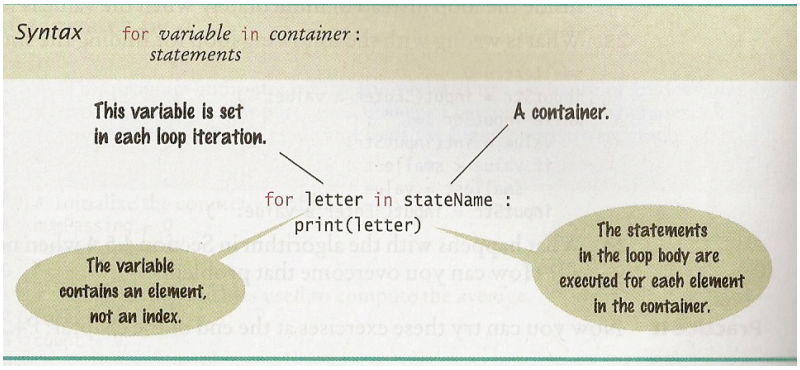
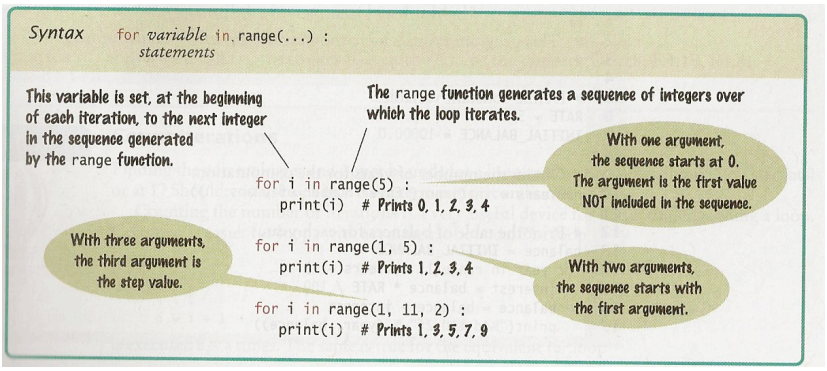
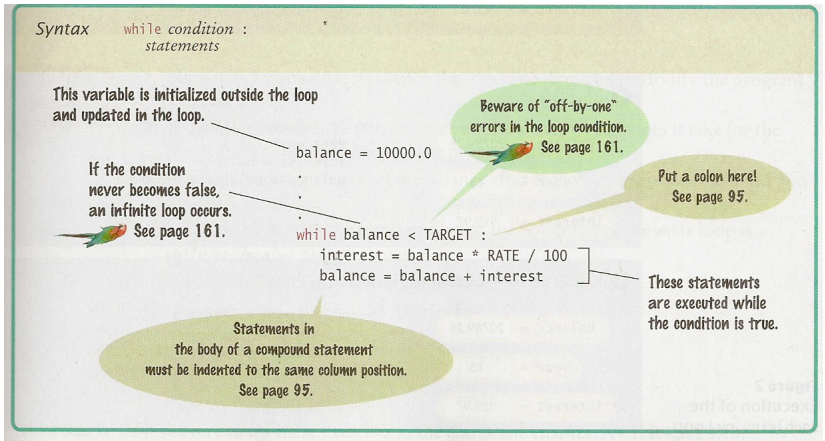
Boucles inconditionnelles – FOR – WHILE

