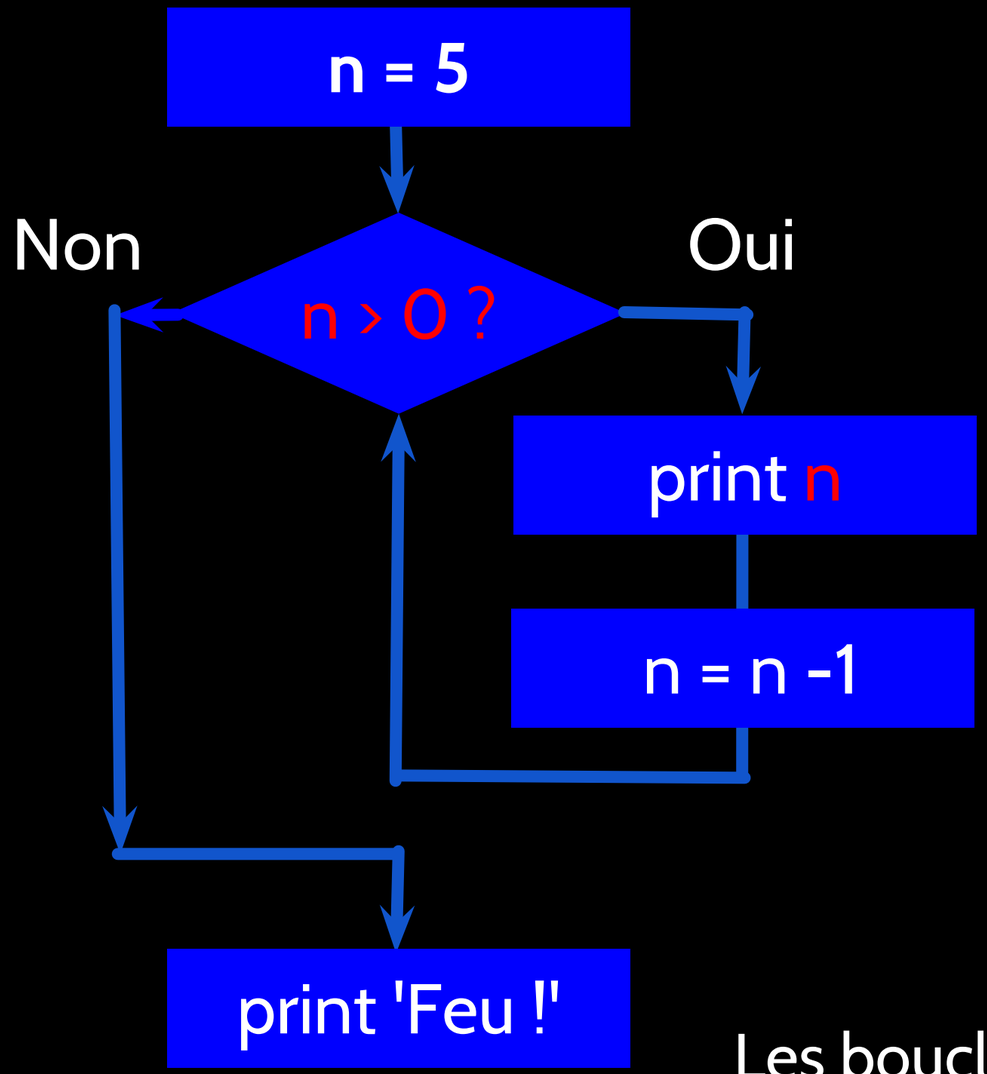
Boucles et Itérations

Chapitre 5

Python pour les sciences de l'information : Explorer les données
www.pythonlearn.com

Répétition d'opérations



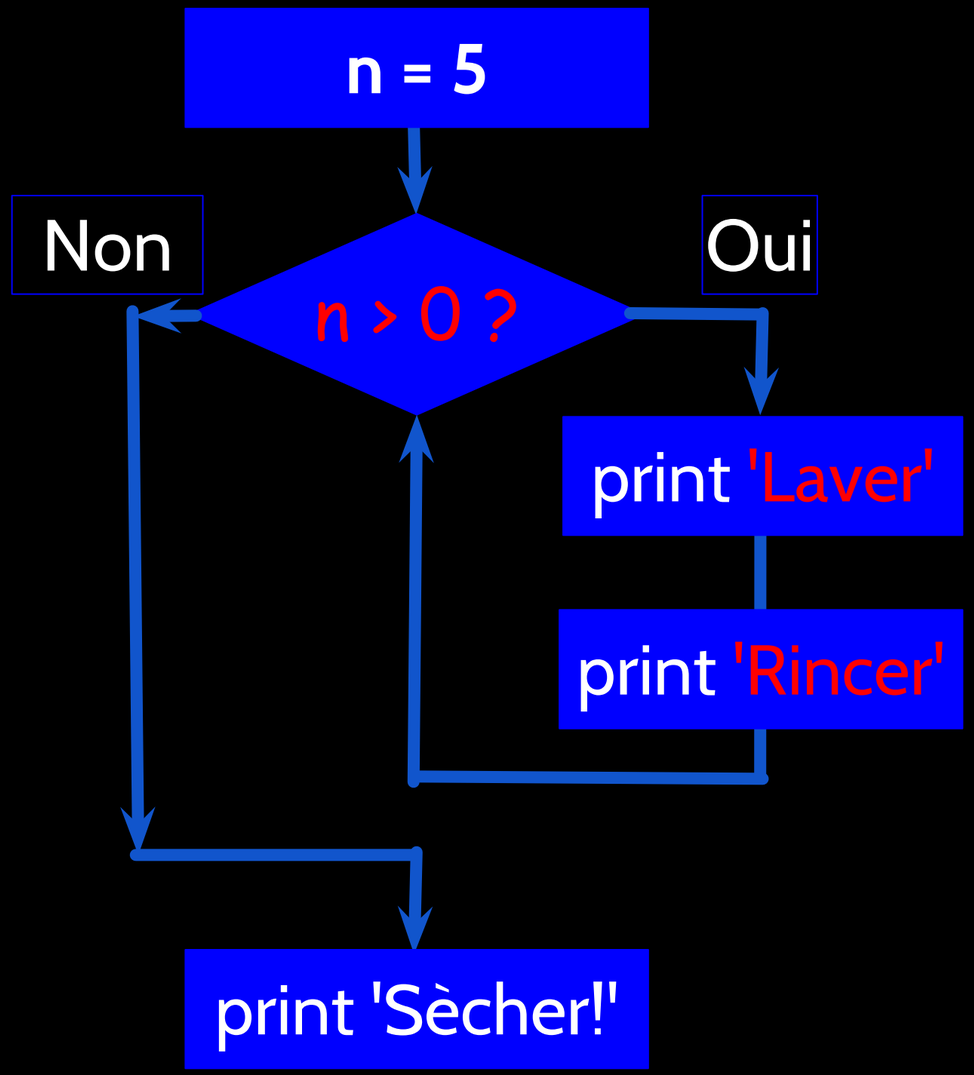
Programme:

```
n = 5
while n > 0 :
    print n
    n = n - 1
print 'Feu !'
print n
```

