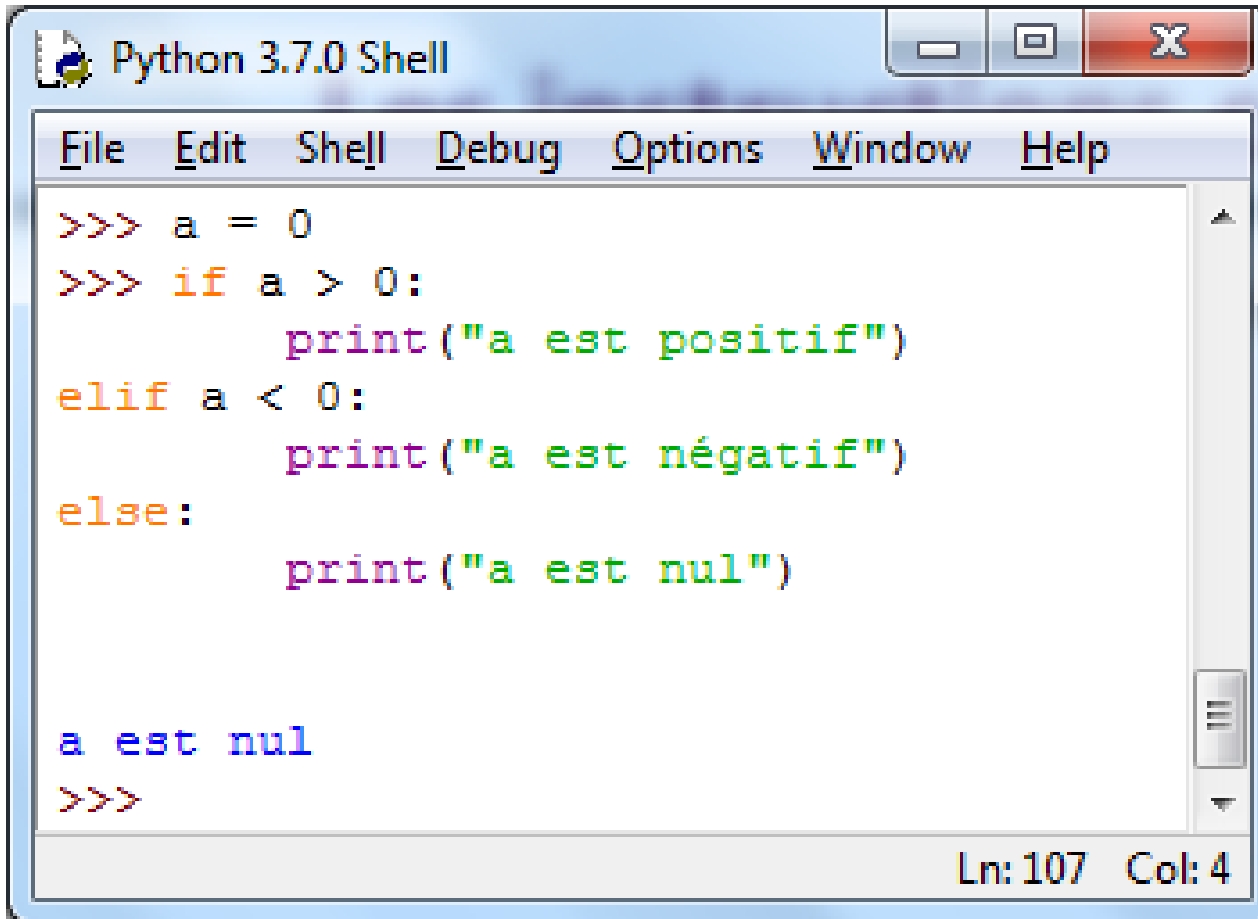
## Structure *si-sinon* imbriquées

- Si condition1 alors
  - ▶ On exécute le bloc 1
- Sinon
  - ▶ Si condition2 alors
    - On exécute le bloc 2
  - ▶ Sinon
    - On exécute le bloc 3
- Suite



# Les instructions conditionnelles

On peut également utiliser l'instruction **elif** (contraction de << else if >>) :



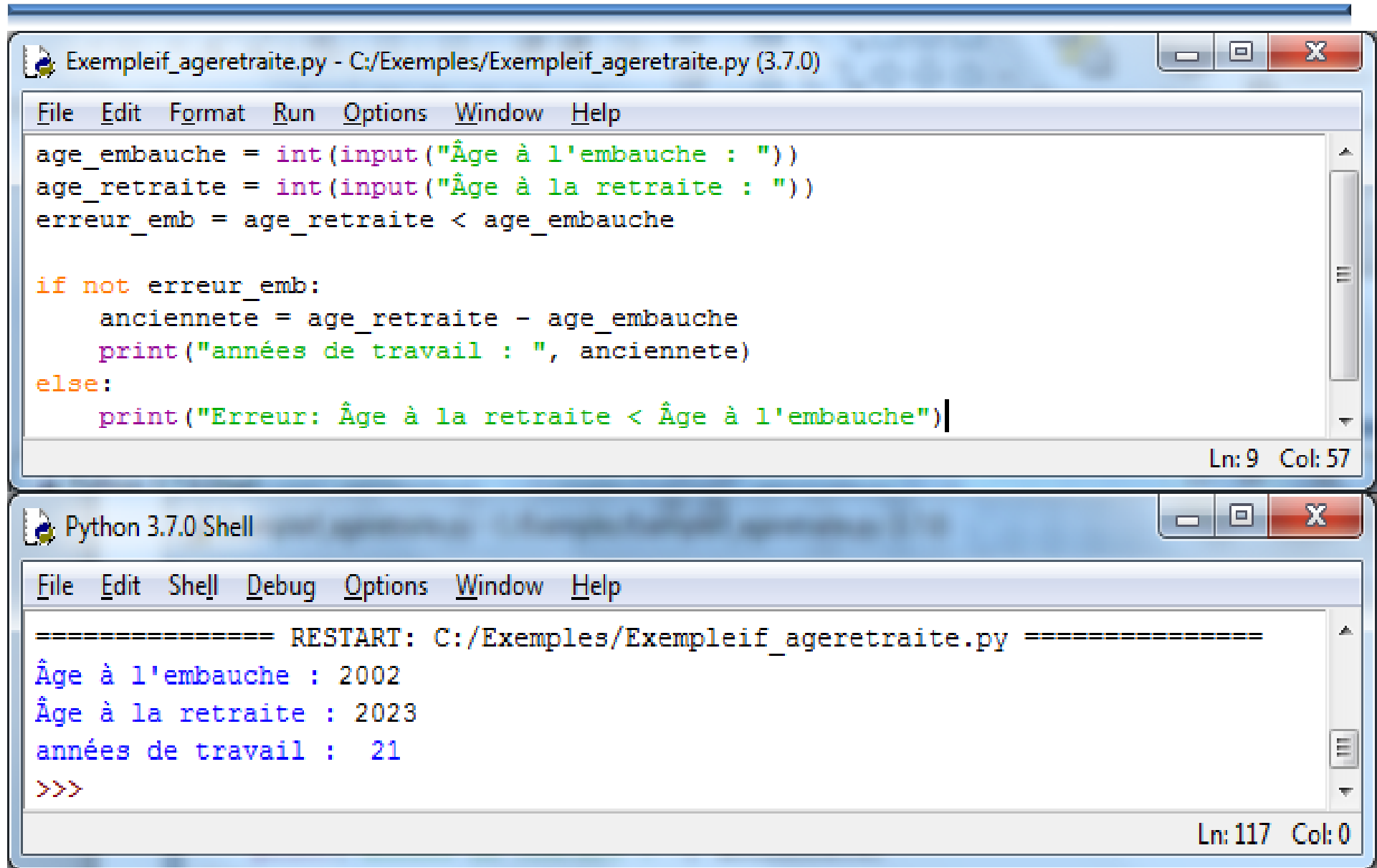
```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> a = 0
>>> if a > 0:
>>>     print("a est positif")
elif a < 0:
>>>     print("a est négatif")
else:
>>>     print("a est nul")

a est nul
>>>
```

Ln: 107 Col: 4

Les parenthèses (pour la condition du *if*) ne sont pas obligatoires.

# Exemple 1



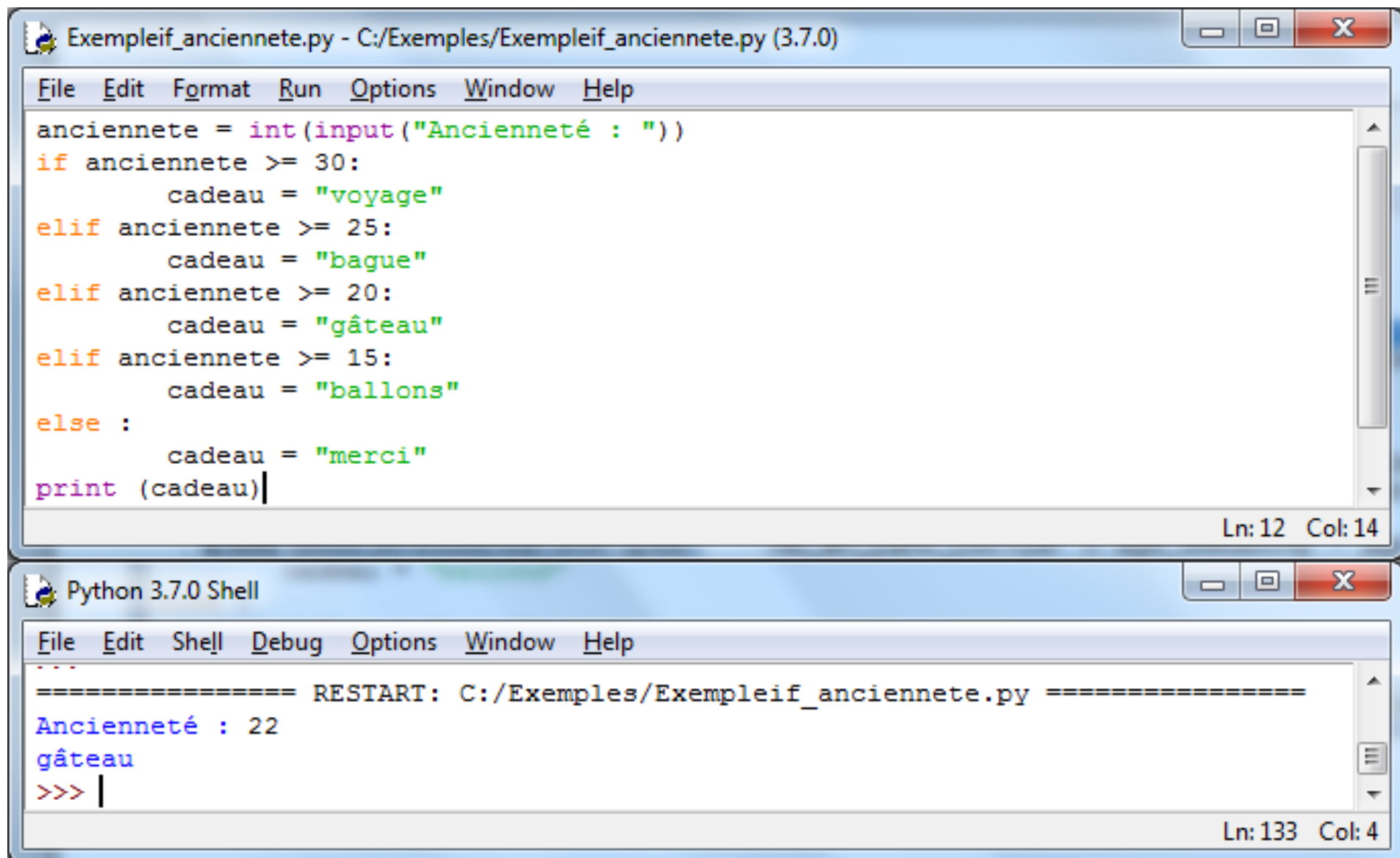
The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exempleif\_ageretraite.py - C:/Exemples/Exempleif\_ageretraite.py (3.7.0)', contains a Python script. The script prompts the user for their age at hire and age at retirement, calculates the number of years worked, and prints the result. The bottom window, titled 'Python 3.7.0 Shell', shows the output of the script after execution, with the user inputting 2002 for the hire age and 2023 for the retirement age, resulting in 21 years of work.

```
Exempleif_ageretraite.py - C:/Exemples/Exempleif_ageretraite.py (3.7.0)
File Edit Format Run Options Window Help
age_embauche = int(input("Âge à l'embauche : "))
age_retraite = int(input("Âge à la retraite : "))
erreur_emb = age_retraite < age_embauche

if not erreur_emb:
    anciennete = age_retraite - age_embauche
    print("années de travail : ", anciennete)
else:
    print("Erreur: Âge à la retraite < Âge à l'embauche")
Ln: 9 Col: 57
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:/Exemples/Exempleif_ageretraite.py =====
Âge à l'embauche : 2002
Âge à la retraite : 2023
années de travail : 21
>>>
Ln: 117 Col: 0
```

