

PARADIGMES DE PROGRAMMATION

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

Module 3

Mondher Bouden



2025-2026

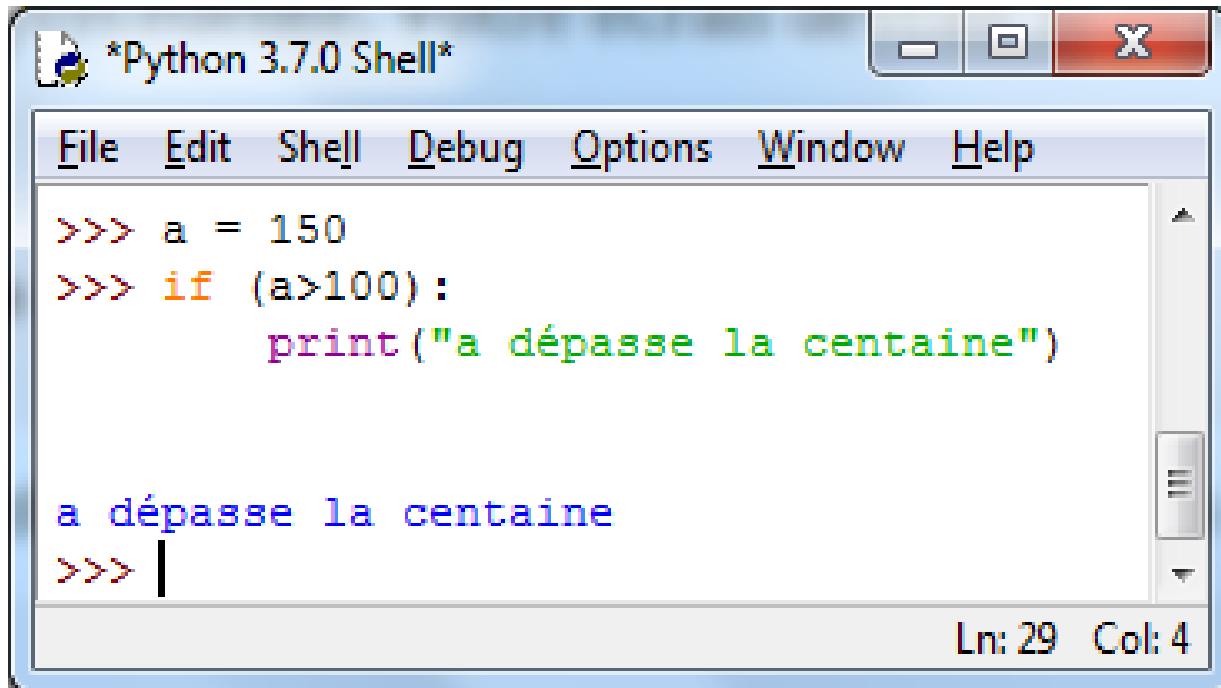
Les structures de contrôle

(Instructions conditionnelles vs répétitives)

Les instructions conditionnelles

Structure *si*

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



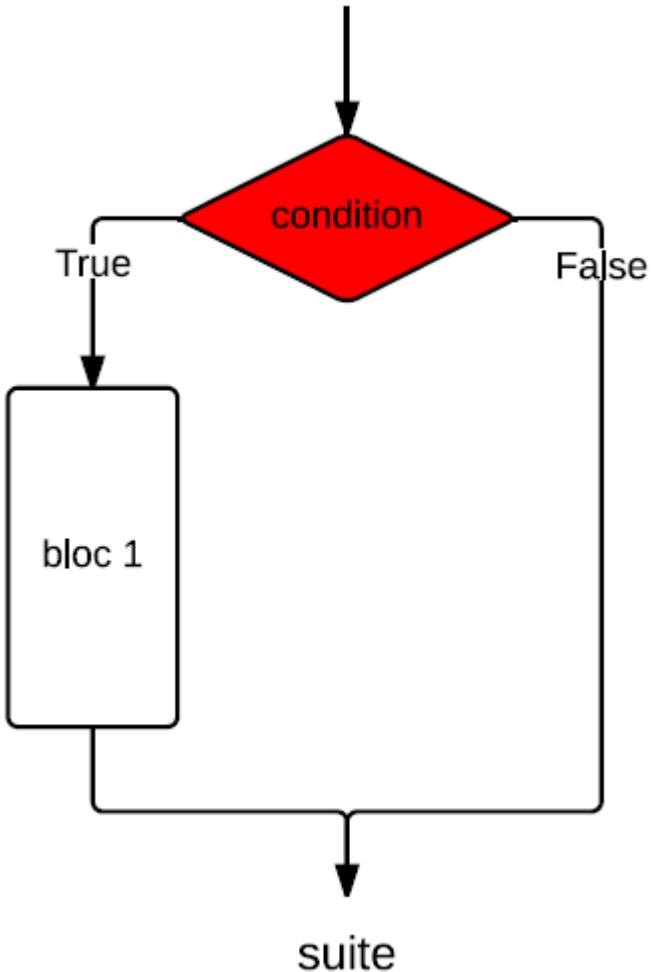
The screenshot shows a Python 3.7.0 Shell window. The code entered is:

```
>>> a = 150
>>> if (a>100):
    print("a dépasse la centaine")
```

The output is:

```
a dépasse la centaine
>>>
```

At the bottom, it says "Ln: 29 Col: 4".



Les instructions conditionnelles

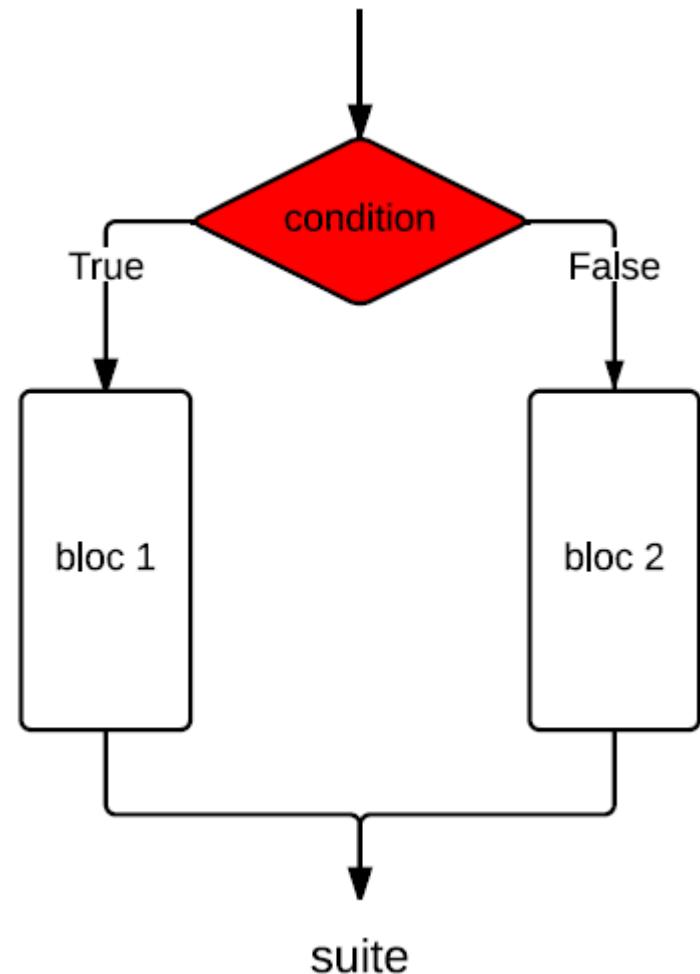
Structure *si - sinon*

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

The screenshot shows a Python 3.7.0 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The code input area contains:

```
>>> a = 20
>>> if (a > 100):
    print("a dépasse la centaine")
else:
    print("a ne dépasse pas cent")
a ne dépasse pas cent
>>>
```