Affichage:

```
5
4
3
2
1
Feu !
0
```

Les boucles (répétition d'opérations) possèdent des variables d'itération dont la valeur change à chaque parcours de la boucle. Ces variables d'itération sont souvent numériques : leurs valeurs successives seront alors une séquence de nombres.

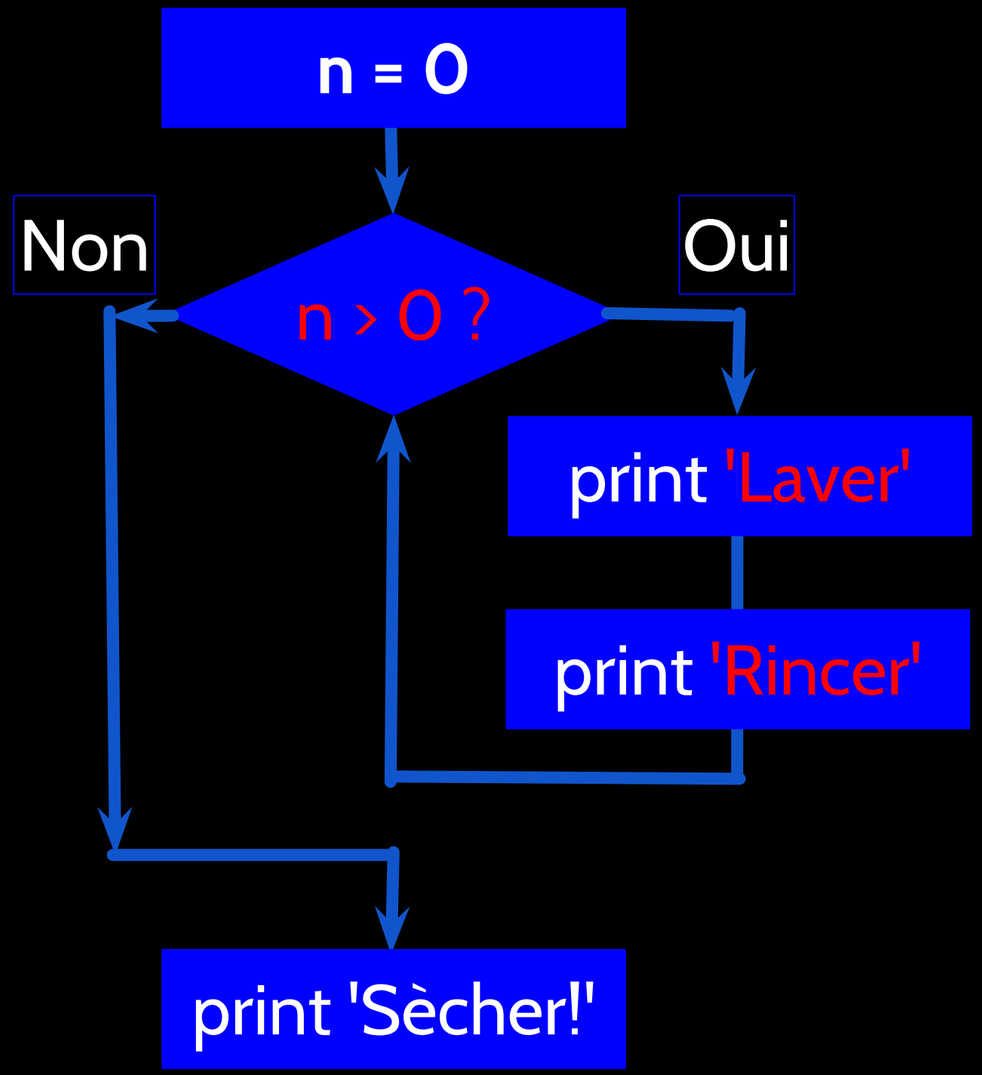


Une boucle sans fin

(on parle de boucle infinie)

```
n = 5
while n > 0 :
    print 'Laver'
    print 'Rincer'
print 'Sècher!'
```

Quel est le problème de cette boucle?



Une autre Boucle

```
n = 0
while n > 0 :
    print 'Laver'
    print 'Rincer'
print 'Sècher!'
```

Que fait cette boucle ?