## Exemple 2



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exempleif\_anciennete.py - C:/Exemples/Exempleif\_anciennete.py (3.7.0)', contains a Python script. The script prompts the user for their age ('Ancienneté : ') and assigns a gift ('cadeau') based on the age using an if-elif-else structure. The bottom window, titled 'Python 3.7.0 Shell', shows the execution of the script. It displays the prompt 'Ancienneté : 22', the output 'gâteau', and the interactive prompt '>>>'. The status bars of both windows indicate the current line and column numbers.

```
File Edit Format Run Options Window Help
anciennete = int(input("Ancienneté : "))
if anciennete >= 30:
    cadeau = "voyage"
elif anciennete >= 25:
    cadeau = "bague"
elif anciennete >= 20:
    cadeau = "gâteau"
elif anciennete >= 15:
    cadeau = "ballons"
else :
    cadeau = "merci"
print (cadeau)
```

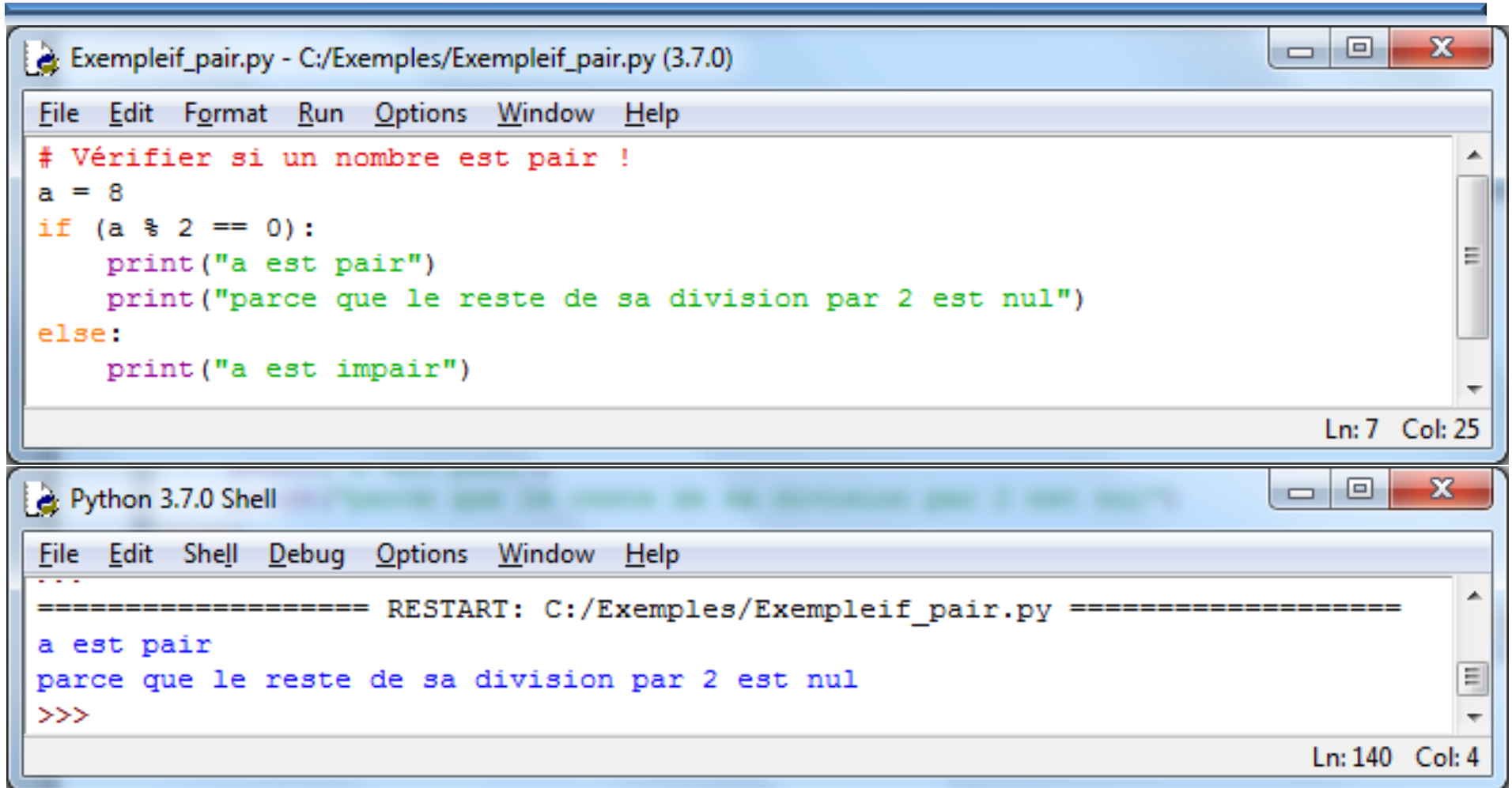
Ln: 12 Col: 14

```
File Edit Shell Debug Options Window Help
===== RESTART: C:/Exemples/Exempleif_anciennete.py =====
Ancienneté : 22
gâteau
>>> |
```

Ln: 133 Col: 4



# Exemple 3



The image shows two windows from a Python IDE. The top window, titled 'Exempleif\_pair.py - C:/Exemples/Exempleif\_pair.py (3.7.0)', contains a Python script. The script has a comment in French, assigns the value 8 to variable 'a', and uses an if-else statement to check if 'a' is even. The bottom window, titled 'Python 3.7.0 Shell', shows the output of running the script, which prints two lines of text in French. The status bars of both windows indicate the current line and column numbers.

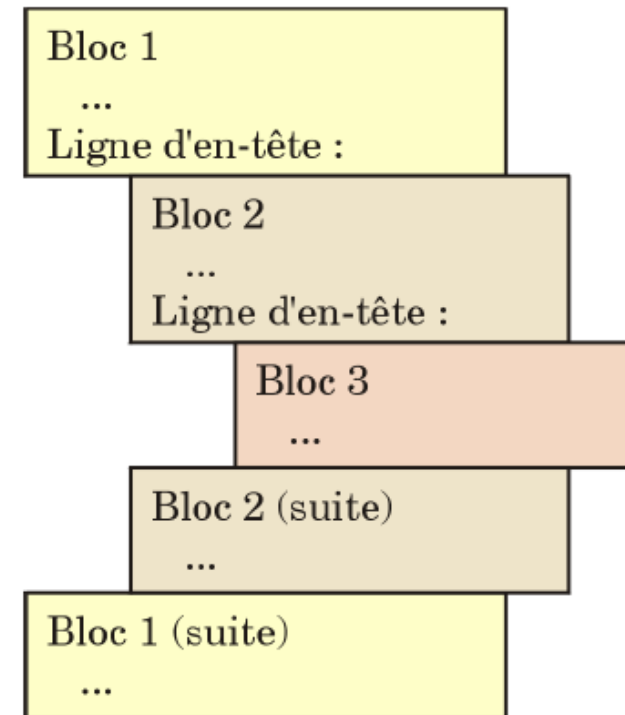
```
File Edit Format Run Options Window Help
# Vérifier si un nombre est pair !
a = 8
if (a % 2 == 0):
    print("a est pair")
    print("parce que le reste de sa division par 2 est nul")
else:
    print("a est impair")
Ln: 7 Col: 25
```

```
File Edit Shell Debug Options Window Help
===== RESTART: C:/Exemples/Exempleif_pair.py =====
a est pair
parce que le reste de sa division par 2 est nul
>>>
Ln: 140 Col: 4
```

- Notez bien que l'opérateur de comparaison pour l'égalité de deux valeurs est constitué de deux signes << égale >>.
- Un commentaire Python commence toujours par le caractère spécial #.

# Blocs d'instructions

- Dans la plupart des autres langages, un bloc d'instructions doit être délimité par des symboles spécifiques.
  - Des accolades pour C++ et Java.
- Avec Python, vous devez utiliser les sauts à la ligne et l'indentation.
  - Permettant d'écrire du code lisible, et à prendre de bonnes habitudes.
- Les blocs d'instructions sont toujours associés à une ligne d'en-tête contenant une instruction bien spécifique (if, elif, else, while, def, etc.) se terminant par un double point.
- Les blocs sont délimités par l'indentation : toutes les lignes d'un même bloc doivent être indentées exactement de la même manière.
- Le nombre d'espaces à utiliser pour l'indentation est quelconque, mais la plupart des programmeurs utilisent des multiples de 4.



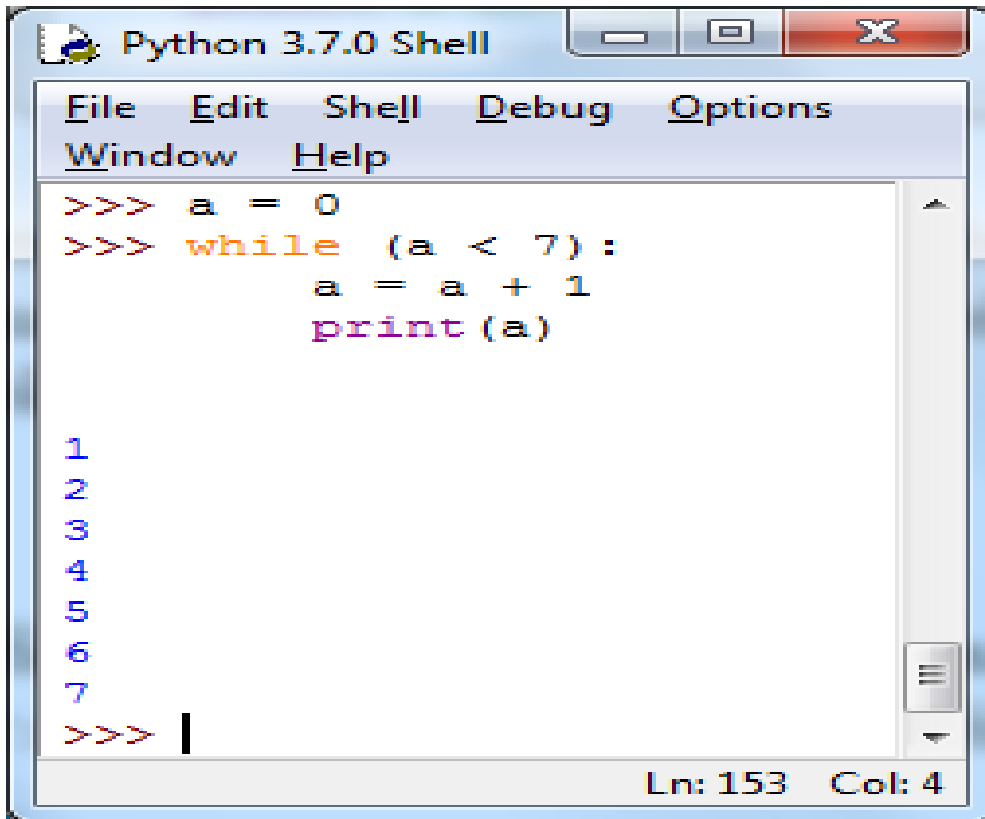
# Les instructions répétitives

---

- Nous distinguons deux types d'instruction répétitives: les boucles while et les boucles for:
- **Boucle while** : on exécute un bloc d'instructions tant que la condition est vraie
  - On ne peut pas (facilement) prédire le nombre d'exécutions du bloc d'instructions, car cela dépend d'une condition à évaluer
- **Boucle for** : on exécute un bloc d'instructions un certain nombre de fois
  - On peut prédire (généralement par calcul) le nombre d'exécutions du bloc d'instructions
  - On exécute systématiquement le bloc d'instructions ce nombre de fois
  - Utilisée pour **parcourir** les éléments d'une structure *itérable*.