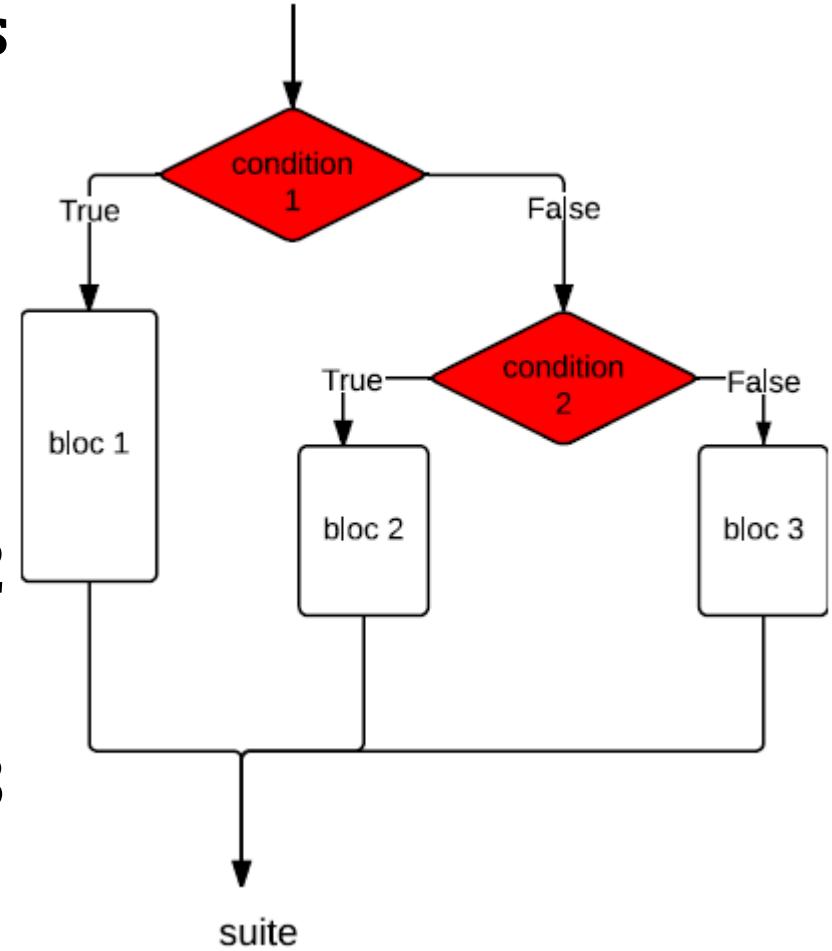
The status bar at the bottom indicates Ln: 95 Col: 4.



Les instructions conditionnelles

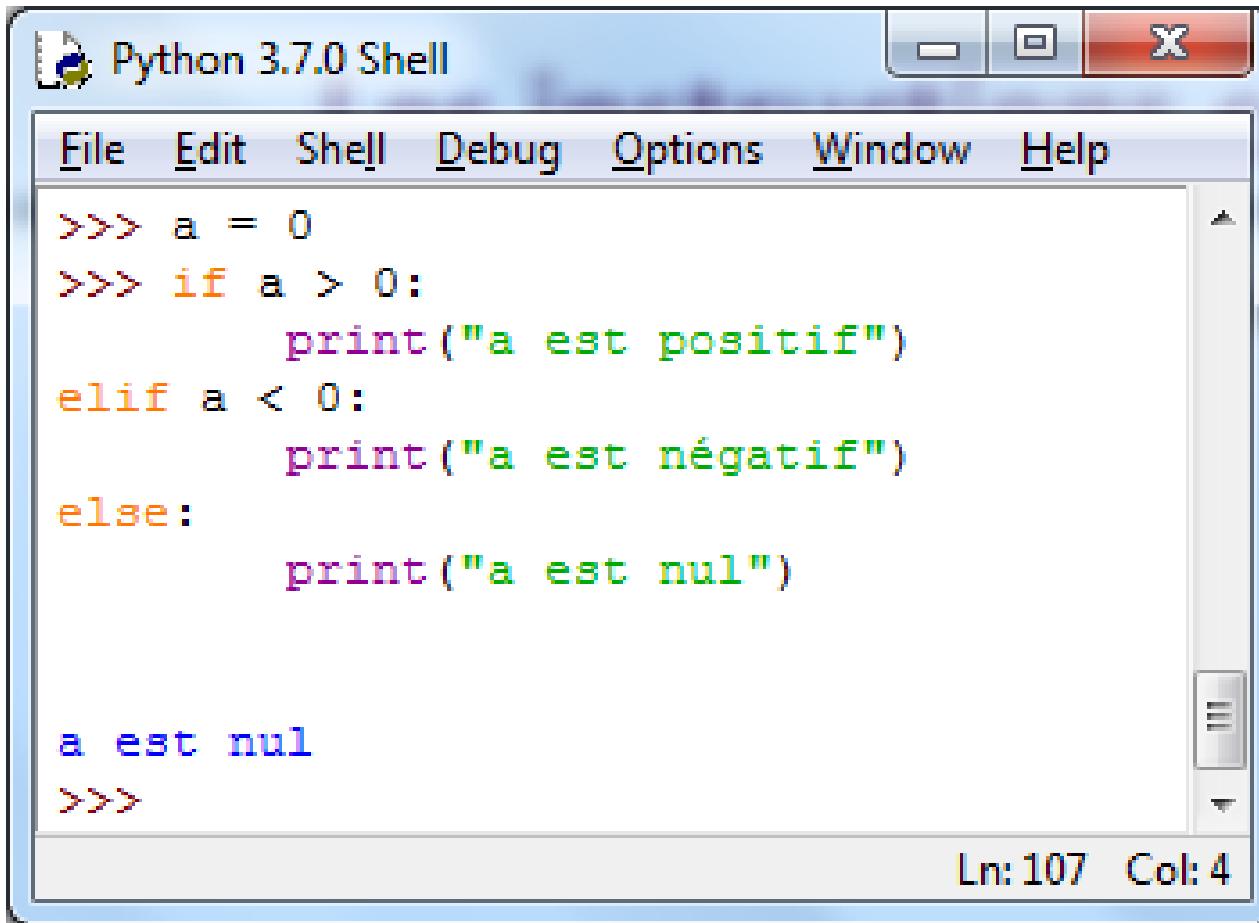
Structure *si-sinon* imbriquées

- 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 >>) :



The screenshot shows the Python 3.7.0 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The code input area contains the following Python script:

```
>>> a = 0
>>> if a > 0:
        print("a est positif")
elif a < 0:
        print("a est négatif")
else:
        print("a est nul")
```

The output window displays the result of the script execution:

```
a est nul
>>>
```

The status bar at the bottom right indicates "Ln: 107 Col: 4".