S'échapper d'une boucle

- Le mot-clé break met fin à la boucle courante et réalise un saut vers l'instruction suivant immédiatement cette boucle.
- Assimilable à un test réalisable n'importe où dans le corps de la boucle.

```
while True:
    ligne = raw_input('> ')
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```

```
> Bonjour
Bonjour
> termine
termine
> fini
Fini!
```

S'échapper d'une boucle

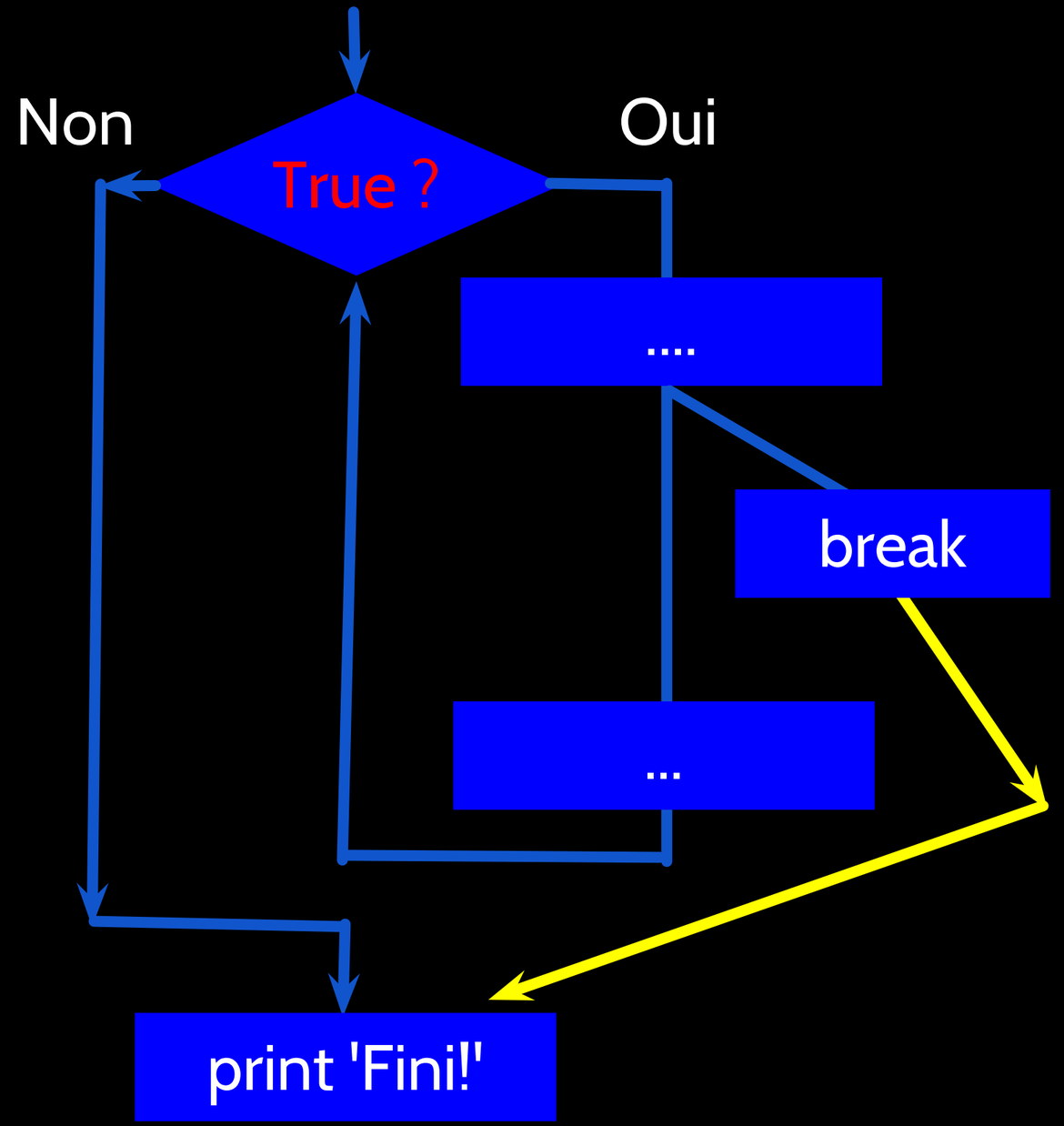
- Le mot-clé break met fin à la boucle courante et réalise un saut vers l'instruction suivant immédiatement cette boucle.
- Assimilable à un test réalisable n'importe où dans le corps de la boucle.