# La boucle while

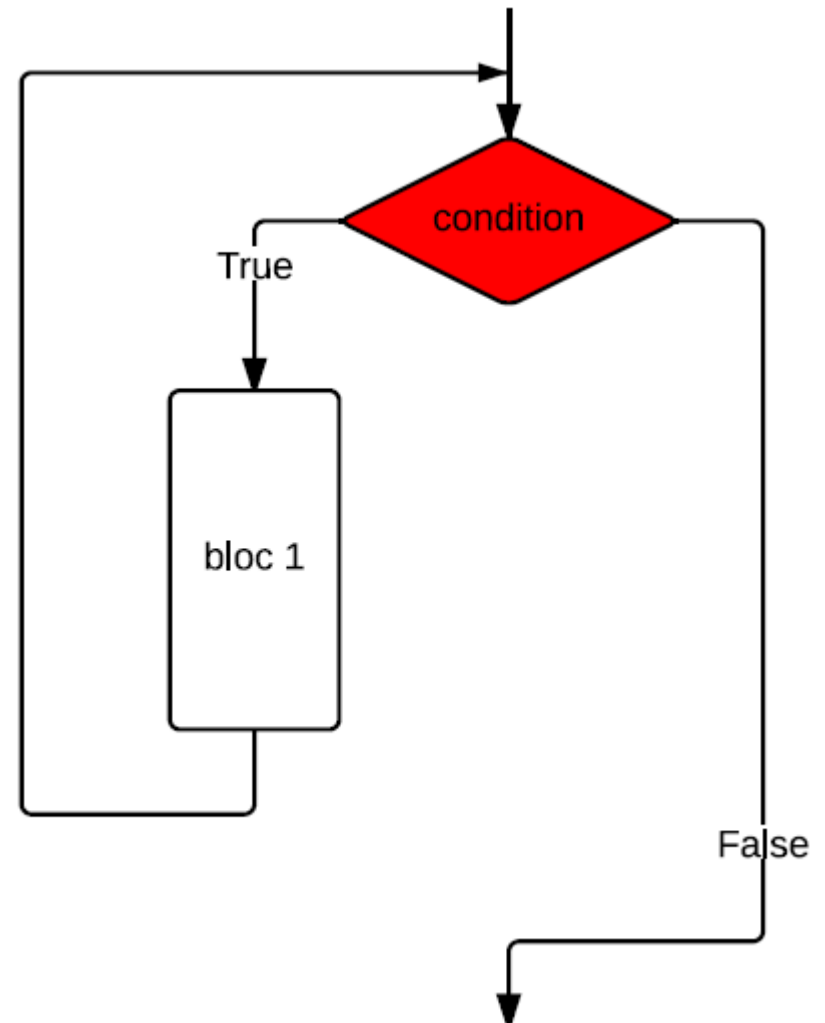
- Tant que condition :
  - ▶ On exécute le bloc 1
- Suite



```
Python 3.7.0 Shell
File Edit Shell Debug Options
Window Help
>>> a = 0
>>> while (a < 7):
    a = a + 1
    print(a)

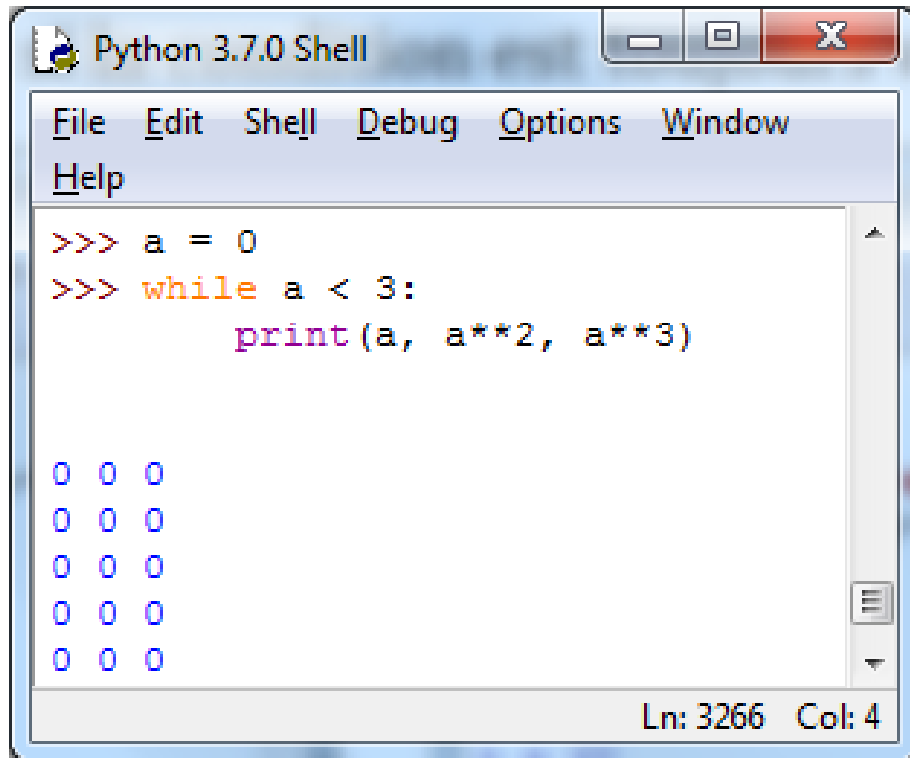
1
2
3
4
5
6
7
>>> |
```

Ln: 153 Col: 4



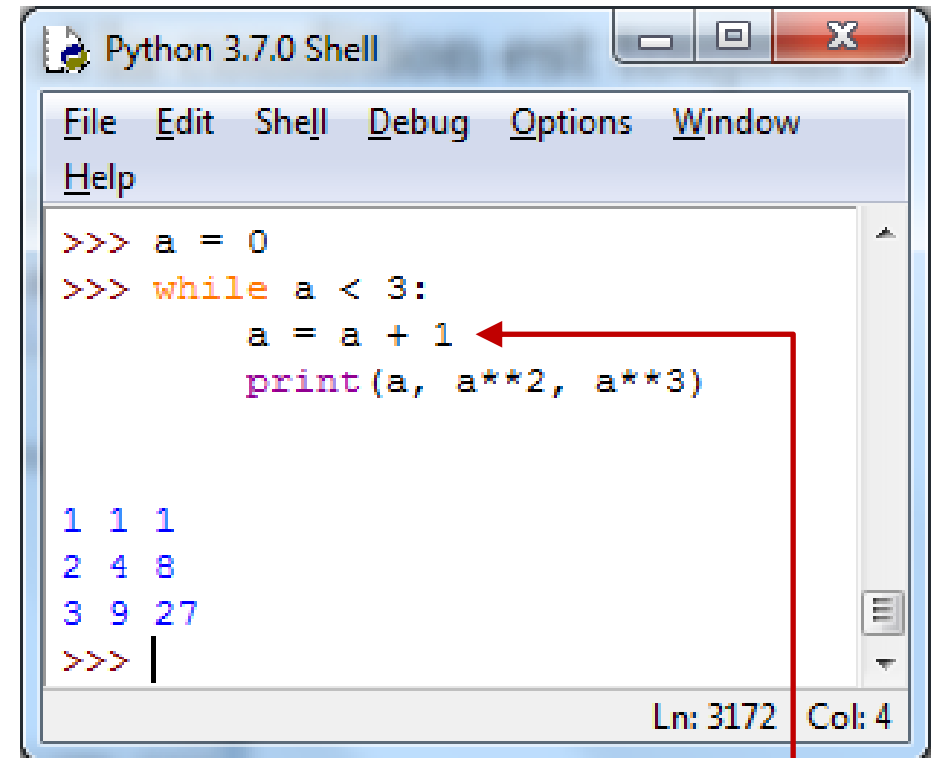
# La boucle while

- Qu'arrive-t-il si la condition est toujours vraie ?



```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> a = 0
>>> while a < 3:
>>>     print(a, a**2, a**3)

0 0 0
0 0 0
0 0 0
0 0 0
0 0 0
Ln: 3266 Col: 4
```



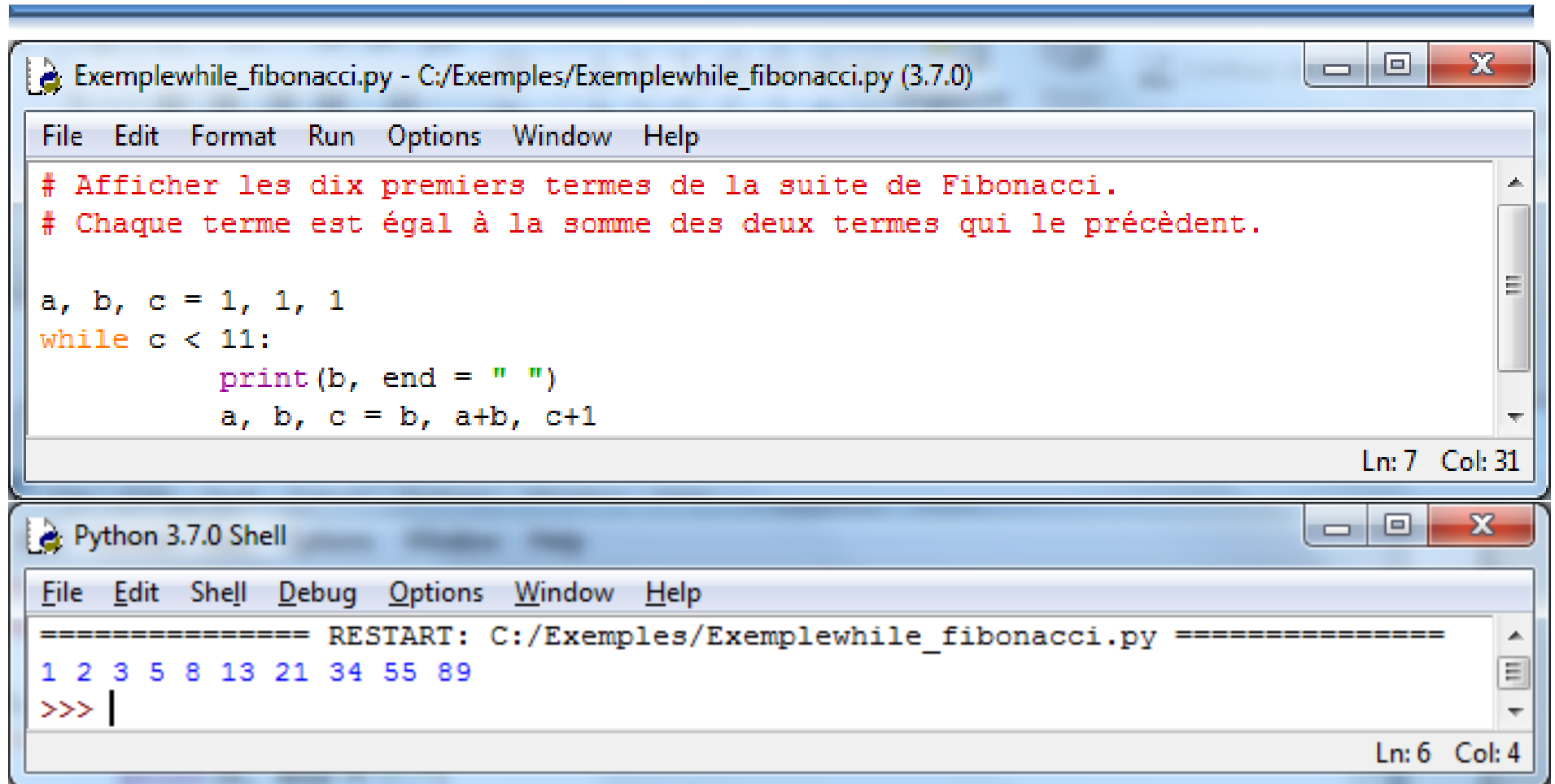
```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> a = 0
>>> while a < 3:
>>>     a = a + 1
>>>     print(a, a**2, a**3)

1 1 1
2 4 8
3 9 27
>>> |
Ln: 3172 Col: 4
```

La fonction **print()** permet d'afficher plusieurs expressions l'une à la suite de l'autre sur la même ligne. Python insère automatiquement un espace entre les éléments affichés.

Incrémentation  
(ou `a += 1`)

# Exemple 1



The image shows two windows from a Python IDE. The top window, titled 'Exemplewhile\_fibonacci.py - C:/Exemples/Exemplewhile\_fibonacci.py (3.7.0)', contains a Python script. The script has a menu bar (File, Edit, Format, Run, Options, Window, Help) and a code editor. The code is written in red and black text. It includes two comments in red, followed by variable initialization and a while loop that prints Fibonacci numbers with spaces. The status bar at the bottom right of this window shows 'Ln: 7 Col: 31'. The bottom window, titled 'Python 3.7.0 Shell', has a menu bar (File, Edit, Shell, Debug, Options, Window, Help) and a shell area. It shows the output of the script as a single line of numbers separated by spaces, followed by a prompt '>>>>'. The status bar at the bottom right of this window shows 'Ln: 6 Col: 4'.

```
# Afficher les dix premiers termes de la suite de Fibonacci.  
# Chaque terme est égal à la somme des deux termes qui le précèdent.  
  
a, b, c = 1, 1, 1  
while c < 11:  
    print(b, end = " ")  
    a, b, c = b, a+b, c+1
```

Ln: 7 Col: 31

```
===== RESTART: C:/Exemples/Exemplewhile_fibonacci.py =====  
1 2 3 5 8 13 21 34 55 89  
>>> |
```

Ln: 6 Col: 4

- L'argument **end** = " " permet de remplacer le saut à la ligne par un simple espace.
- Si vous supprimez cet argument, les nombres seront affichés les uns en-dessous des autres.