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

Exemple 1

Exempleif_agerafraite.py - C:/Exemples/Exempleif_agerafraite.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")
```

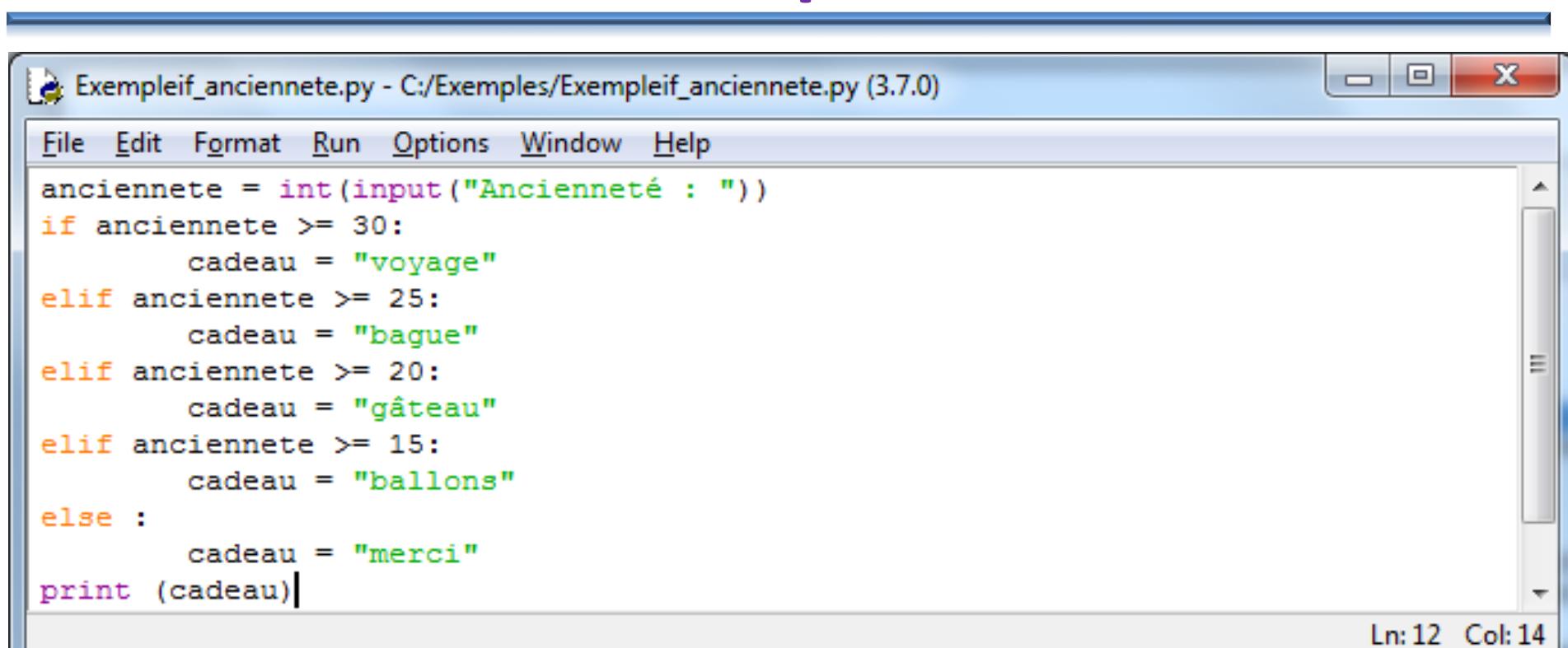
Ln: 9 Col: 57

Python 3.7.0 Shell

```
File Edit Shell Debug Options Window Help
=====
RESTART: C:/Exemples/Exempleif_agerafraite.py =====
Âge à l'embauche : 2002
Âge à la retraite : 2023
années de travail : 21
>>>
```

Ln: 117 Col: 0

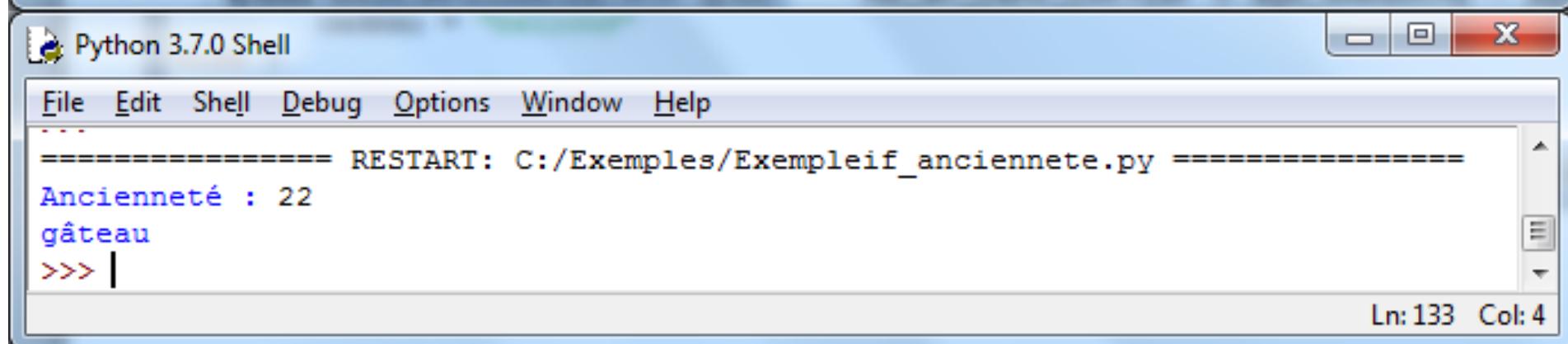
Exemple 2



The screenshot shows a Python code editor window titled "Exempleif_anciennete.py - C:/Exemples/Exempleif_anciennete.py (3.7.0)". The code displays a nested if-elif-else statement that prints a gift based on age. The code is as follows:

```
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)
```

The status bar at the bottom right indicates "Ln: 12 Col: 14".



The screenshot shows a Python shell window titled "Python 3.7.0 Shell". It displays the output of running the script. The command "python Exempleif_anciennete.py" was run, followed by the input "Ancienneté : 22", and the output "gâteau". The status bar at the bottom right indicates "Ln: 133 Col: 4".

```
=====
===== RESTART: C:/Exemples/Exempleif_anciennete.py =====
Ancienneté : 22
gâteau
>>> |
```

The status bar at the bottom right indicates "Ln: 133 Col: 4".

Exemple 3

The screenshot shows a Windows desktop environment with two open windows:

- Code Editor Window:** The title bar reads "Exempleif_pair.py - C:/Exemples/Exempleif_pair.py (3.7.0)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor window is:

```
# 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")
```

The status bar at the bottom right indicates "Ln: 7 Col: 25".
- Python Shell Window:** The title bar reads "Python 3.7.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The output from running the script is:

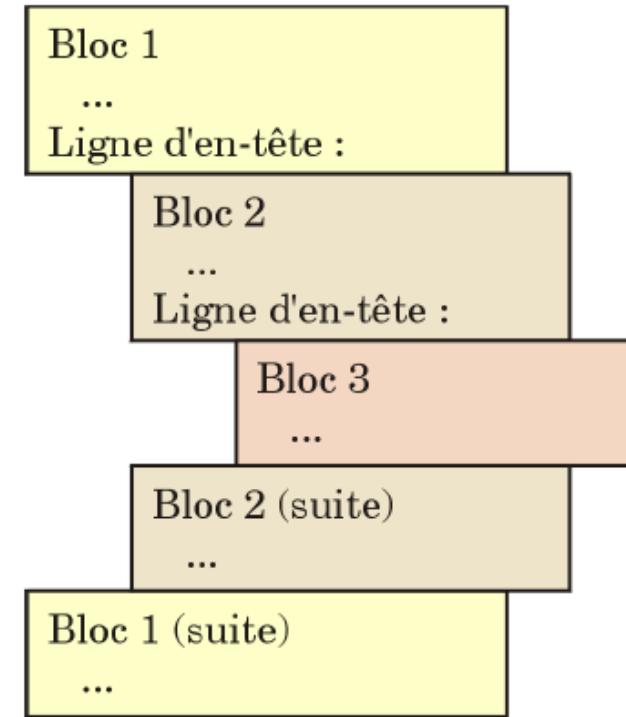
```
=====
===== RESTART: C:/Exemples/Exempleif_pair.py =====
a est pair
parce que le reste de sa division par 2 est nul
>>>
```

The status bar at the bottom right indicates "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

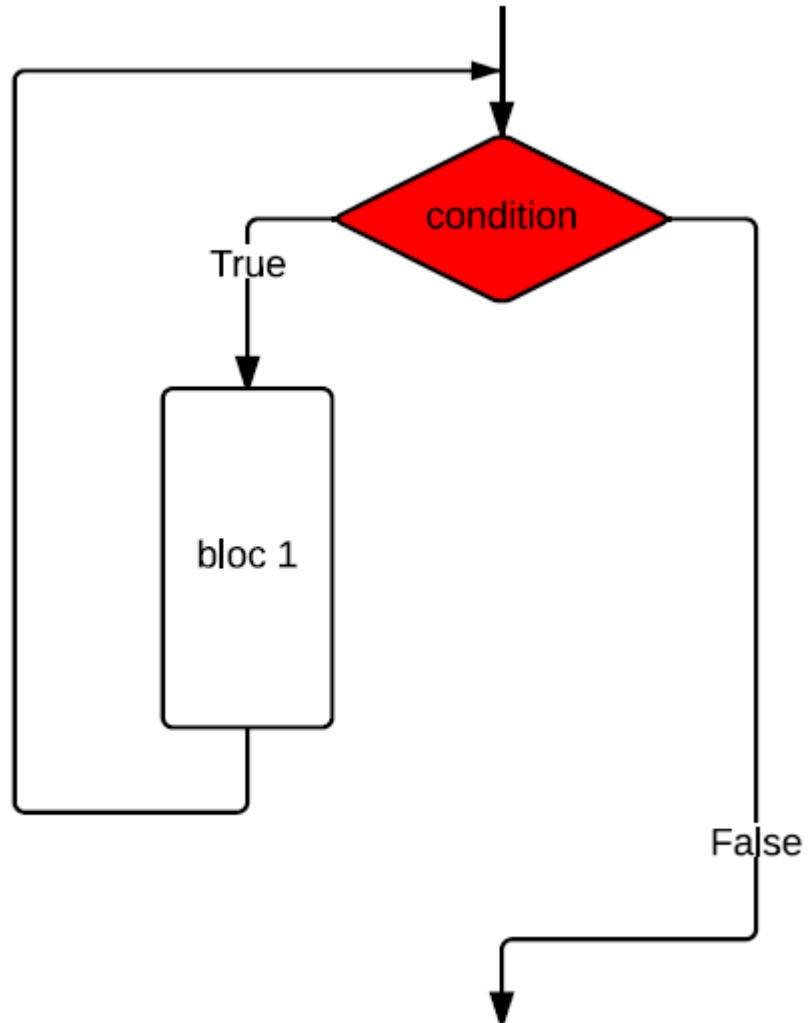
- Tant que condition :
 - ▶ On exécute le bloc 1
- Suite

The screenshot shows the Python 3.7.0 Shell window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The code input area contains the following Python code:

```
>>> a = 0
>>> while (a < 7):
...     a = a + 1
...     print(a)

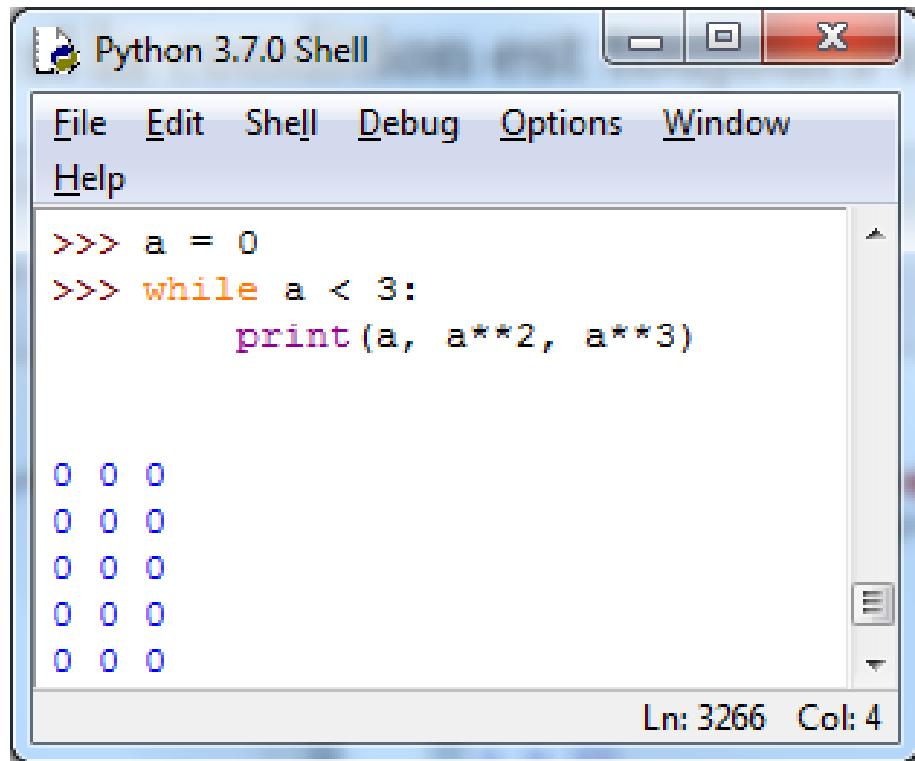
1
2
3
4
5
6
7
>>>
```

The status bar at the bottom indicates Ln: 153 Col: 4.



La boucle while

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

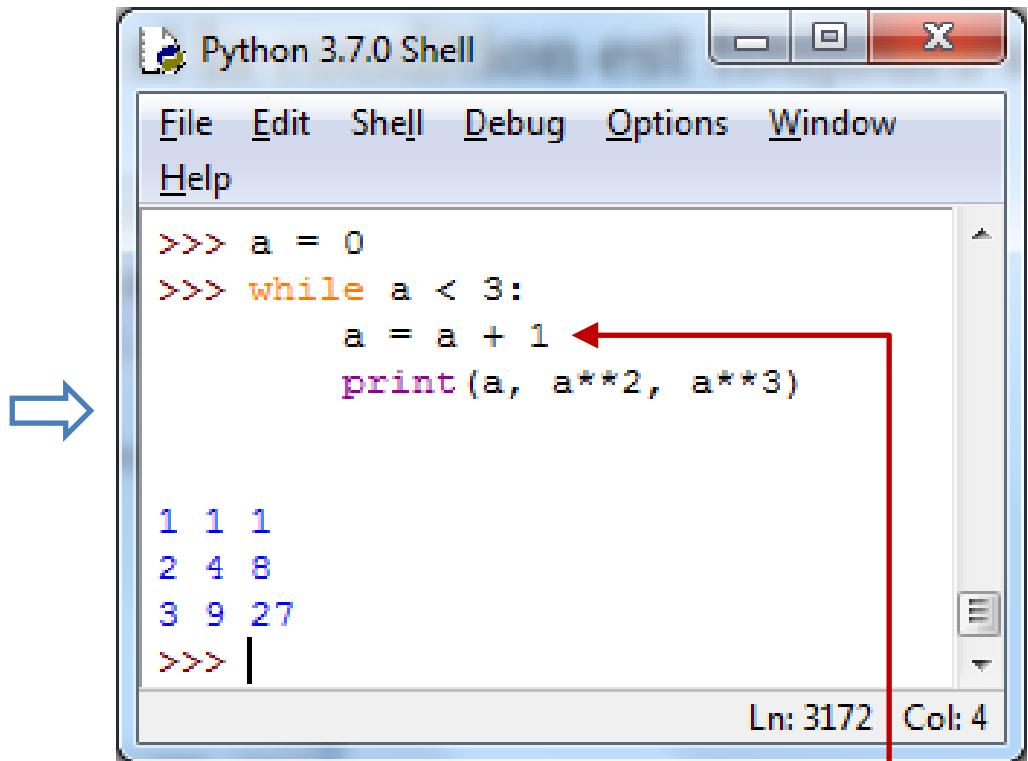


Python 3.7.0 Shell

```
>>> 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

```
>>> 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.

Incrémantation
(ou $a += 1$)

Exemple 1

The screenshot shows two windows from a Python IDE. The top window is titled "Exemplewhile_fibonacci.py - C:/Exemples/Exemplewhile_fibonacci.py (3.7.0)". It contains the following Python code:

```
# 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
```

The bottom window is titled "Python 3.7.0 Shell". It shows the output of running the script:

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

- 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

$x^y = x \cdot x \cdot x \cdot \dots \cdot x$ ($y - 1$ multiplications)

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

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

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

$$x^1 = x$$

$$x^0 = 1$$

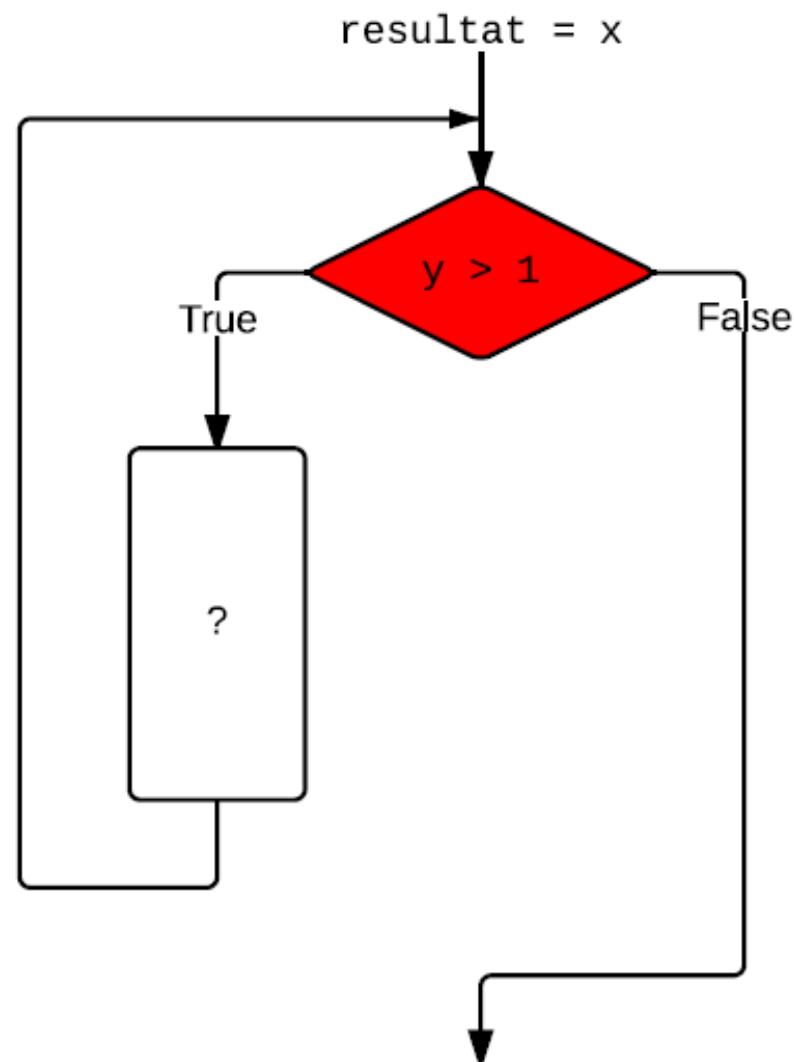
Exemple 2 : calculer x^y

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

```
resultat = x
```

```
...
```

```
print(resultat)
```



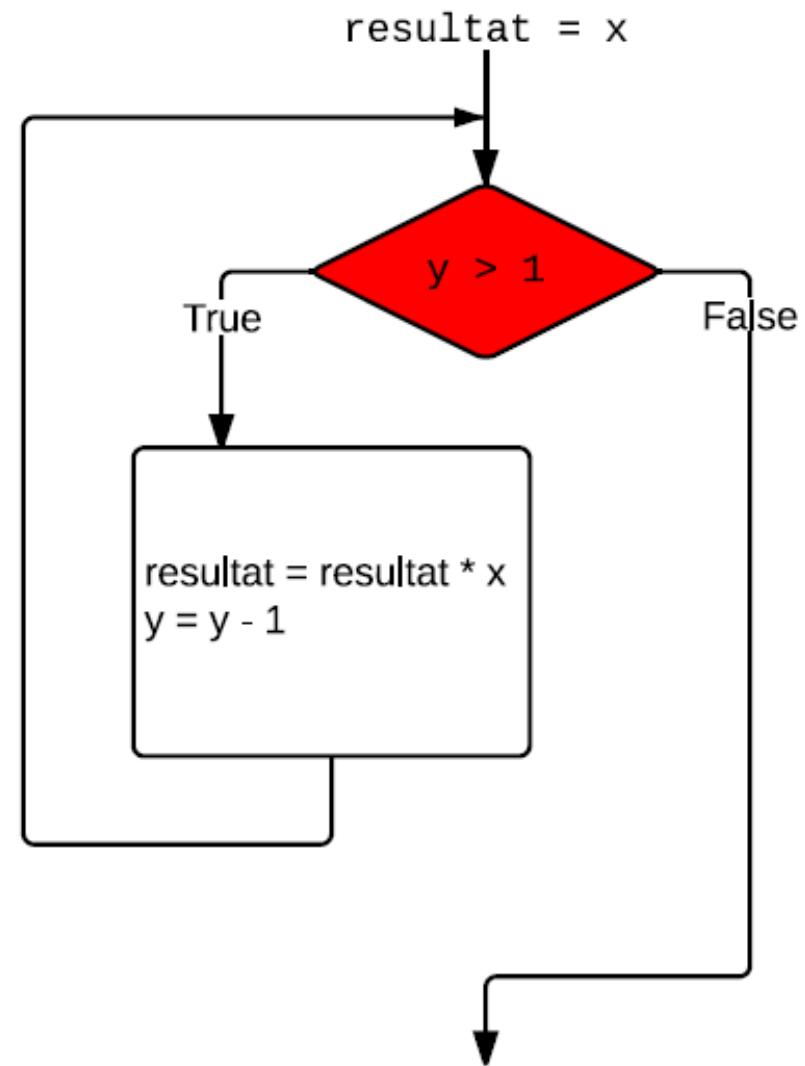
Exemple 2 : calculer x^y

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