```
while True:
    ligne = raw_input('> ')
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```

```
> Bonjour
Bonjour
> termine
termine
> fini
Fini!
```

```
while True:
    ligne = raw_input('> ')
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```

http://en.wikipedia.org/wiki/Transporter_(Star_Trek)



Terminer une itération avec “continue”

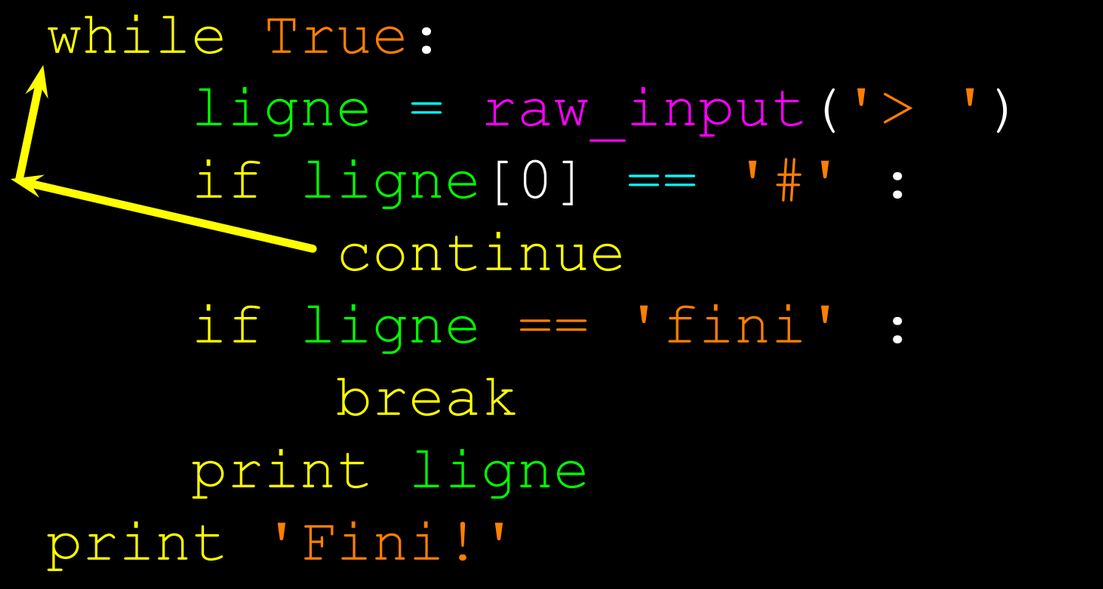
Le mot-clé continue met fin à l’itération actuelle, réalise un saut vers le début de la boucle et démarre une nouvelle itération.

```
while True:
    ligne = raw_input('> ')
    if ligne[0] == '#' :
        continue
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```

```
> bonjour
bonjour
> # a ne pas afficher
> a afficher!
a afficher!
> fini
Fini!
```

Terminer une itération avec “continue”

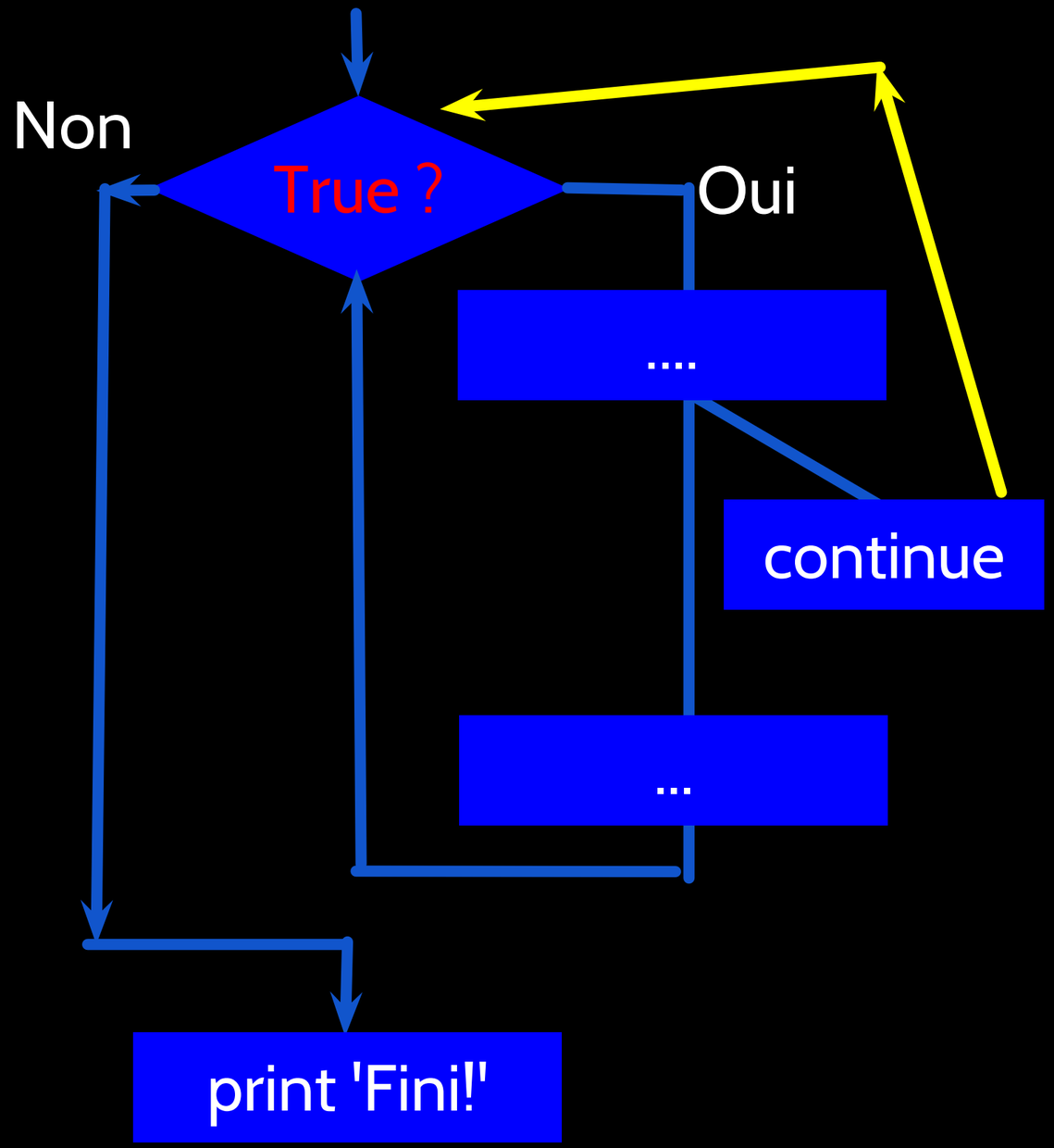
Le mot-clé continue met fin à *l’itération actuelle*, réalise un saut vers le début de la boucle et démarre une nouvelle itération



```
while True:
    ligne = raw_input('> ')
    if ligne[0] == '#' :
        continue
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```