## Exemple 2 : calculer $x^y$

---

$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ multiplications})$$

$$x^4 = x \cdot x \cdot x \cdot x$$

$$x^3 = x \cdot x \cdot x$$

$$x^2 = x \cdot x$$

$$x^1 = x$$

$$x^0 = 1$$

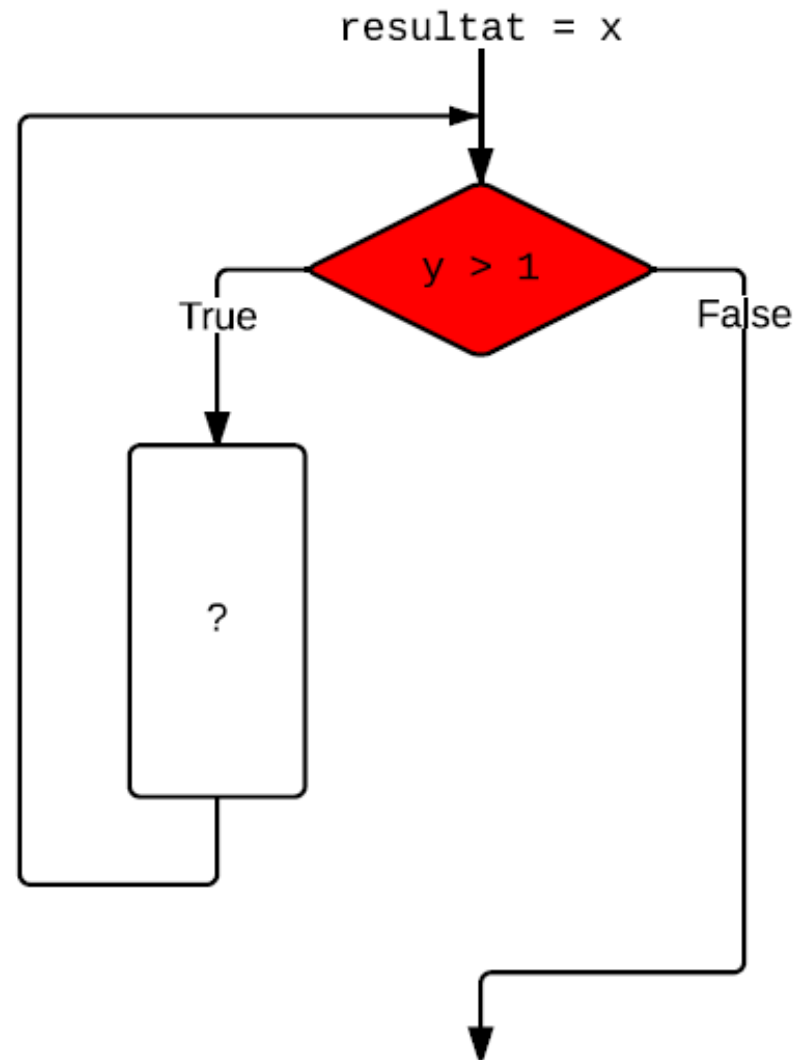
## Exemple 2 : calculer $x^y$

$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
resultat = x
```

```
...
```

```
print(resultat)
```





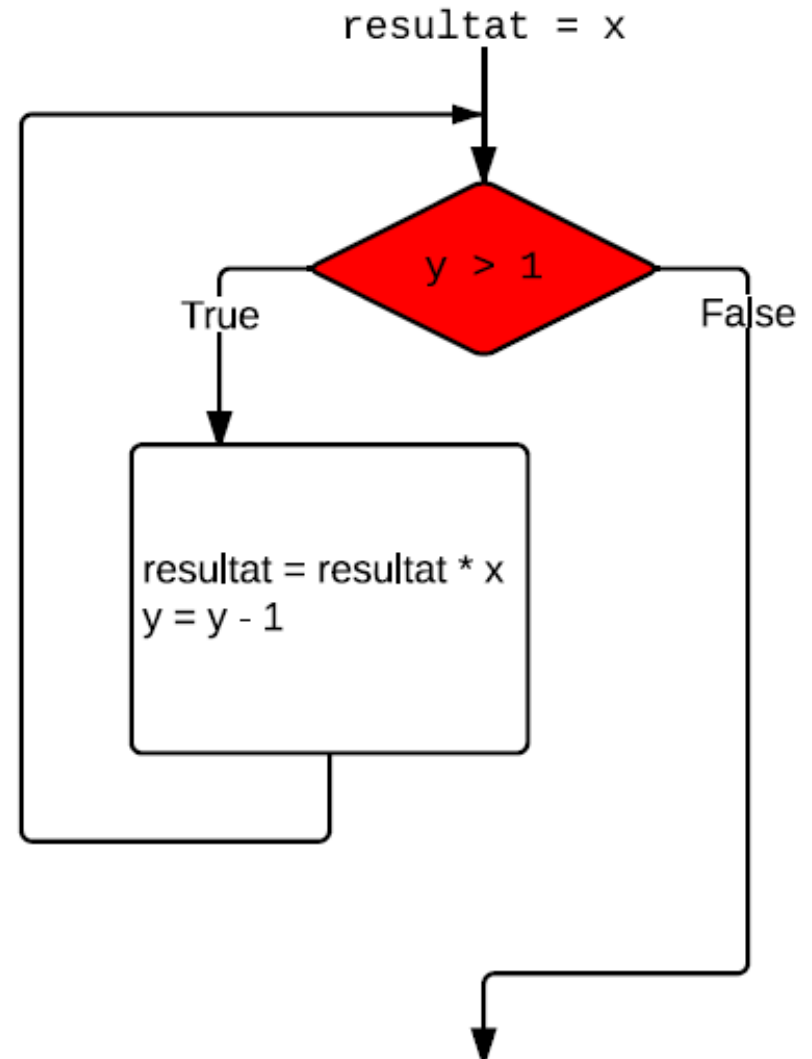
## Exemple 2 : calculer $x^y$

$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
resultat = x
```

```
while y > 1:  
    resultat = resultat * x  
    y = y - 1
```

```
print(resultat)
```



## Exemple 2 : calculer $x^y$

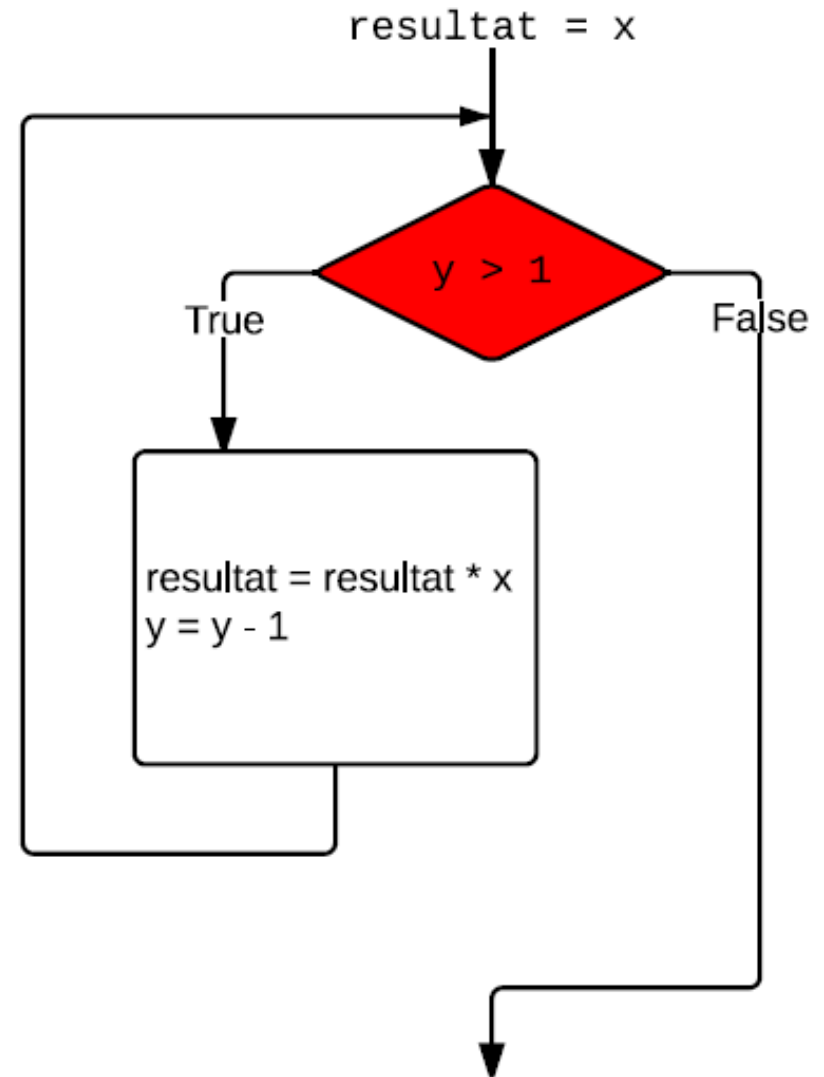
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
resultat = x
```

```
while y > 1:  
    resultat = resultat * x  
    y = y - 1
```

```
print(resultat)
```

Est-ce que la boucle produit toujours le bon résultat ?



## Exemple 2 : calculer $x^y$

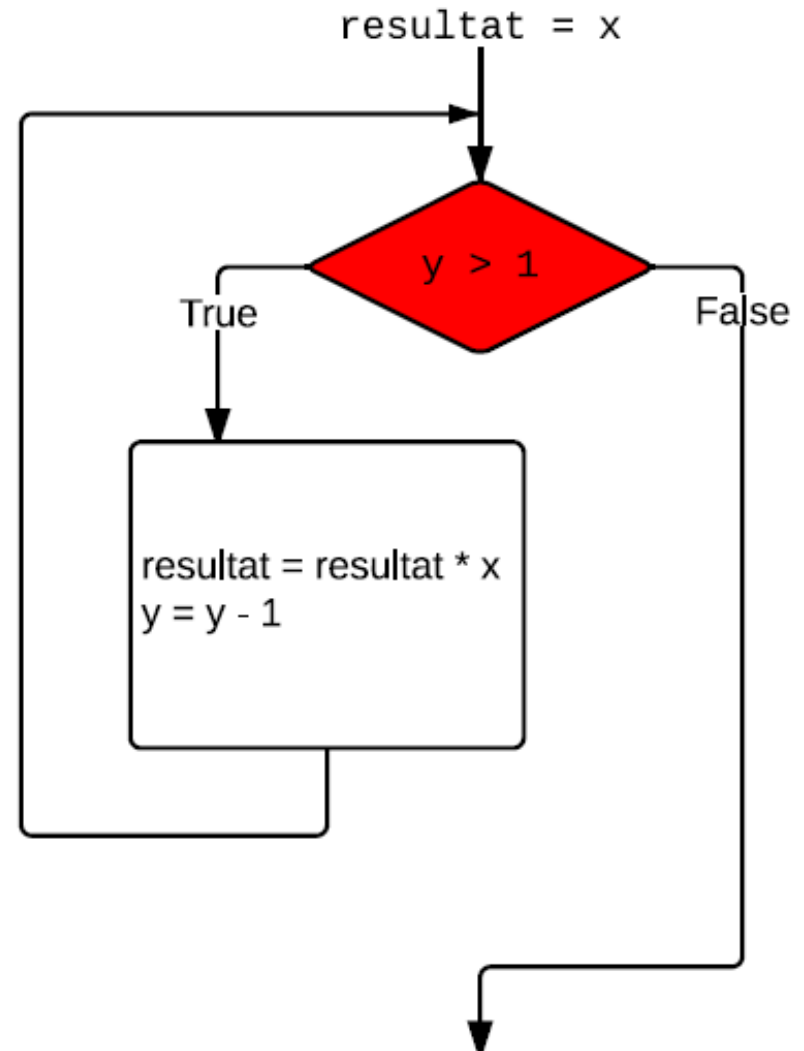
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

resultat = x