```
resultat = x

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

print(resultat)
```

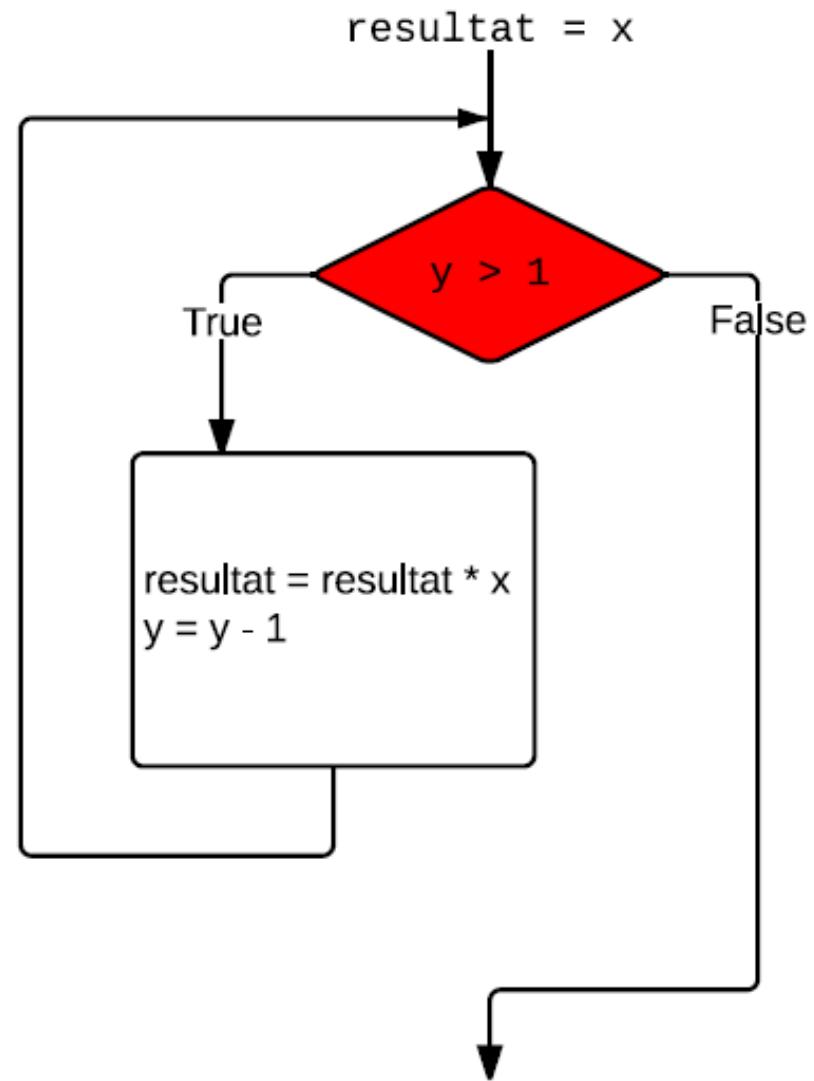


Exemple 2 : calculer x^y

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

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

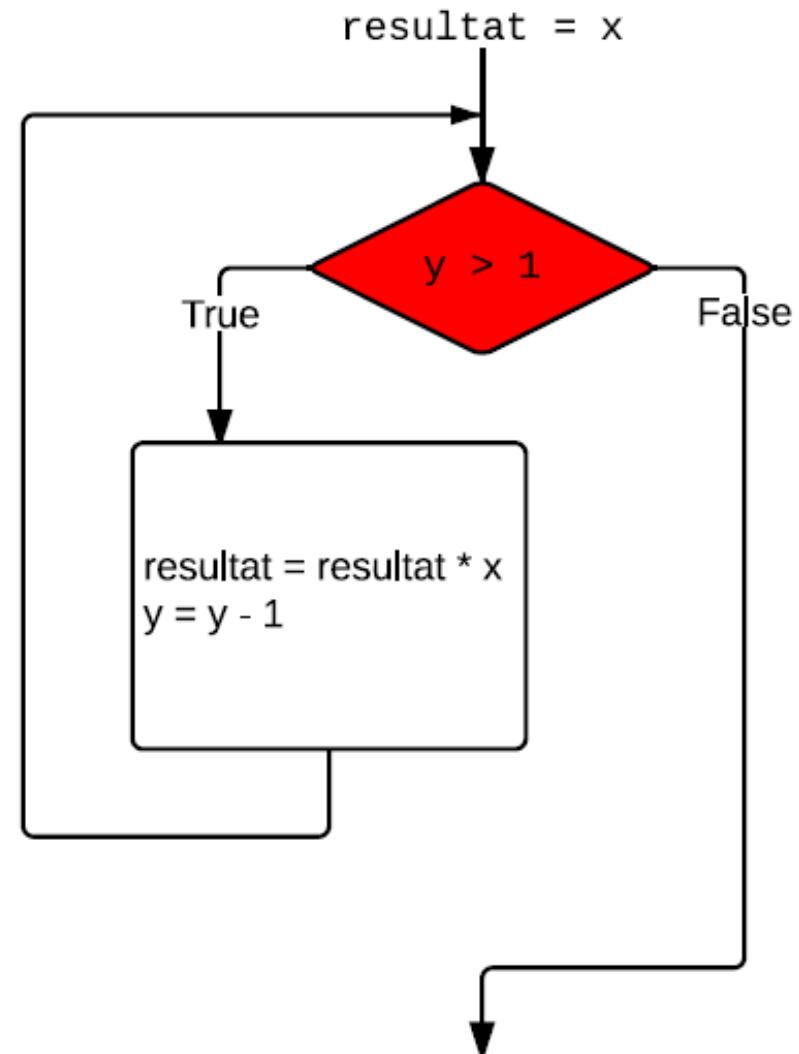
Est-ce que la boucle produit toujours le bon résultat ?



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

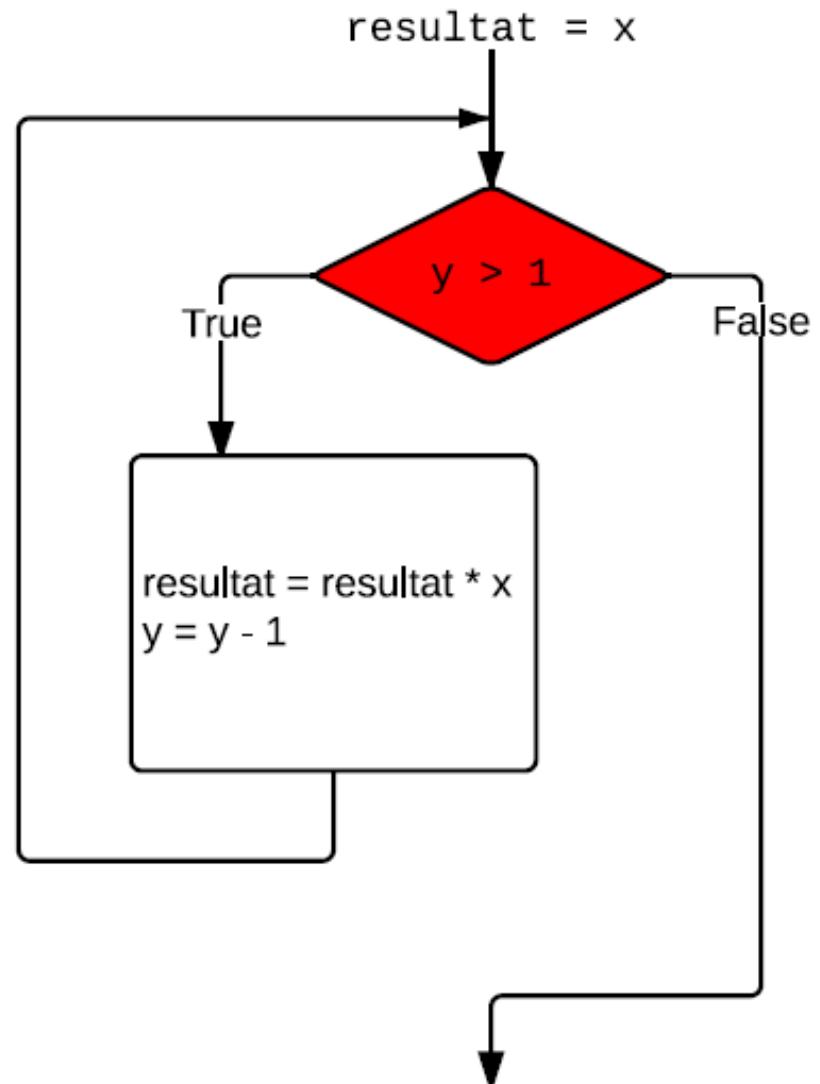
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
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

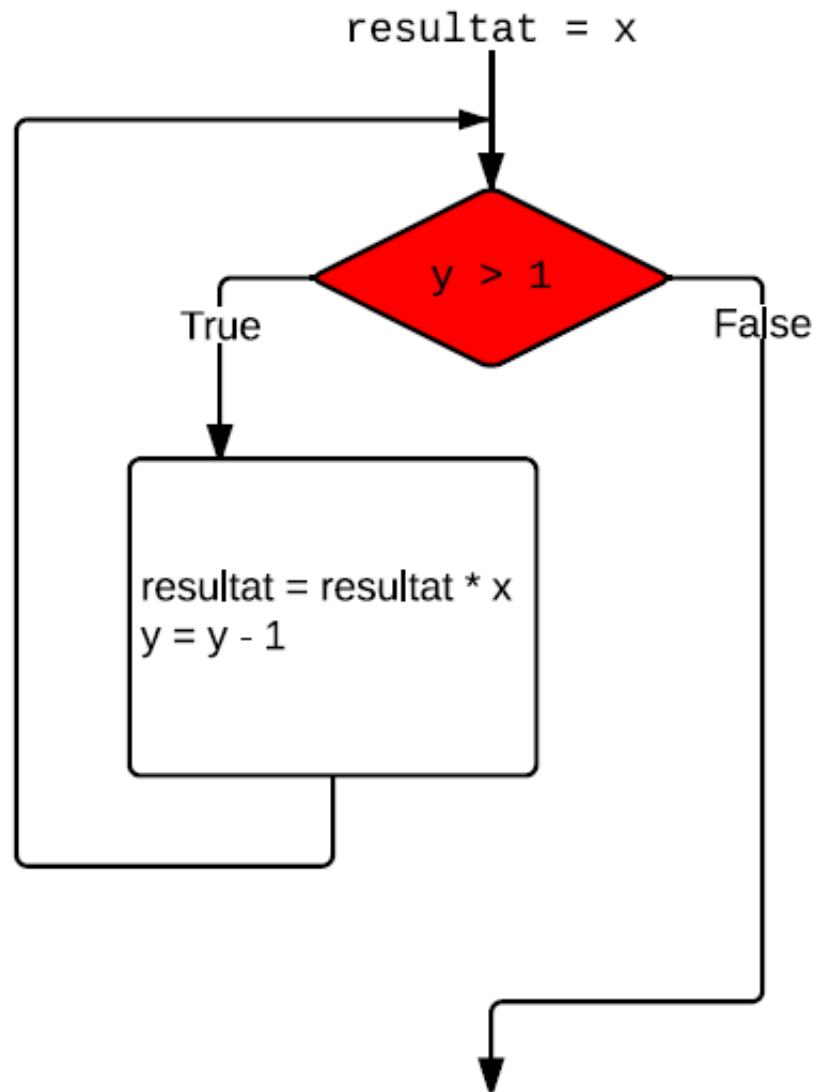
$$x^y = x \cdot x \cdot x \cdot \dots \cdot x \quad (y - 1 \text{ fois})$$