```
> bonjour
bonjour
> # a ne pas afficher
> a afficher!
a afficher!
> fini
Fini!
```

```
while True:
    ligne = raw_input('> ')
    if ligne[0] == '#' :
        continue
    if ligne == 'fini' :
        break
    print ligne
print 'Fini!'
```



Boucle indéfinie

- Les boucles “While” sont qualifiées de “boucles indéfinies” car elles s’éxécutent jusqu’à ce qu’une condition logique devienne fausse
- Pour les boucles rencontrées jusqu’ici, il est assez facile de déterminer si elles se termineront ou seront des “boucles infinies”
- Il est parfois plus ardue d’être sûr qu’une boucle se terminera

Boucles définies

- Bien souvent on dispose d'une liste d'éléments, par exemple des lignes dans un fichier - constituant un ensemble fini d'éléments
- On peut construire une boucle qui sera parcourue une fois pour chacun des éléments d'un ensemble en utilisant le mot-clé Python for
- Ces boucles sont appelées "boucles définies" car elles s'exécutent un nombre fini de fois
- On dit que "les boucles définies itèrent en suivant les éléments d'un ensemble"

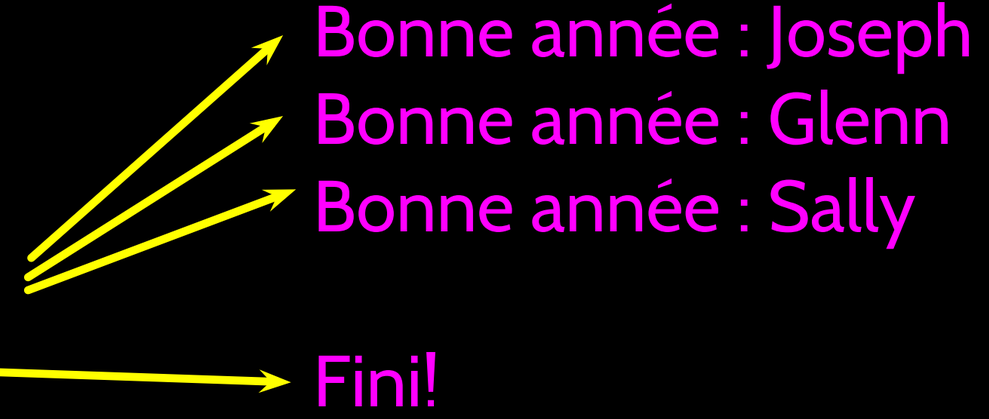
Une boucle définie simple

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'Feu !'
```

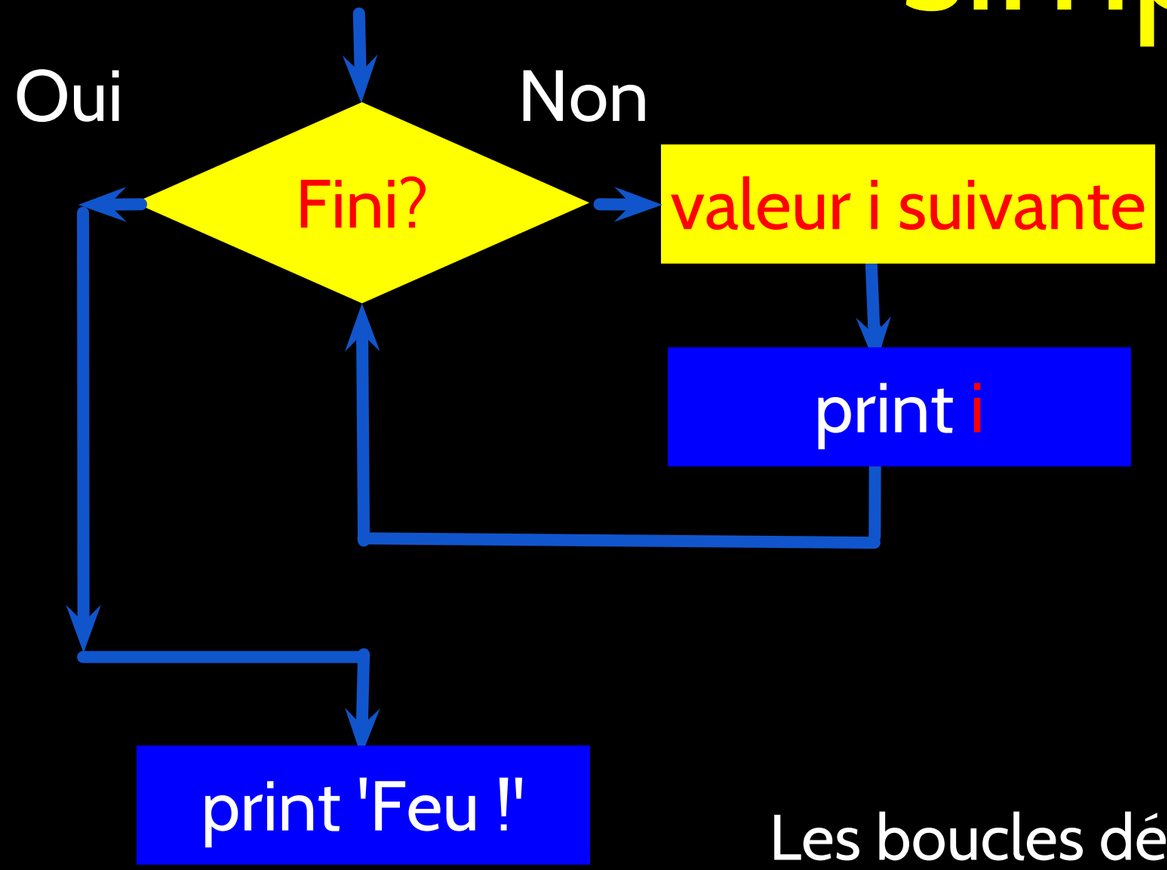
```
5
4
3
2
1
Feu !
```

Une boucle définie par des chaînes

```
amis = ['Joseph', 'Glenn', 'Sally']
for ami in amis :
   print 'Bonne année :',  friend
print 'Fini!'
```