```
while y > 1:  
    resultat = resultat * x  
    y = y - 1
```

```
print(resultat)
```

Et si y est négatif ou nul ?



## Exemple 2 : calculer $x^y$

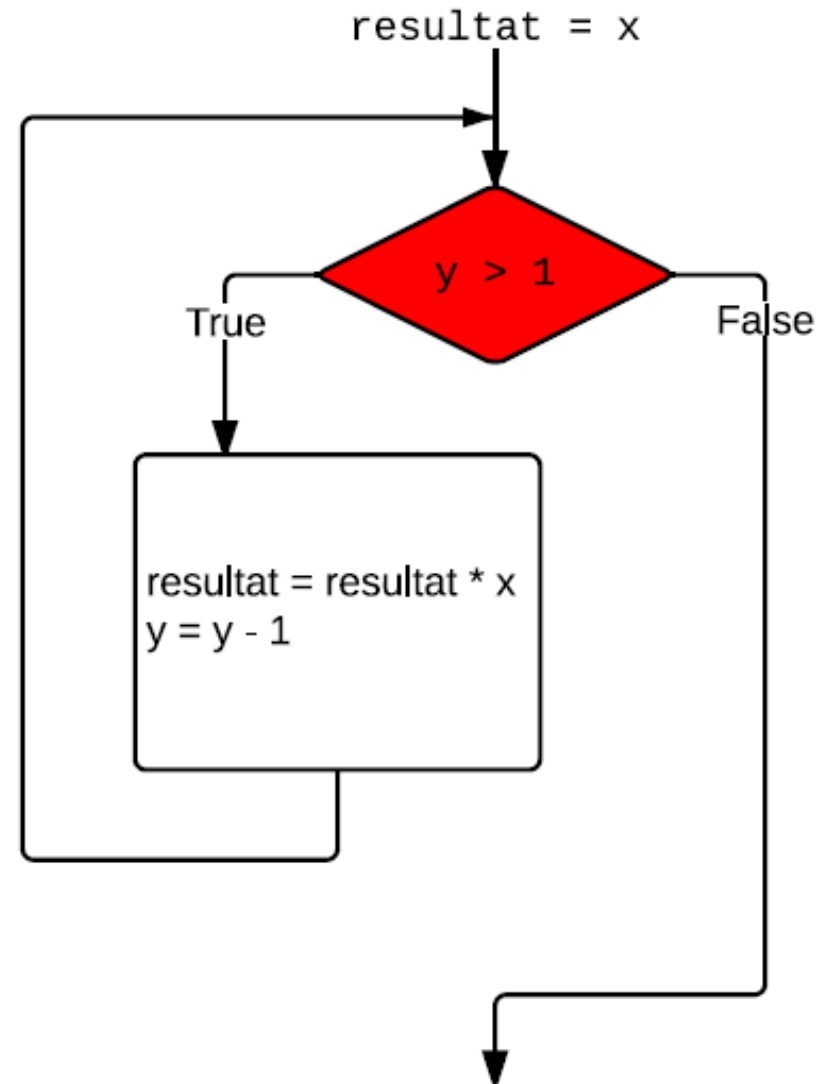
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
resultat = x
```

```
while y > 1:  
    resultat = resultat * x  
    y = y - 1
```

```
print(resultat)
```

Comment modifier le code pour inclure le cas de  $y = 0$  ?

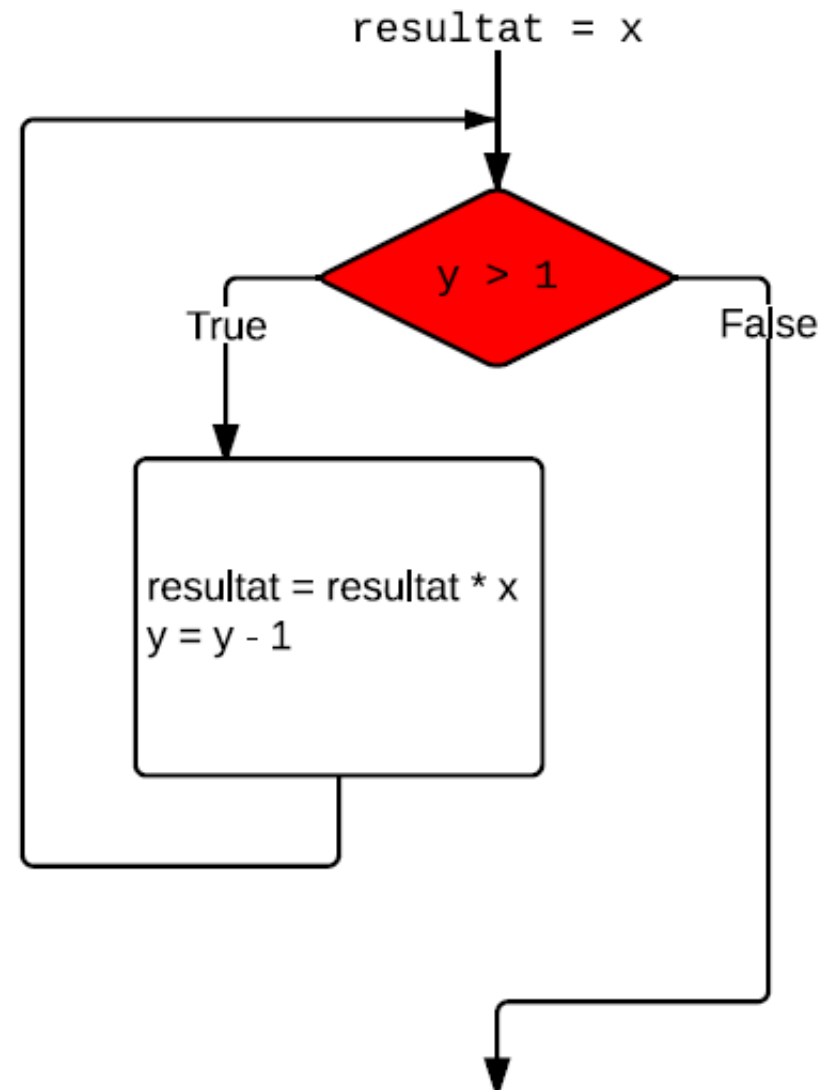


## Exemple 2 : calculer $x^y$

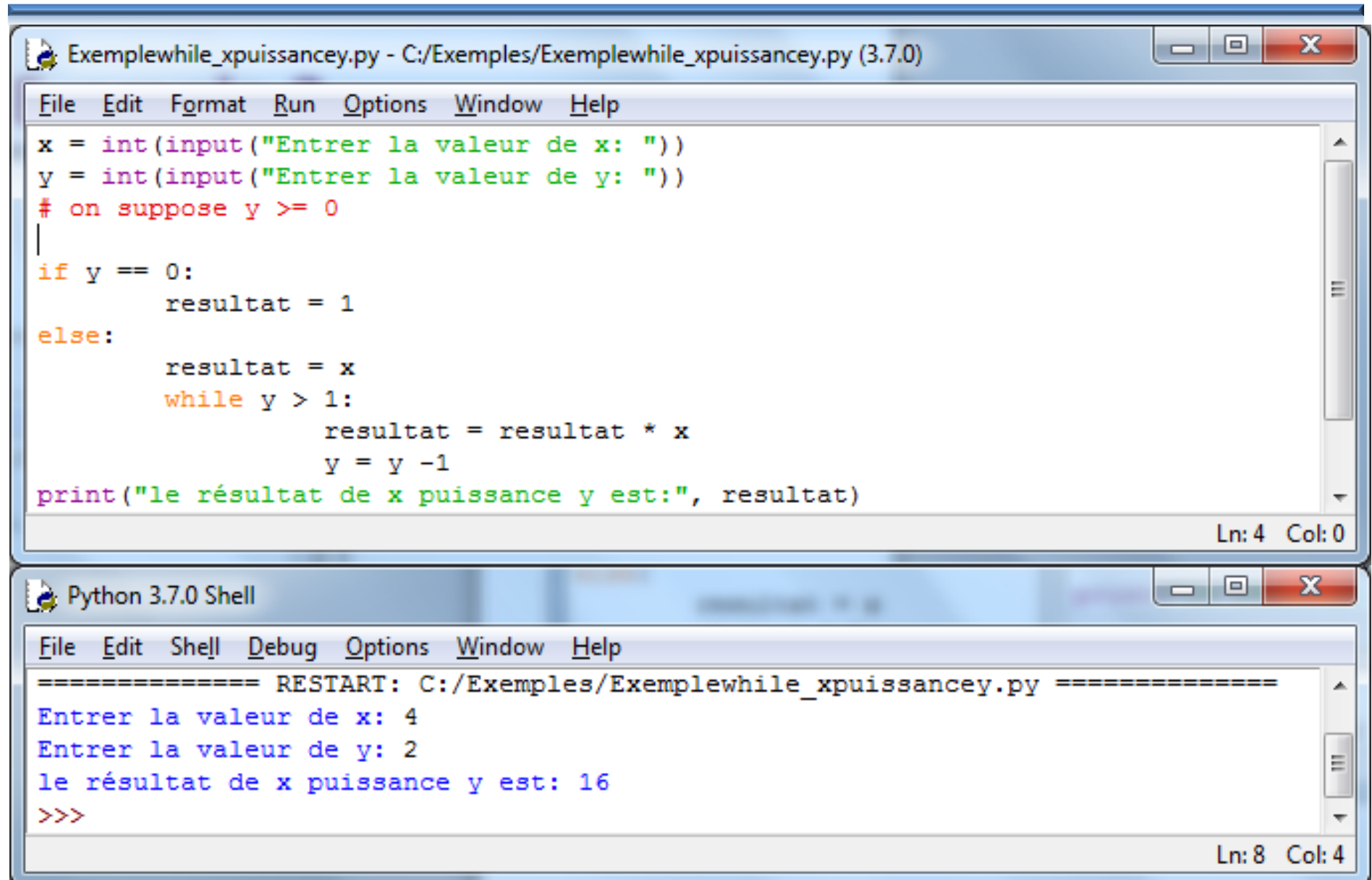
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

# On suppose  $y \geq 0$

```
if y == 0:
    resultat = 1
else:
    resultat = x
    while y > 1:
        resultat = resultat * x
        y = y - 1
print(resultat)
```



## Exemple 2: calculer $x^y$



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exemplewhile\_xpuissancey.py - C:/Exemples/Exemplewhile\_xpuissancey.py (3.7.0)', contains the following Python code:

```
File Edit Format Run Options Window Help
x = int(input("Entrer la valeur de x: "))
y = int(input("Entrer la valeur de y: "))
# on suppose y >= 0
|
if y == 0:
    resultat = 1
else:
    resultat = x
    while y > 1:
        resultat = resultat * x
        y = y - 1
print("le résultat de x puissance y est:", resultat)
```

The bottom window, titled 'Python 3.7.0 Shell', shows the execution of the script:

```
File Edit Shell Debug Options Window Help
===== RESTART: C:/Exemples/Exemplewhile_xpuissancey.py =====
Entrer la valeur de x: 4
Entrer la valeur de y: 2
le résultat de x puissance y est: 16
>>>
```

## Exemple 3 : Boucles while imbriquées

- Exemple d'utilisation : calculer une table de multiplication [1 à x] par [1 à y], où x et y sont deux paramètres tels que  $x \geq 1$  et  $y \geq 1$ .

1	1	2	3	4	5	...	y
1	1	2	3	4	5	...	y
2	2	4	6	8	10	...	$2 \cdot y$
3	3	6	9	12	15	...	$3 \cdot y$
...	...	...	...	...	...	...	...
x	x	$x \cdot 2$	$x \cdot 3$	$x \cdot 4$	$x \cdot 5$	...	$x \cdot y$

# Exemple 3 : Boucles while imbriquées

---

- Exemple d'utilisation : calculer une table de multiplication [1 à x] par [1 à y], où x et y sont deux paramètres tels que  $x \geq 1$  et  $y \geq 1$ .

```
# On suppose que  $x \geq 1$  et  $y \geq 1$ 
```

```
multiplicateur = 1
```

```
while multiplicateur <= x: # création d'une ligne  
    print("Les multiples de ", multiplicateur,  
          " sont : ", end="")
```

```
...
```

```
    multiplicateur = multiplicateur + 1 # on passe à la  
                                       # ligne suivante  
                                       # du tableau
```



# Exemple 3 : Boucles while imbriquées

---

- Exemple d'utilisation : calculer une table de multiplication [1 à x] par [1 à y], où x et y sont deux paramètres tels que  $x \geq 1$  et  $y \geq 1$ .