```
# On suppose y >= 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 screenshot shows a Python IDE window titled "Exemplewhile_xpuissancey.py - C:/Exemples/Exemplewhile_xpuissancey.py (3.7.0)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor window is:

```
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 status bar at the bottom right indicates "Ln: 4 Col: 0".

The screenshot shows the Python 3.7.0 Shell window titled "Python 3.7.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell output shows the execution of the script:

```
=====
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
>>>
```

The status bar at the bottom right indicates "Ln: 8 Col: 4".

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	$x + 2$	$x + 3$	$x + 4$	$x + 5$...	$x + y$

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 >= 1 et y >= 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 >= 1 et y >= 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("")
```

on passe à la
ligne suivante
du tableau

Exemple 3 : Boucles while imbriquées

The screenshot shows a Python development environment with two windows. The top window is a code editor titled "Exemple_WhileMultiplicateurs.py" containing Python code. The bottom window is a terminal titled "Python 3.7.0 Shell" showing the execution and output of the code.

Code Editor Content:

```
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
```

Terminal Output:

```
=====
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
>>>
```

Simuler la boucle do..while (Ø en Python)

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

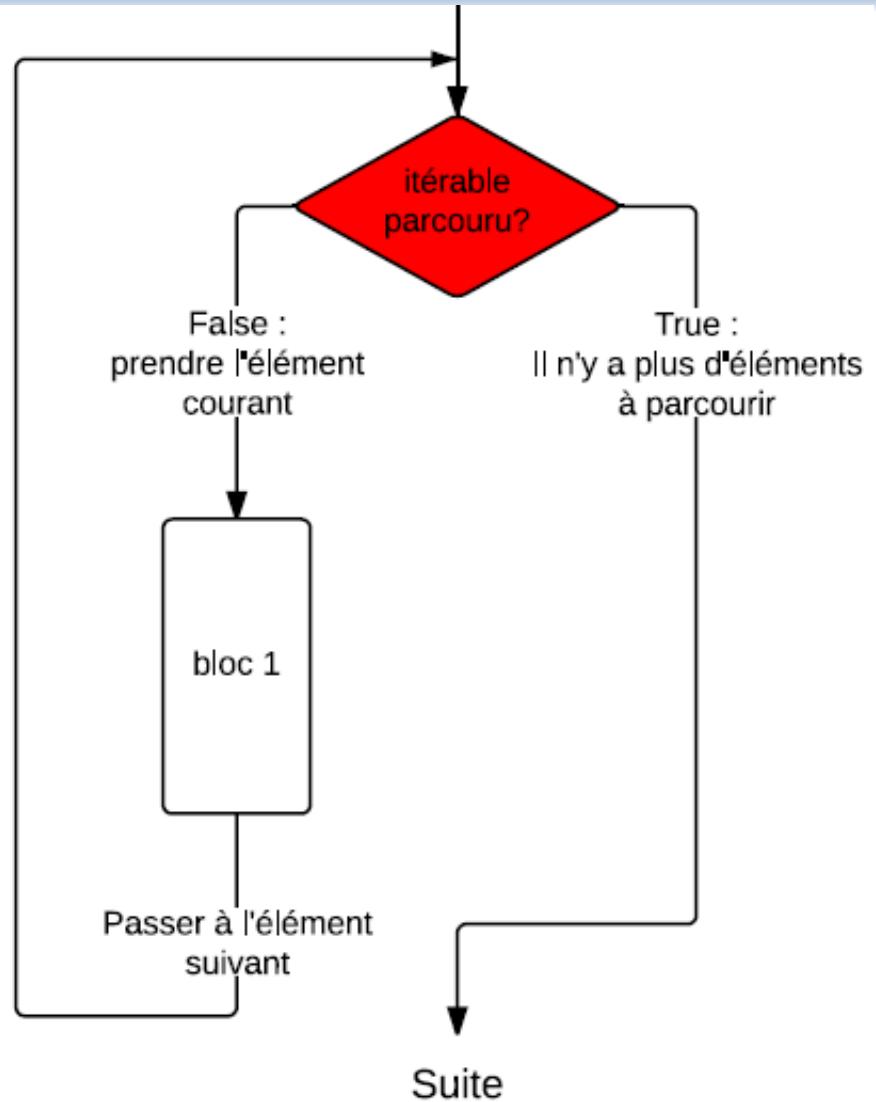
La boucle for

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

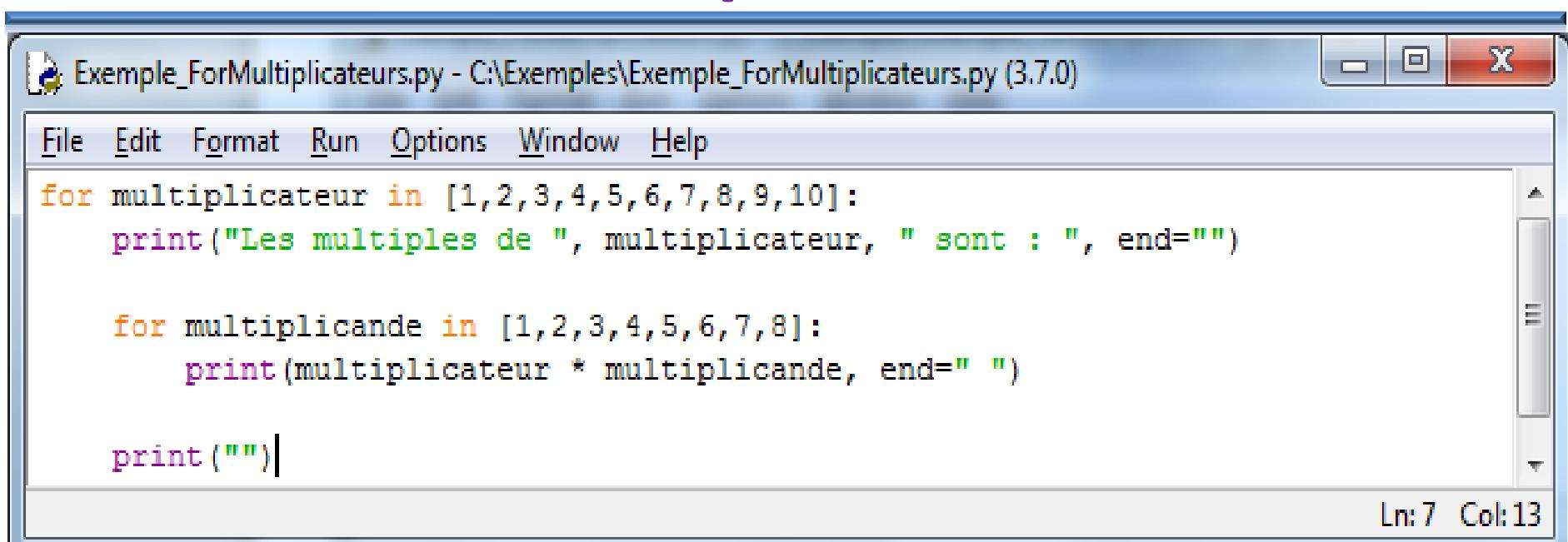
Une *liste* est un objet itérable

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

On y reviendra !