Une boucle définie simple



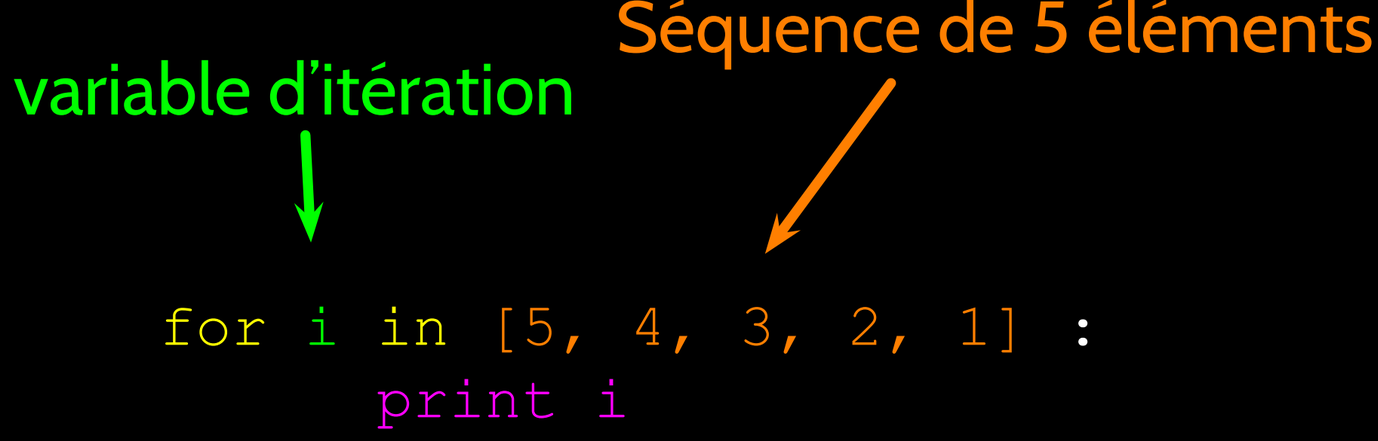

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'Feu !'
```

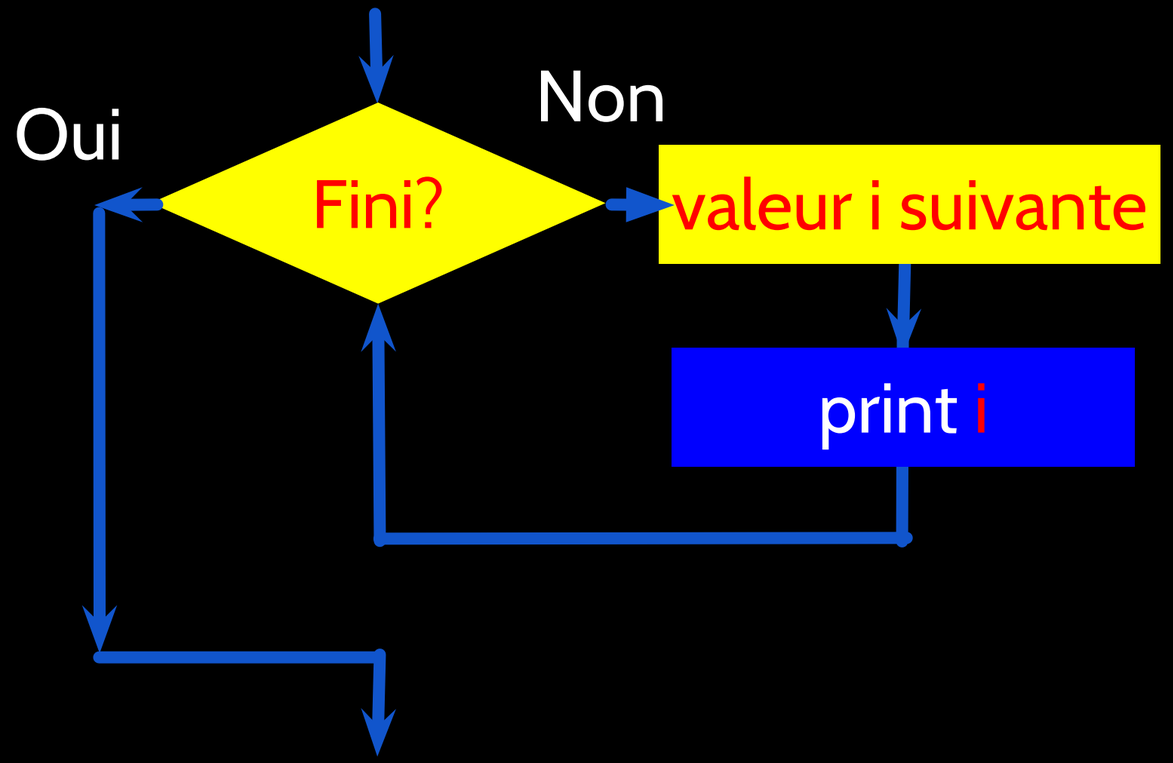
```
5
4
3
2
1
Feu !
```

Les boucles définies ont des variables d'itération explicites dont la valeur change à chaque parcours de la boucle. Ces variables d'itération parcourent la séquence ou l'ensemble les définissant.

Zoom sur “In”

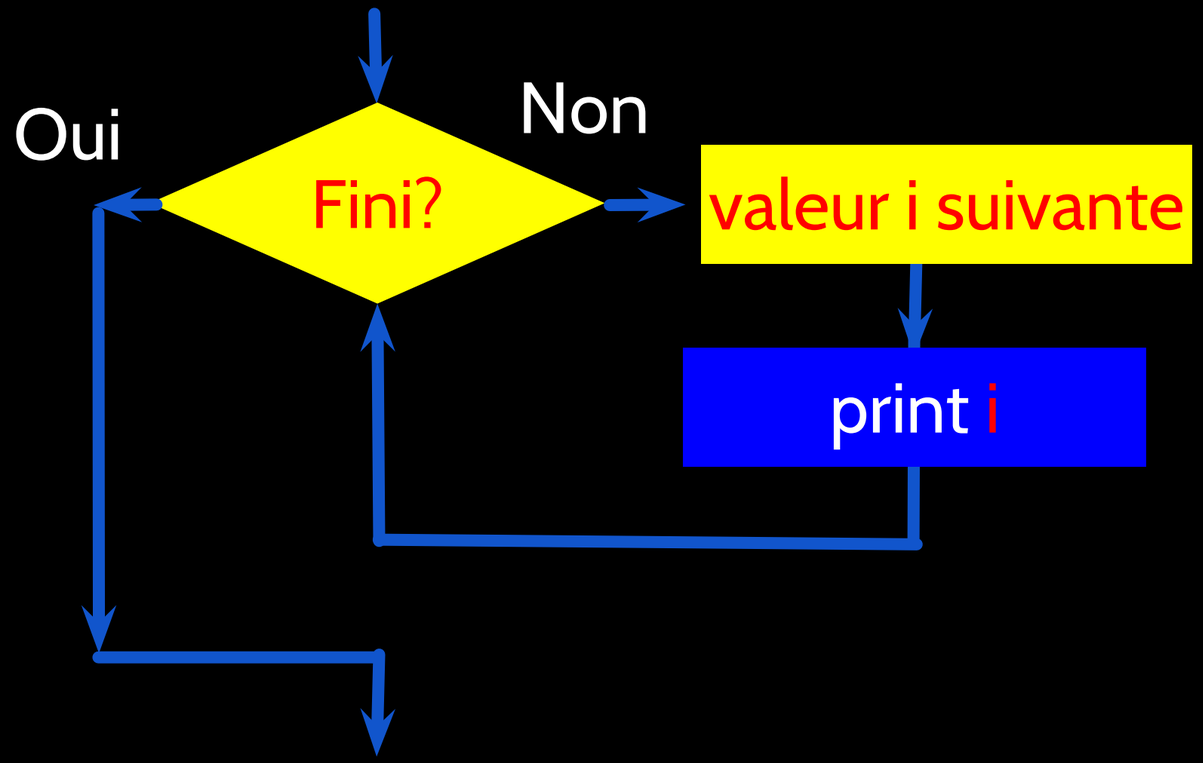
- La variable d’itération “itère” le long d’une séquence (ensemble ordonné)