# On suppose que  $x \geq 1$  et  $y \geq 1$

```
multiplicateur = 1
```

```
while multiplicateur <= x: # création d'une ligne
    print("Les multiples de ", multiplicateur,
          " sont : ", end="")
```

```
    multiplicande = 1
```

```
    while multiplicande <= y:
```

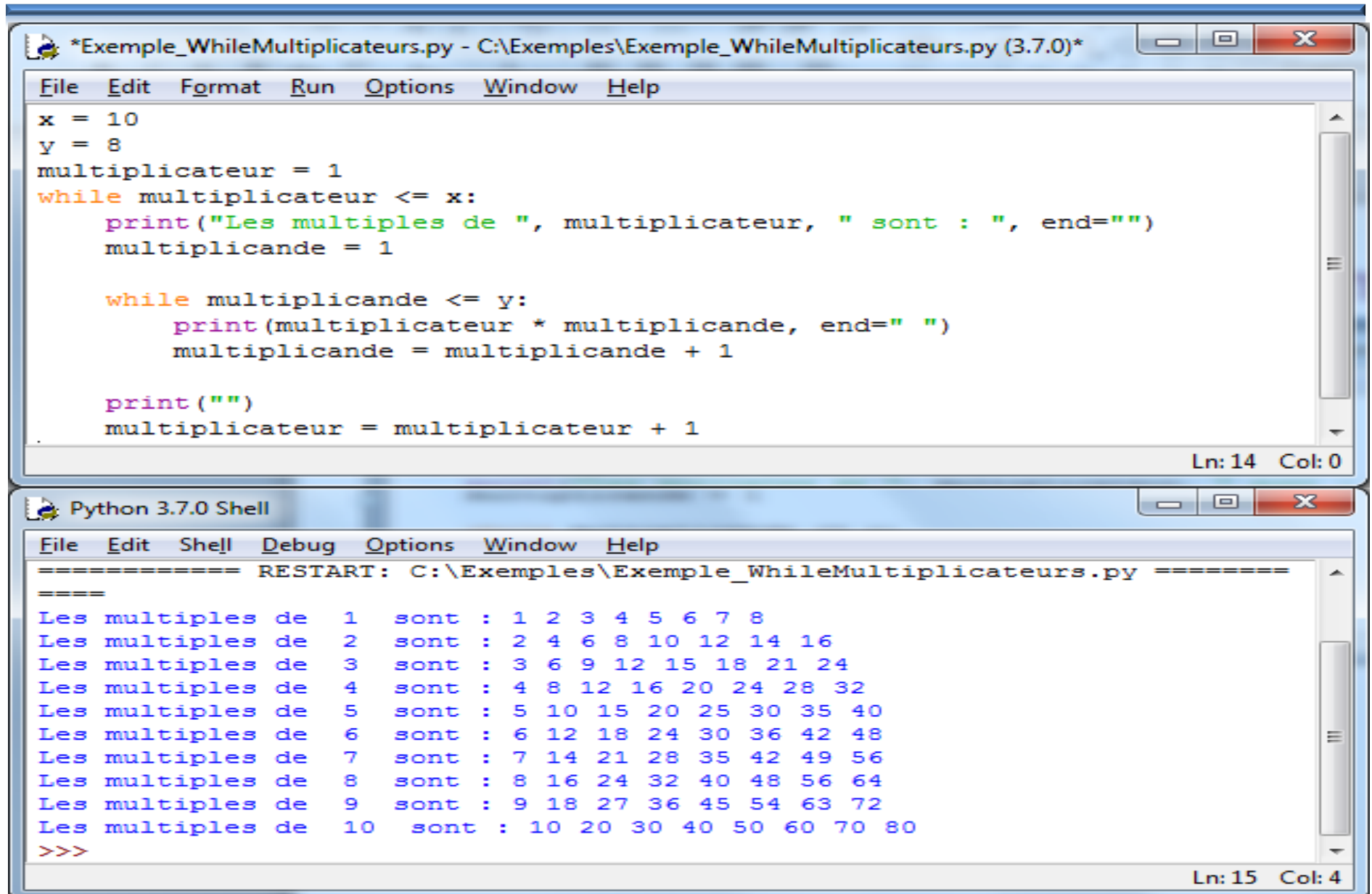
```
        print(multiplicateur * multiplicande, end=" ")
```

```
        multiplicande = multiplicande + 1
```

```
    print("")
```

```
    multiplicateur = multiplicateur + 1 # on passe à la
                                         # ligne suivante
                                         # du tableau
```

# Exemple 3 : Boucles while imbriquées



The image shows a Python IDE window titled '\*Exemple\_WhileMultiplicateurs.py - C:\Exemples\Exemple\_WhileMultiplicateurs.py (3.7.0)\*'. The code defines variables x=10 and y=8, then uses nested while loops to print the multiples of each number from 1 to x and y respectively. The output window, titled 'Python 3.7.0 Shell', shows the execution results, displaying the multiples of 1 through 10.

```
*Exemple_WhileMultiplicateurs.py - C:\Exemples\Exemple_WhileMultiplicateurs.py (3.7.0)*
File Edit Format Run Options Window Help
x = 10
y = 8
multiplicateur = 1
while multiplicateur <= x:
    print("Les multiples de ", multiplicateur, " sont : ", end="")
    multiplicande = 1

    while multiplicande <= y:
        print(multiplicateur * multiplicande, end=" ")
        multiplicande = multiplicande + 1

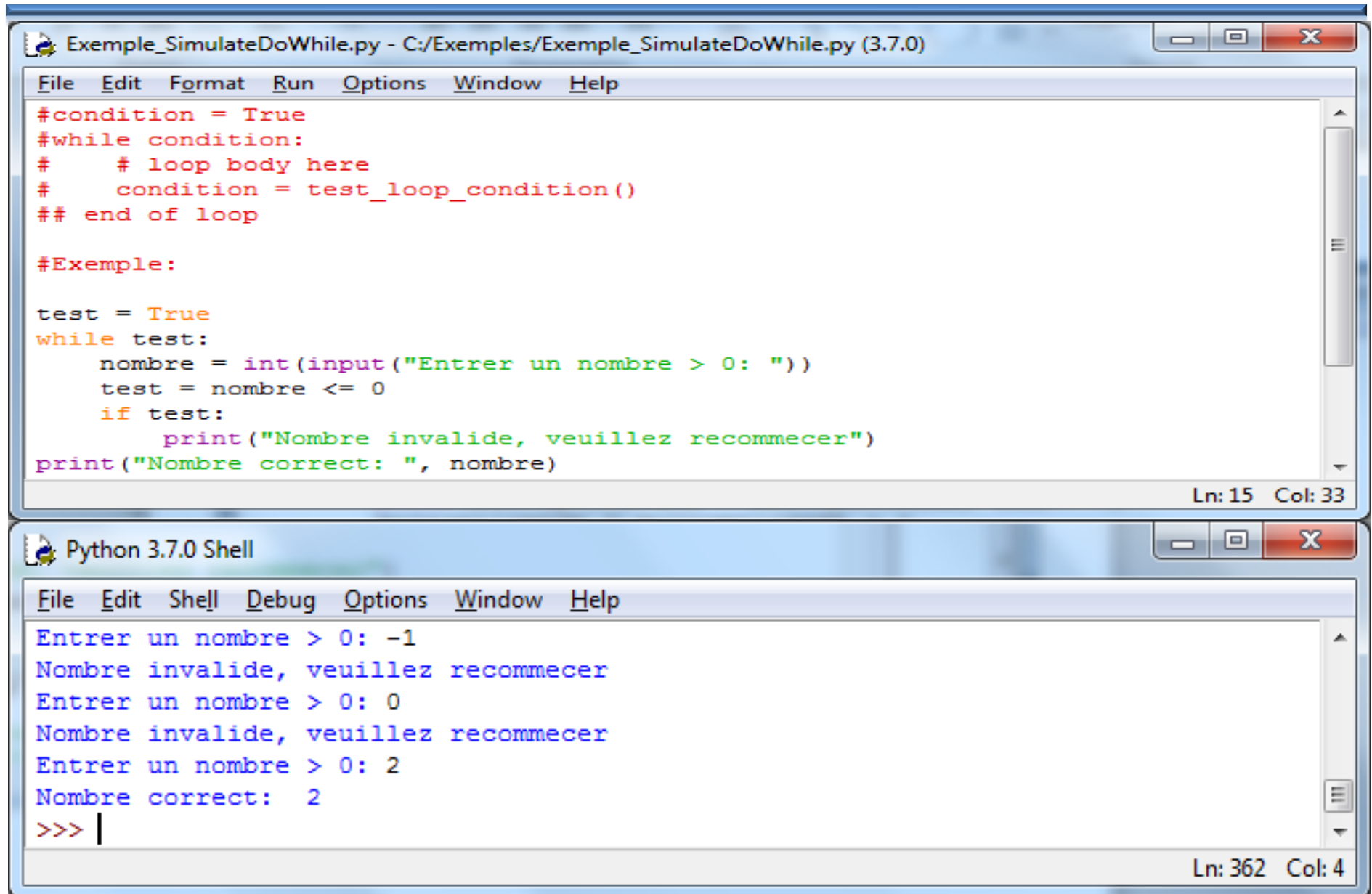
    print("")
    multiplicateur = multiplicateur + 1
.
```

Ln: 14 Col: 0

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
===== RESTART: C:\Exemples\Exemple_WhileMultiplicateurs.py =====
=====
Les multiples de 1 sont : 1 2 3 4 5 6 7 8
Les multiples de 2 sont : 2 4 6 8 10 12 14 16
Les multiples de 3 sont : 3 6 9 12 15 18 21 24
Les multiples de 4 sont : 4 8 12 16 20 24 28 32
Les multiples de 5 sont : 5 10 15 20 25 30 35 40
Les multiples de 6 sont : 6 12 18 24 30 36 42 48
Les multiples de 7 sont : 7 14 21 28 35 42 49 56
Les multiples de 8 sont : 8 16 24 32 40 48 56 64
Les multiples de 9 sont : 9 18 27 36 45 54 63 72
Les multiples de 10 sont : 10 20 30 40 50 60 70 80
>>>
```

Ln: 15 Col: 4

# Simuler la boucle do..while (∄ en Python)



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exemple\_SimulateDoWhile.py - C:/Exemples/Exemple\_SimulateDoWhile.py (3.7.0)', contains Python code that simulates a do-while loop. The code defines a function 'test\_loop\_condition()' that returns True if a number is greater than 0, and False otherwise. It then uses a 'while' loop to repeatedly prompt the user for a number until a valid number (greater than 0) is entered. The bottom window, titled 'Python 3.7.0 Shell', shows the execution of the code. It displays the prompts 'Entrer un nombre > 0:' and the corresponding user input. The first two inputs are -1 and 0, both of which result in the message 'Nombre invalide, veuillez recommencer'. The third input is 2, which results in the message 'Nombre correct: 2'. The shell prompt '>>>' is visible at the bottom.

```
Exemple_SimulateDoWhile.py - C:/Exemples/Exemple_SimulateDoWhile.py (3.7.0)
File Edit Format Run Options Window Help
#condition = True
#while condition:
#    # loop body here
#    condition = test_loop_condition()
## end of loop

#Exemple:

test = True
while test:
    nombre = int(input("Entrer un nombre > 0: "))
    test = nombre <= 0
    if test:
        print("Nombre invalide, veuillez recommencer")
print("Nombre correct: ", nombre)
Ln: 15 Col: 33
```

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
Entrer un nombre > 0: -1
Nombre invalide, veuillez recommencer
Entrer un nombre > 0: 0
Nombre invalide, veuillez recommencer
Entrer un nombre > 0: 2
Nombre correct: 2
>>> |
Ln: 362 Col: 4
```