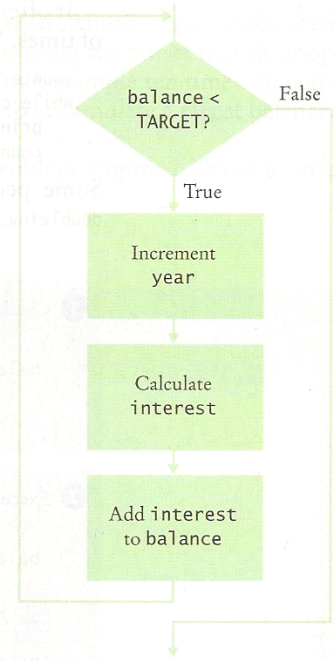
La boucle FOR





La boucle WHILE

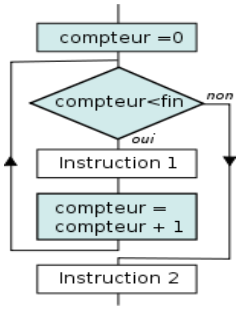




Boucles inconditionnelles – FOR - WHILE

LES BOUCLES FOR

1. Introduction



La boucle FOR est en informatique une structure de contrôle de programmation permettant de réaliser une boucle associée à une variable entière qui sera incrémentée à chaque itération.

L'intérêt du "FOR" est d'épargner un peu de fatigue au programmeur, en lui évitant de gérer lui-même la progression de la variable qui lui sert de compteur (on parle d'incrémentation).

On l'utilise lorsque l'on connait le nombre d'itération dans une boucle.

2. L'algorithme d'une boucle FOR

Exercice

Comment obtenir cet affichage ?

```
Allo ?
Allo ?
Allo ?
Non mais allo quoi !
```

pour *i allant de* 10 *à* 12 **faire**
  Afficher « Allo ? »
Afficher « Non mais allo quoi ! »

Programme :

```
for i in range(10,13):
    print("Allo ?")
print("Non mais allo quoi !")
```

Exercice

Comment compter le nombre d'entiers ≤ 1000 qui sont premiers ? Et leur somme ?

$c \leftarrow 0$ # un compteur
pour *i allant de* 1 *à* 1000 **faire**
  si *i est premier* **alors**
    $c \leftarrow c + 1$
Résultat : c

$s \leftarrow 0$ # un «sommeur»
pour *i allant de* 1 *à* 1000 **faire**
  si *i est premier* **alors**
    $s \leftarrow s + i$
Résultat : s

3. La boucle For dans Python

SAVOIR-FAIRE Écrire un programme utilisant une boucle for

1. On détermine combien de fois la boucle devra s'exécuter, ce nombre étant en général exprimé en fonction d'une ou plusieurs variables du programme.
2. On choisit une variable pour le compteur et on identifie si elle doit jouer un rôle dans le corps de la boucle.
3. On écrit le corps de la boucle.
4. Comme pour la boucle `while`, il peut être nécessaire de prévoir une initialisation des variables en amont de la boucle et un post-traitement en aval.

Boucles inconditionnelles – FOR - WHILE

Utilisation de range()

- "*range*" permet de générer une liste d'entiers croissants successifs qui peuvent être utilisés comme **index dans une itération.**
- "*range*" peut avoir **de un à trois arguments**.

1. Si il n'y a **qu'un argument** il représente le nombre d'éléments de la liste partant **depuis zéro** et incrémenté de un à chaque fois.

```
>>> for i in range(10):
        print (i, end=" ")
0 1 2 3 4 5 6 7 8 9
```

2. Si "*range*" est appelé avec **deux arguments**, il s'agit de la **borne de départ de la liste et de celle d'arrivée** (non comprise), l'incrément entre deux éléments est de un.

```
>>> for i in range(10,20):
        print (i, end=" ")
10 11 12 13 14 15 16 17 18 19
```

3. Quand "*range*" comporte **trois arguments** il s'agit de la borne de départ de la liste, celle d'arrivée non comprise et **du pas d'incrémentation.**

```
>>> for i in range(10,20,3):
        print (i, end=" ")
10 13 16 19
```

4. Quelques exemples

Exercice

Comment calculer 841^{42} avec seulement des multiplications ?

```
r=841
for i in range(0,40):
    r=r*841
print("le resultat de 841 puissance 42 est:",r)
```