- Le bloc (corps) de code est exécuté une fois pour chaque valeur (in) de la séquence
- La variable d’itération parcourt l’ensemble des valeurs de la séquence



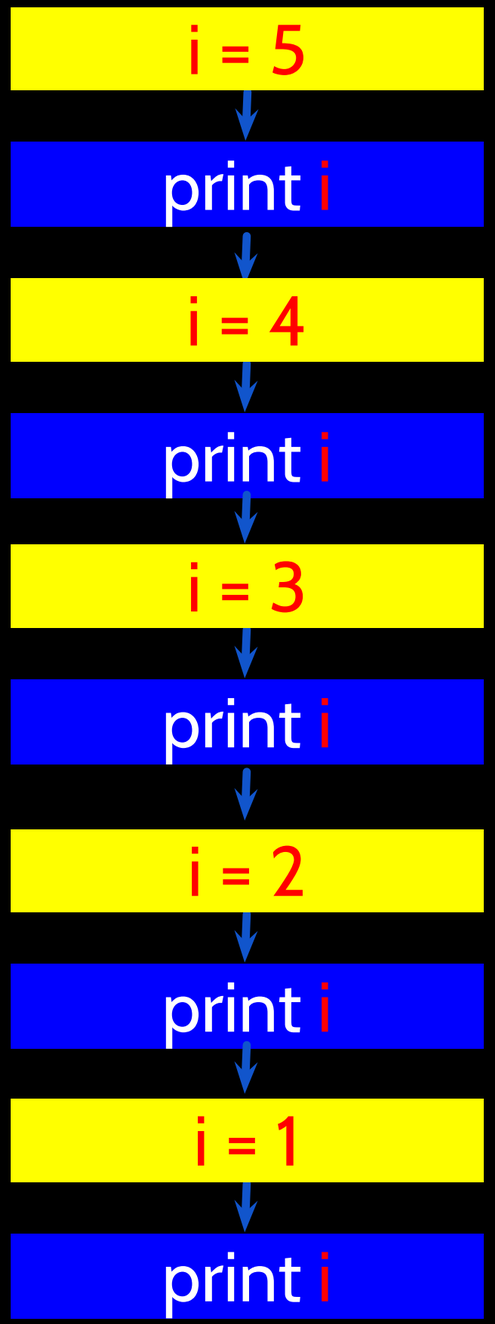



```
for i in [5, 4, 3, 2, 1] :
    print i
```

- La variable d'itération "itère" le long d'une séquence (ensemble ordonné)
- Le bloc (corps) de code est exécuté une fois pour chaque valeur (in) de la séquence
- La variable d'itération parcourt l'ensemble des valeurs de la séquence