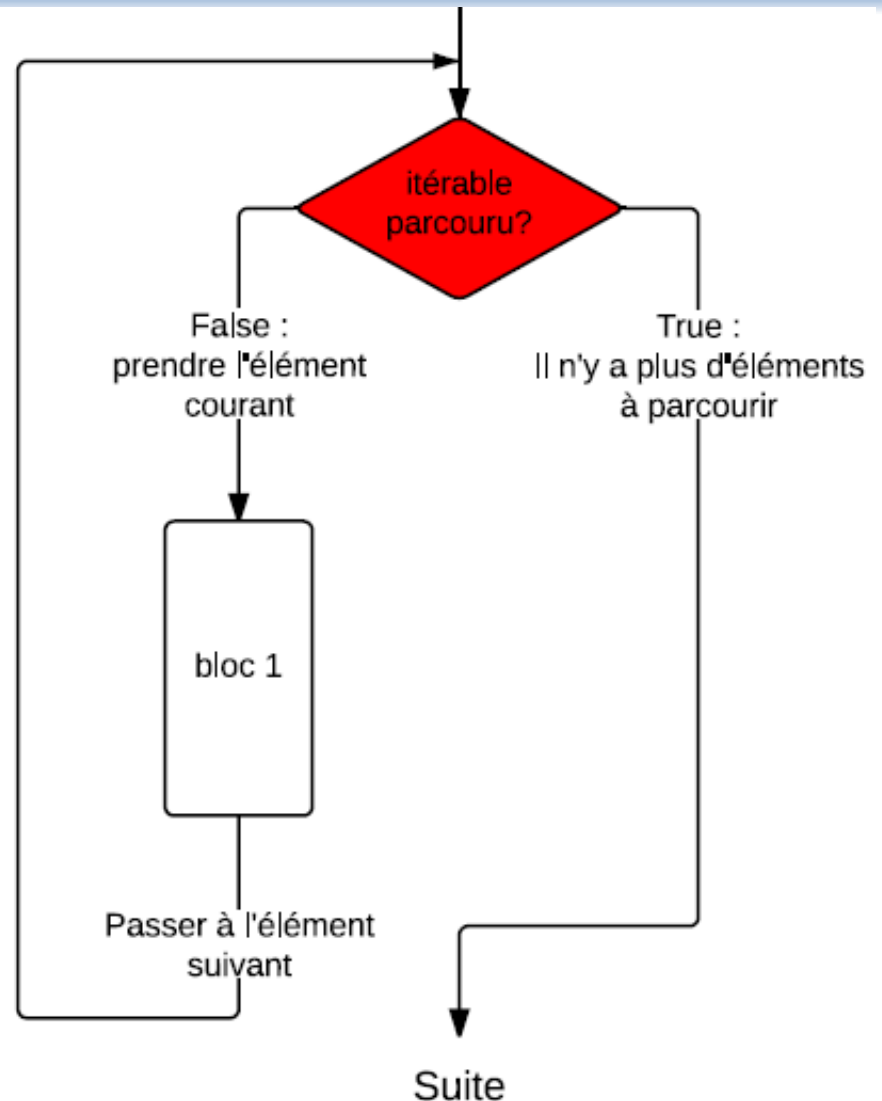
# La boucle for

- Pour *cible* dans *itérable* :
  - ▶ On exécute le bloc 1
- Suite

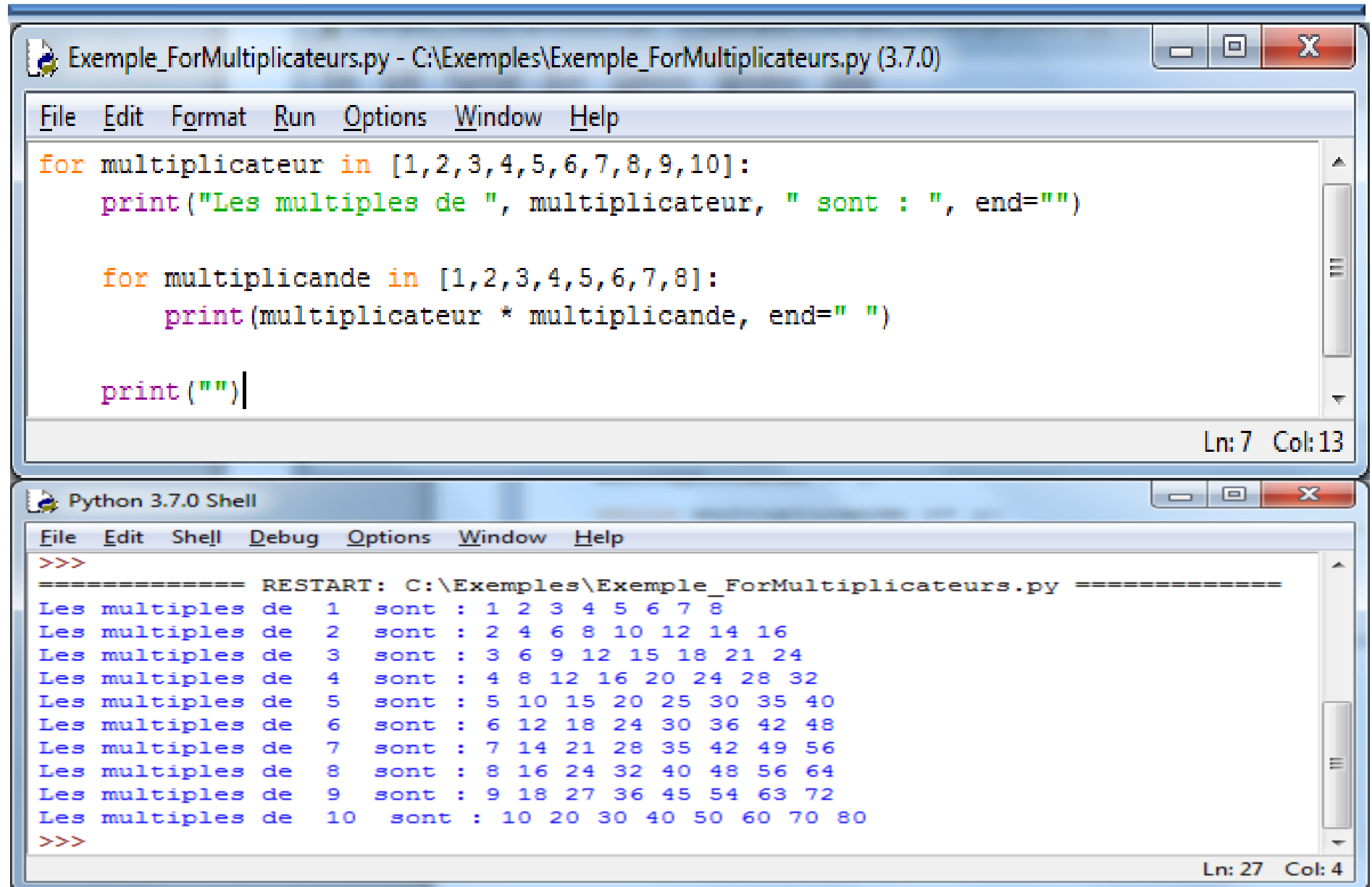
Une *liste* est un objet itérable

```
for nb in [1, 2, 3]:  
    print(nb)  
print("terminé")
```

On y reviendra !



# Example 1 : for



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exemple\_ForMultiplicateurs.py - C:\Exemples\Exemple\_ForMultiplicateurs.py (3.7.0)', contains a Python script. The script uses nested 'for' loops to calculate and print the multiples of numbers from 1 to 10. The bottom window, titled 'Python 3.7.0 Shell', shows the output of the script, which is a list of lines for each multiplier, showing its multiples up to 80. The status bars of both windows indicate the current line and column numbers.

```
File Edit Format Run Options Window Help
for multiplicateur in [1,2,3,4,5,6,7,8,9,10]:
    print("Les multiples de ", multiplicateur, " sont : ", end="")

    for multiplicande in [1,2,3,4,5,6,7,8]:
        print(multiplicateur * multiplicande, end=" ")

    print("")|
Ln: 7 Col: 13
```

```
File Edit Shell Debug Options Window Help
>>>
===== RESTART: C:\Exemples\Exemple_ForMultiplicateurs.py =====
Les multiples de 1 sont : 1 2 3 4 5 6 7 8
Les multiples de 2 sont : 2 4 6 8 10 12 14 16
Les multiples de 3 sont : 3 6 9 12 15 18 21 24
Les multiples de 4 sont : 4 8 12 16 20 24 28 32
Les multiples de 5 sont : 5 10 15 20 25 30 35 40
Les multiples de 6 sont : 6 12 18 24 30 36 42 48
Les multiples de 7 sont : 7 14 21 28 35 42 49 56
Les multiples de 8 sont : 8 16 24 32 40 48 56 64
Les multiples de 9 sont : 9 18 27 36 45 54 63 72
Les multiples de 10 sont : 10 20 30 40 50 60 70 80
>>>
Ln: 27 Col: 4
```

# La boucle for

`range([start], stop[, step])`

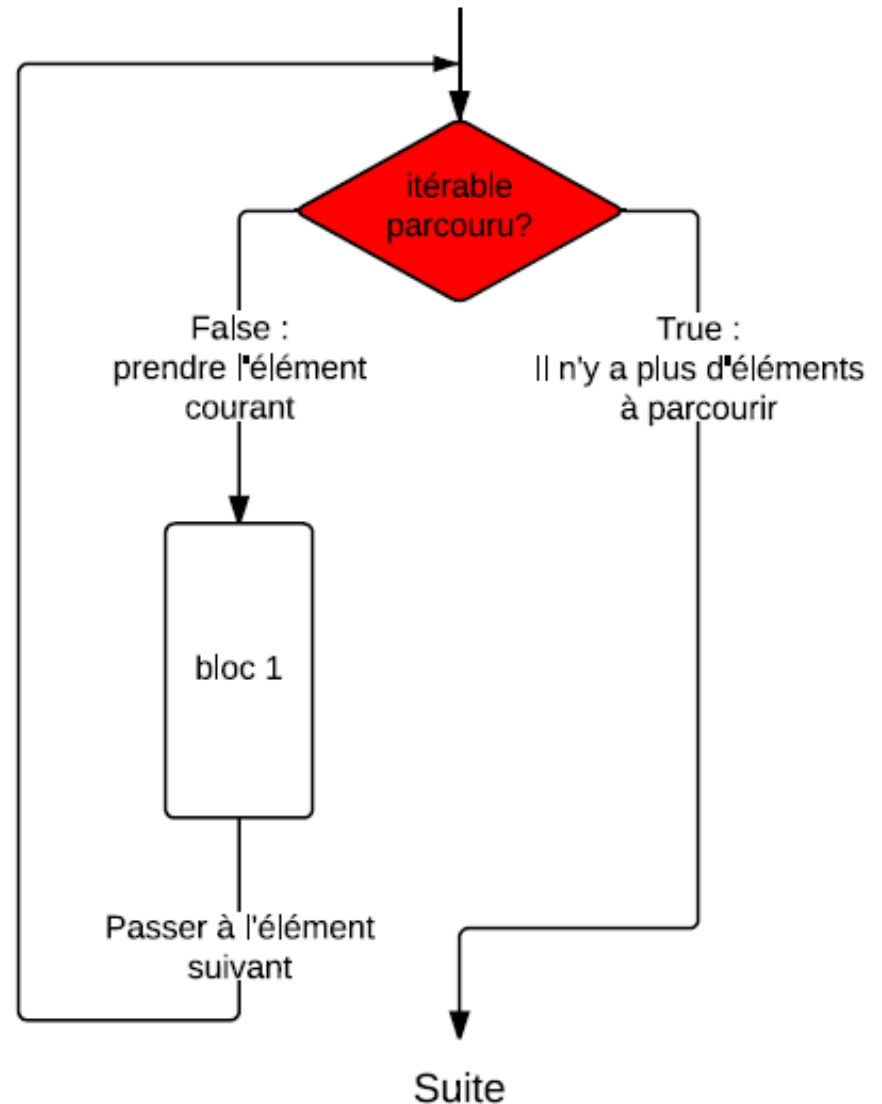
Par défaut:

`start = 0`

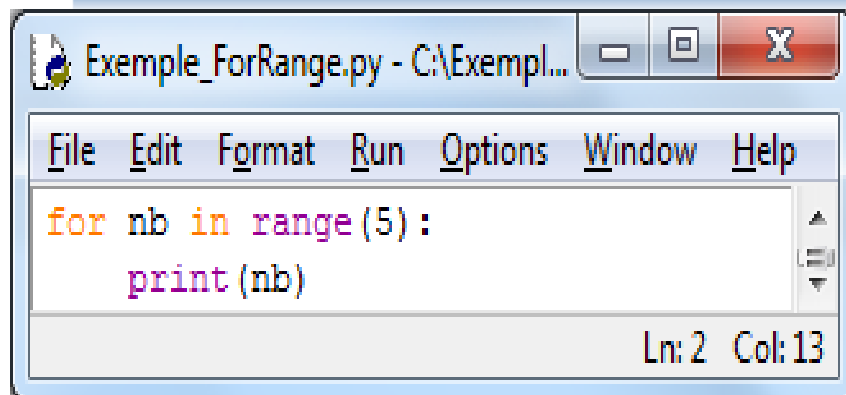
`step = 1`

```
for nb in range(5):  
    print(nb)
```

On y reviendra !