- $r \leftarrow 841$ (r va être multiplié par 841... 41 fois)
- $r \leftarrow r \times 841$ (r vaut alors 841^2)
- $r \leftarrow r \times 841$ (r vaut alors 841^3)
- ...
- $r \leftarrow r \times 841$ (r vaut alors 841^{42})

Algorithme

$r \leftarrow 841$
pour *i allant de* 0 *à* 40 **faire**
  $r \leftarrow r \times 841$
Résultat : r

Exercice

Comment calculer 841! avec seulement des multiplications ?

```
r=1
for i in range(2,841):
    r=r*i
print("le resultat de 841! est:",r)
```

- $r \leftarrow 1$ (r va être multiplié par 2, puis 3, ... puis 841)
- $r \leftarrow r \times 2$ (r vaut alors 2)
- $r \leftarrow r \times 3$ (r vaut alors 2×3)
- ...
- $r \leftarrow r \times 841$ (r vaut alors 841!)

Algorithme

$r \leftarrow 1$
pour *i allant de* 2 *à* 841 **faire**
  $r \leftarrow r \times i$
Résultat : r

Boucles inconditionnelles – FOR - WHILE

5. Initialisation des variables avant la boucle

SAVOIR-FAIRE Initialiser les variables

On a déjà vu que, tant que l'on n'a pas affecté une valeur à une variable, celle-ci est absente de l'état. Il faut donc s'assurer qu'à chaque fois qu'une variable est utilisée dans une expression, elle est déjà présente dans l'état, sous peine d'une erreur. Pour cela :

1. On identifie la première ligne du programme où la variable est utilisée. Lorsque l'algorithme comporte des instructions conditionnelles, il peut y avoir plusieurs telles lignes pour la même variable, en fonction du résultat du test.
2. On vérifie que cette première ligne est toujours une affectation de cette variable. Elle peut être rendue difficile à repérer parce qu'une instruction `for` la gère ou parce qu'elle est combinée avec une saisie au clavier.
3. Enfin, si une initialisation est manquante, il faut déterminer une valeur d'initialisation cohérente avec la suite du programme et ajouter l'instruction correspondante avant la première utilisation de la variable.

Exercice : Corriger le programme suivant

```
n = int(input())
for i in range(n):
    if i % 2 == 0:
        carre = i ** 2
    else:
        carre = 0
    total = total + carre
```

Ce programme semble prévu pour calculer la somme des carrés des entiers pairs strictement inférieurs à n.

Il comporte quatre variables :

- *`n` est initialisée par une entrée au clavier en tout début de programme.*
- *`i` est un compteur de boucle et n'a donc pas besoin d'être initialisée.*
- *`carre` apparaît pour la première fois dans l'instruction conditionnelle et elle y est correctement initialisée, quel que soit le résultat du test `i % 2 == 0`.*
- *`total` n'apparaît que dans la dernière ligne de la boucle. Cette ligne est bien une affectation de `total`, mais avec une expression qui dépend elle-même de `total` : il sera impossible d'évaluer cette expression lors de la première itération. Il faut donc initialiser `total` avant d'entrer dans la boucle, avec la valeur 0 pour ne pas perturber la somme qui sera calculée par la suite.*

6. Ruptures de séquences

6.1. L'instruction BREAK

Dans certains cas, il peut être intéressant d'afficher un résultat en cours de boucle (FOR ou WHILE). On dit que l'on veut sortir de la boucle. Python utilise l'instruction BREAK.

Exemple :

```
>>> for x in range(1, 11):
...     if x == 5:
...         break
...     print(x, end=" ")
...
1 2 3 4
```

L'instruction « end » est intéressante : `print (x, end=" ")`

Elle permet de séparer d'un espace les résultats, sinon, ils sont affichés en colonne.

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

6.2. L'instruction CONTINUE

Cette instruction permet de passer immédiatement à l'itération suivante de la boucle FOR ou WHILE. Elle reprend à la ligne de l'en-tête de la boucle.