```
for i in [5, 4, 3, 2, 1] :
    print i
```



Boucles définies

- Bien souvent on dispose d'une liste d'éléments, par exemple des lignes dans un fichier - constituant un ensemble fini d'éléments
- On peut construire une boucle qui sera parcourue une fois pour chacun des éléments d'un ensemble en utilisant le mot-clé Python for
- Ces boucles sont appelées "boucles définies" car elles s'exécutent un nombre fini de fois
- On dit que "les boucles définies itèrent en suivant les éléments d'un ensemble"

Structure des boucles: Regardons ce qui se passe

Note: Même si ces exemples sont simples, les structures présentées ici s'appliquent à toutes les sortes de boucles

Construire des boucles “intelligentes”

Quand on est coincé à écrire du code qui effectue une étape à la fois, l'astuce est d'avoir une connaissance claire de ce qui doit arriver dans la boucle.

Définir la valeur initiale des variables

qui agissent sur les données:

Toujours effectuer une tâche à la fois : Rechercher, modifier ou mettre à jour une variable.

Vérifier les variables

Effectuer une boucle dans un ensemble

```
print 'Avant'
for thing in [9, 41, 12, 3, 74, 15] :
     print thing
print 'Après'
```

```
$ python basicloop.
py
Avant
9
41
12
3
74
15
Après
```

Quel est le plus grand nombre?

Quel est le plus grand nombre?

3 41 12 9 74 15

le plus grand
(jusqu'à présent) :

-1

Trouver la plus grande valeur

```
largest_so_far = -1
print 'Avant', largest_so_far
for the_num in [9, 41, 12, 3, 74, 15] :
    if the_num > largest_so_far :
        largest_so_far = the_num
    print largest_so_far, the_num

print 'Après', largest_so_far
```

```
$ python largest.py
Avant -1
9 9
41 41
41 12
41 3
74 74
74 15
Après 74
```

Nous créons une variable qui contient la plus grande valeur vue jusqu'à présent. Si un nouveau nombre apparait plus grand, il devient la nouvelle plus grande valeur.

Comptage dans une boucle

```
compteur = 0
print 'Avant', zork
for thing in [9, 41, 12, 3, 74, 15] :
    compteur = compteur + 1
    print compteur, thing
print 'Après', compteur
```

```
$ python countloop.py
Avant 0
1 9
2 41
3 12
4 3
5 74
6 15
Après 6
```

Afin de compter combien de fois une boucle est exécutée, nous introduisons une variable "compteur" qui est initialisé à 0 et à laquelle nous ajoutons 1 pour chaque exécution du corps de la boucle.

Addition dans une boucle

```
accumulateur = 0
print 'Avant', accumulateur
for valeur in [9, 41, 12, 3, 74, 15] :
    accumulateur = accumulateur + valeur
    print accumulateur, valeur
print 'Après', accumulateur
```

```
$ python countloop.py
Avant 0
9 9
50 41
62 12
65 3
139 74
154 15
Après 154
```

Nous voulons additionner une valeur que nous rencontrons dans une boucle, nous introduisons une variable accumulateur initialisée à 0 et nous ajoutons la valeur à la somme à chaque passage dans la boucle.

Une boucle pour trouver la moyenne

```
count = 0
sum = 0
print 'Avant', count, sum
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print count, sum, value
print 'Après', count, sum, sum / count
```

```
$ python averageloop.py
Avant 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
Après 6 154 25
```

La moyenne combine le modéle du comptage et de l'addition et divise la somme lorsque la boucle se termine.

Filtrer les valeurs dans une boucle

```
print 'Avant'
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
       print 'Grand nombre',value
print 'Après'
```

```
$ python search1.py
Avant
Grand nombre 41
Grand nombre 74
Après
```

Pour identifier/filtrer une valeur particulière, nous utiliserons l'instruction if à l'intérieur du corps de la boucle.

Recherche avec la variable Boolean

```
found = False
print 'Avant', found
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
            found = True
     print found, value
print 'Après', found
```

```
$ python search1.py
Avant False
False 9
False 41
False 12
True 3
True 74
True 15
Après True
```

Si nous voulons juste chercher et savoir si une valeur a été trouvée, nous utilisons une variable booléenne initialisée à False puis déclarée comme True dès que nous avons trouvé ce que nous recherchions.

Quel est le plus petit nombre?

Quel est le plus petit nombre?

9 41 12 3 74 15

Le plus petit jusqu'à présent: **–1**

Quel est le plus grand nombre?

3

Le plus grand jusqu'à présent:

3

Quel est le plus petit nombre?

9 41 12 3 74 15

Le plus petit jusqu'à présent: None

Quel est le plus grand nombre?

12

Le plus grand jusqu'à présent:

41

Trouver la plus petite valeur

```
smallest = None
print 'Avant'
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'Après', smallest
```

```
$ python smallest.py
Avant
9 9
9 41
9 12
3 3
3 74
3 15
Après 3
```

Nous avons encore une variable smallest qui est “la plus petite jusqu'à présent”. Avant le premier passage dans la boucle smallest est déjà initialisé à None, puis la première (valeur) value devient smallest (le minimum) .

Les opérateurs “is” et “is not”

```
smallest = None
print 'Avant'
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'Après', smallest
```

- Python a un opérateurr is qui est utilisé dans les expressions logiques
- Signifie “est le même que”
- Comparable à == mais dans un sens plus fort
- is not est également un opérateur logique

Résumé

- Boucles While (indéfinies)
- Boucles infinies
- Utilisation de break
- Utilisation de continue
- Boucles For (définies)
- Variables d’itération
- Structure des boucles
- Plus grand ou plus petit

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors and Translators here

...