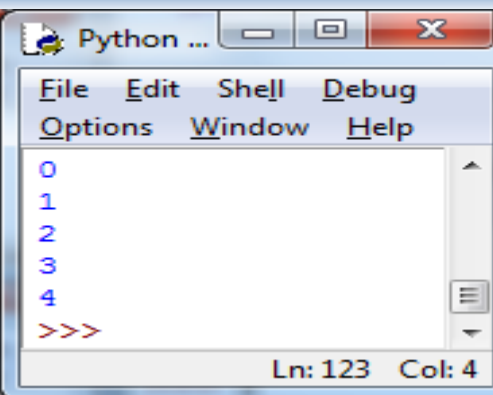
# Example 2: for



Example\_ForRange.py - C:\Exempl...

```
File Edit Format Run Options Window Help
for nb in range(5):
    print(nb)
```

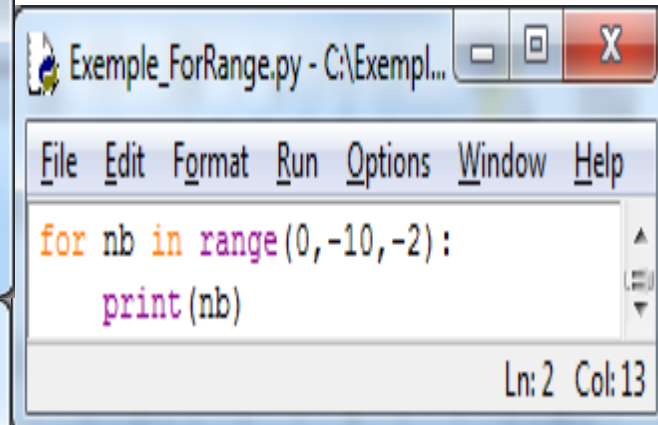
Ln: 2 Col: 13



Python ...

```
File Edit Shell Debug
Options Window Help
0
1
2
3
4
>>>
```

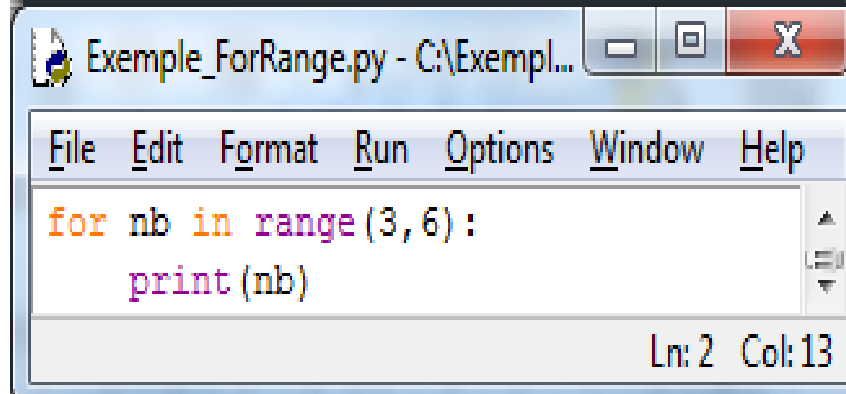
Ln: 123 Col: 4



Example\_ForRange.py - C:\Exempl...

```
File Edit Format Run Options Window Help
for nb in range(0, -10, -2):
    print(nb)
```

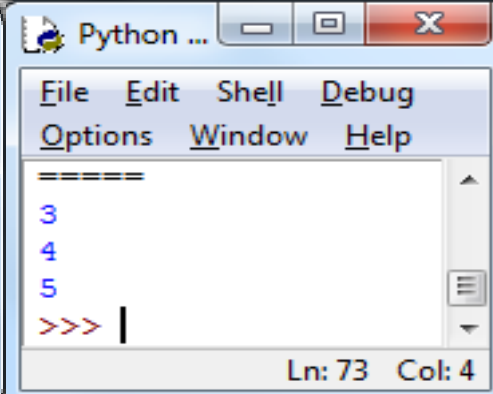
Ln: 2 Col: 13



Example\_ForRange.py - C:\Exempl...

```
File Edit Format Run Options Window Help
for nb in range(3, 6):
    print(nb)
```

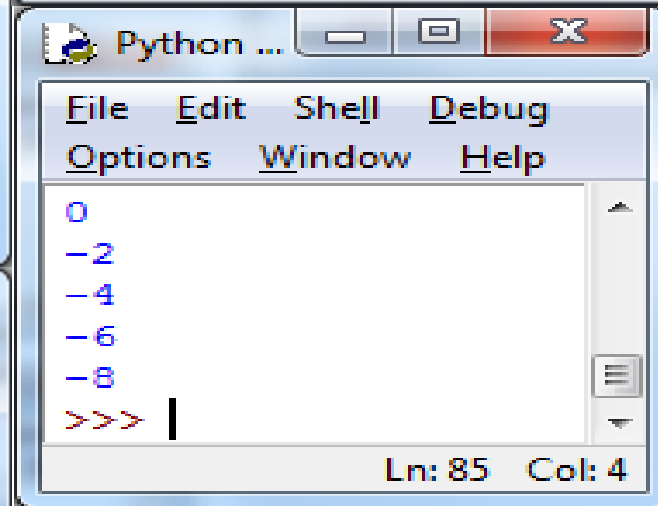
Ln: 2 Col: 13



Python ...

```
File Edit Shell Debug
Options Window Help
=====
3
4
5
>>> |
```

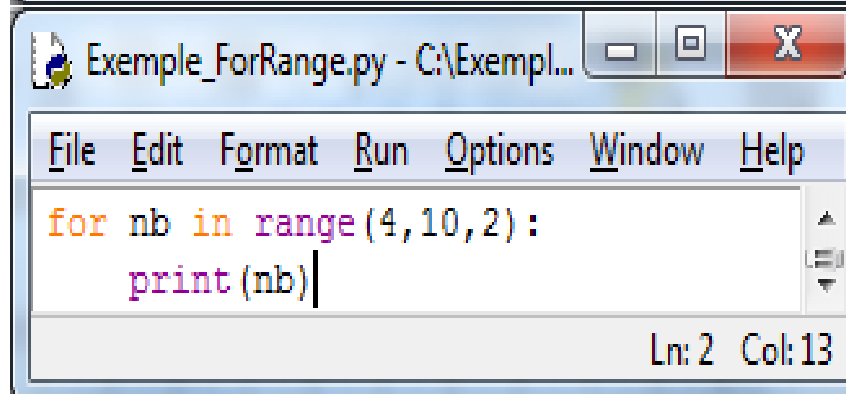
Ln: 73 Col: 4



Python ...

```
File Edit Shell Debug
Options Window Help
0
-2
-4
-6
-8
>>> |
```

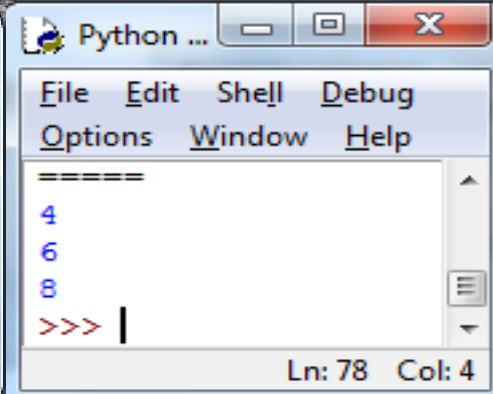
Ln: 85 Col: 4



Example\_ForRange.py - C:\Exempl...

```
File Edit Format Run Options Window Help
for nb in range(4, 10, 2):
    print(nb)
```

Ln: 2 Col: 13

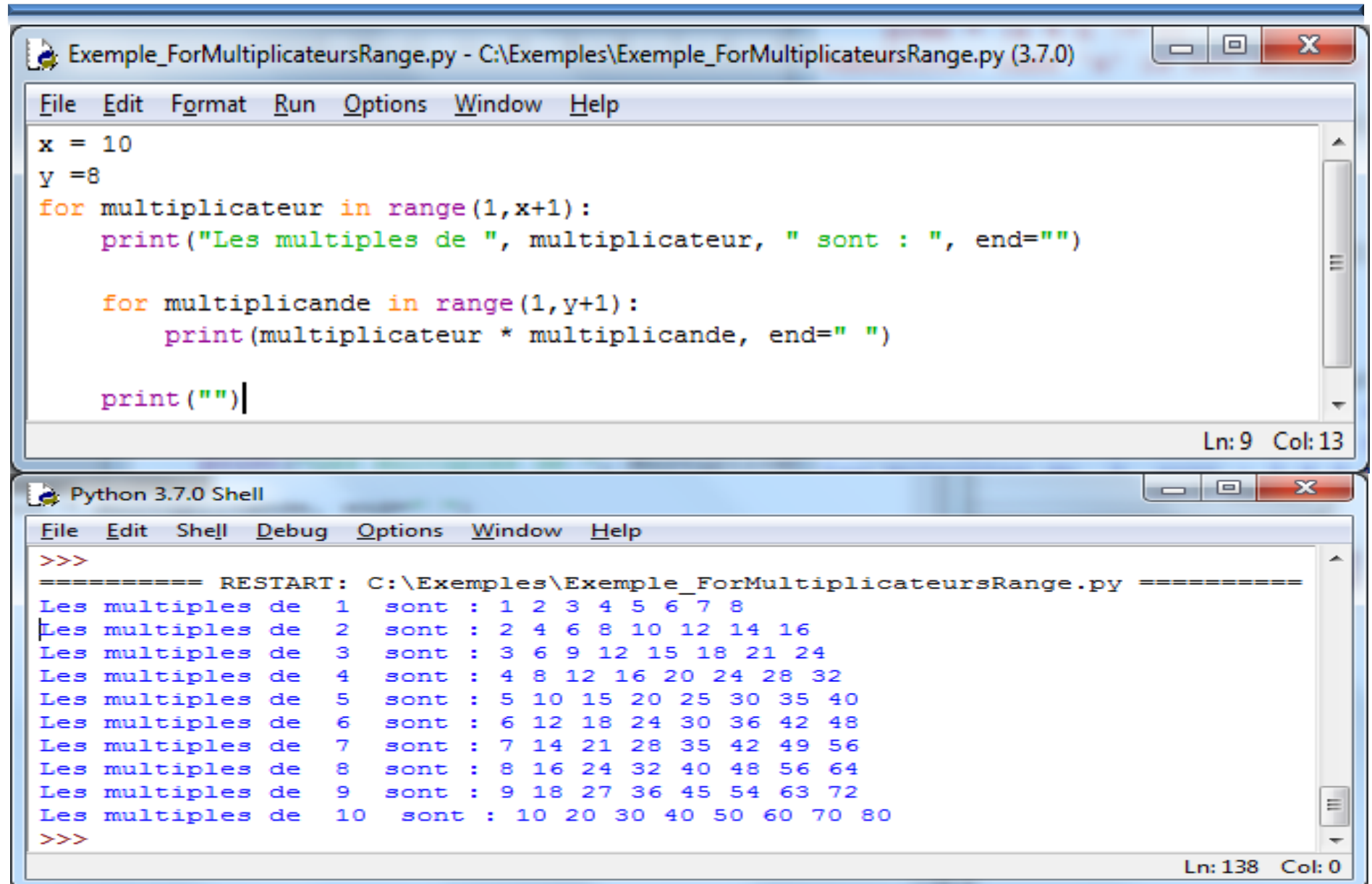


Python ...

```
File Edit Shell Debug
Options Window Help
=====
4
6
8
>>> |
```

Ln: 78 Col: 4

# Example 3 : for



The image shows a screenshot of a Python IDE with two windows. The top window, titled 'Exemple\_ForMultiplicateursRange.py - C:\Exemples\Exemple\_ForMultiplicateursRange.py (3.7.0)', contains the following Python code:

```
File Edit Format Run Options Window Help
x = 10
y = 8
for multiplicateur in range(1,x+1):
    print("Les multiples de ", multiplicateur, " sont : ", end="")

    for multiplicande in range(1,y+1):
        print(multiplicateur * multiplicande, end=" ")

    print("")
```

The status bar at the bottom right of this window indicates 'Ln: 9 Col: 13'.

The bottom window, titled 'Python 3.7.0 Shell', shows the output of the script after execution. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The output is as follows:

```
>>>
===== RESTART: C:\Exemples\Exemple_ForMultiplicateursRange.py =====
Les multiples de 1 sont : 1 2 3 4 5 6 7 8
Les multiples de 2 sont : 2 4 6 8 10 12 14 16
Les multiples de 3 sont : 3 6 9 12 15 18 21 24
Les multiples de 4 sont : 4 8 12 16 20 24 28 32
Les multiples de 5 sont : 5 10 15 20 25 30 35 40
Les multiples de 6 sont : 6 12 18 24 30 36 42 48
Les multiples de 7 sont : 7 14 21 28 35 42 49 56
Les multiples de 8 sont : 8 16 24 32 40 48 56 64
Les multiples de 9 sont : 9 18 27 36 45 54 63 72
Les multiples de 10 sont : 10 20 30 40 50 60 70 80
>>>
```

The status bar at the bottom right of this window indicates 'Ln: 138 Col: 0'.