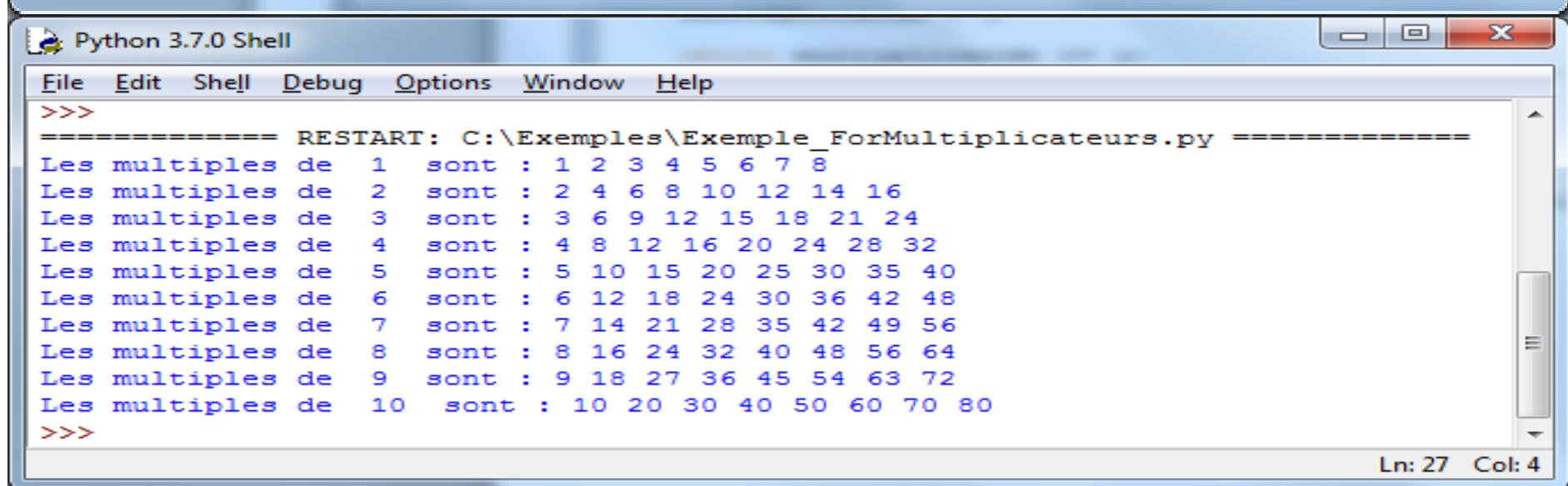
Exemple 1 : for



The screenshot shows a Python code editor window titled "Exemple_ForMultiplicateurs.py - C:\Exemples\Exemple_ForMultiplicateurs.py (3.7.0)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is as follows:

```
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("")
```

The status bar at the bottom right indicates "Ln: 7 Col: 13".



The screenshot shows the Python 3.7.0 Shell window. The command prompt shows "Python 3.7.0 Shell". The history starts with "File Edit Shell Debug Options Window Help" and then "">>>>". The output shows the results of running the script:

```
>>>  
===== 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  
>>>
```

The status bar at the bottom right indicates "Ln: 27 Col: 4".

La boucle for

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

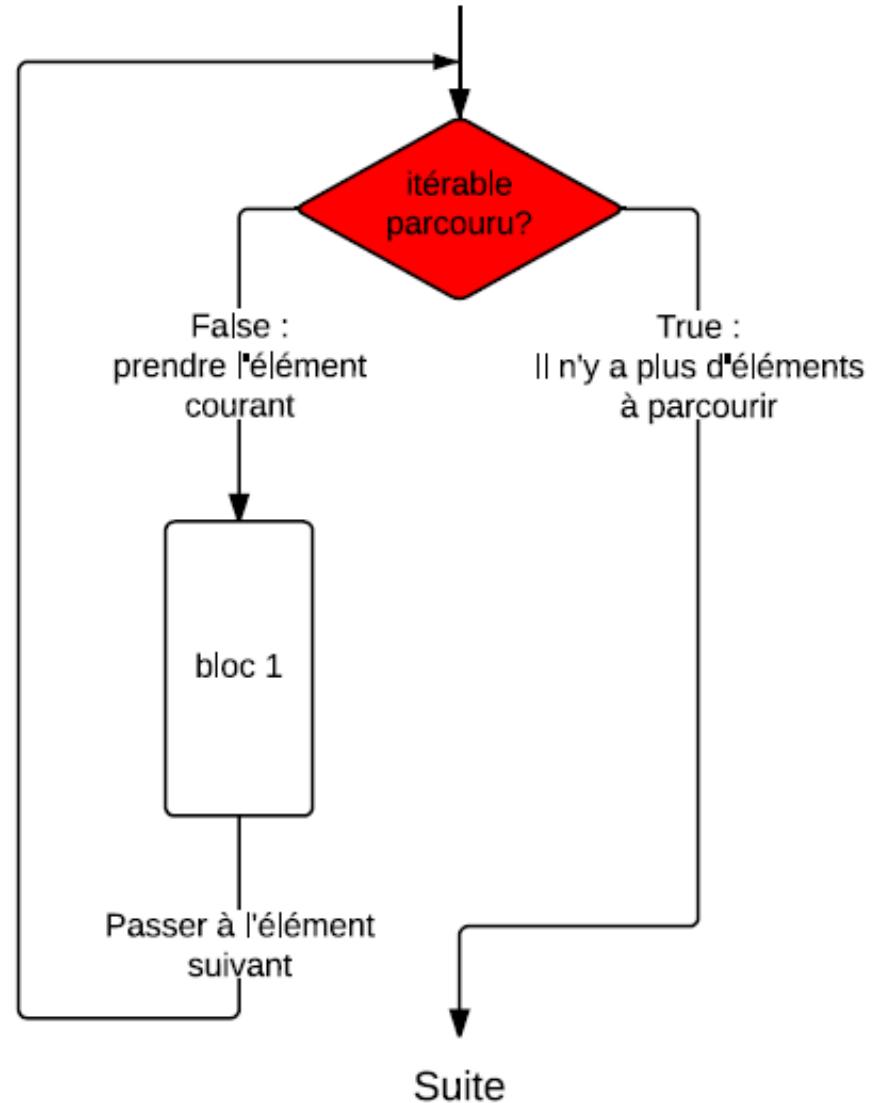
Par défaut:

`start = 0`

`step = 1`

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

On y reviendra !



Exemple 2: for

The image displays six windows arranged in a 3x2 grid, illustrating the use of the `range()` function in Python.

- Top Left Window:** Shows the code `for nb in range(5): print(nb)`. The output in the shell window is:

```
0
1
2
3
4
>>>
```

Ln: 2 Col: 13
- Top Middle Window:** Shows the code `for nb in range(0, -10, -2): print(nb)`. The output in the shell window is:

```
for nb in range(0, -10, -2):
    print(nb)

0
-2
-4
-6
-8
>>>
```

Ln: 2 Col: 13
- Middle Left Window:** Shows the code `for nb in range(3, 6): print(nb)`. The output in the shell window is:

```
for nb in range(3, 6):
    print(nb)

3
4
5
>>>
```

Ln: 2 Col: 13
- Middle Middle Window:** Shows the code `for nb in range(4, 10, 2): print(nb)`. The output in the shell window is:

```
for nb in range(4, 10, 2):
    print(nb)

4
6
8
>>>
```

Ln: 2 Col: 13
- Bottom Left Window:** Shows the code `for nb in range(5): print(nb)`. The output in the shell window is:

```
0
1
2
3
4
>>>
```

Ln: 123 Col: 4
- Bottom Middle Window:** Shows the code `for nb in range(0, -10, -2): print(nb)`. The output in the shell window is:

```
for nb in range(0, -10, -2):
    print(nb)

0
-2
-4
-6
-8
>>>
```

Ln: 78 Col: 4

Exemple 3 : for

The screenshot shows a Python code editor window titled "Exemple_ForMultiplicateursRange.py - C:\Exemples\Exemple_ForMultiplicateursRange.py (3.7.0)". The menu bar includes File, Edit, Format, Run, Options, Window, and Help. The code in the editor is as follows:

```
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 indicates "Ln: 9 Col: 13".

The screenshot shows a Python shell window titled "Python 3.7.0 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The shell output shows the execution of the script and its results:

```
>>>
=====
      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 indicates "Ln: 138 Col: 0".