Leçon d'informatique : Boucle `for`

S. Benlhajlahsen - PCSI$_1$

Sommaire

Une **boucle**[1] est une syntaxe permettant d'écrire des instructions une seule fois et d'en demander la répétition. Il y en a de deux sortes :

- les boucles `for`[2], quand on sait d'avance combien de fois on va répéter les instructions ;
- les boucles `while`[3], quand on continue à répéter les instructions jusqu'à ce qu'une condition cesse d'être vérifiée, peu importe combien de fois.

Chaque répétition des instructions s'appelle un "tour de boucle" ou une "**itération**". Bien sûr, les boucles for peuvent être vues comme un cas particulier des boucles while (il suffit que la condition porte sur le nombre de tours), mais vous verrez qu'il est tout de même bien pratique de disposer des deux syntaxes. Ce chapitre porte uniquement sur les boucles `for`.

I Syntaxes et utilisation

I.A syntaxe

Syntaxe ♡ La boucle for fait intervenir deux mots-clefs, `for` et `in`, et les instructions à répéter sont dans un bloc indenté. variable_de_boucle est la "variable de boucle" : elle prend pour valeur, successivement, les valeurs contenues dans liste. Le changement se fait au début de chaque tour de boucle.

```
for variable_de_boucle in liste:
    instructions
```

FIGURE 1 – Syntaxe de la boucle `for`.

Remarque La variable de boucle est automatiquement créée au moment du démarrage de la boucle, en se voyant affecter la première valeur dans la liste. La boucle s'arrête quand on a parcouru toutes les valeurs de la liste.

I.B générer la liste des valeurs

"liste" La "liste" peut être tout objet structuré dont :

- une "vraie" liste. Par exemple : `[0,1,2,3]` ;
- une chaîne de caractère. Par exemple : `"abcdef"` ;
- le résultat de la fonction `range`.

Syntaxe `range`[a] Dans son usage le plus courant, `range` peut prendre de un à trois arguments **entiers positifs** :

- `range(stop)` génère une liste de stop entiers correspondant à `[0,stop - 1]` ;
- `range(start, stop)` génère une liste de stop-start entiers correspondant à `[start, stop - 1]` ;
- `range(start, stop, step)` génère une liste d'entiers inclus dans `[0,stop - 1]` et deux-à-deux distants de `step`.

1. *loop* en anglais.
2. aussi boucle inconditionnelle.
3. aussi boucle conditionnelle.

a. intervalle en français.

Remarque On constate que le `start est inclus` alors que le `stop` est exclus.

Exemples Etudions les exemples des scripts des figures 2 à 5.

```
for i in range(3):
    print("i vaut", i)
# Affichage sur la console :
i vaut 0
i vaut 1
i vaut 2
```

FIGURE 2 – Exemple de boucle `for`.

```
for x in range(2,6):
    print("x vaut", x)
# Affichage sur la console :
x vaut 2
x vaut 3
x vaut 4
x vaut 5
```

FIGURE 3 – Exemple de boucle `for`.

```
for i in range(1,5,2):
    print("i vaut", i)
# Affichage sur la console :
i vaut 1
i vaut 3
```

FIGURE 4 – Exemple de boucle `for`.

Remarque Le nom de variable peut être choisi librement mais cela crée, à la fin de la boucle une variable globale. Par exemple, après avoir exécuté chacune de ces boucles, `x` vaut 5 et `i` vaut 3.

Remarque Nous verrons plus tard comment itérer sur d'autres éléments, on notera que :

- `for x in "bonjour":` provoquera 7 tours de boucles où `x` vaudra successivement les différentes lettres de la chaîne de caractères `"bonjour"`.
- `for y in [1,2,6,8]:` provoquera 4 tours de boucles où `x` vaudra successivement les éléments de la liste `[1,2,6,8]`.

Remarque La syntaxe `range(start, stop, step)` fonctionne aussi si `start>stop` à condition que `step<0`. Cela revient à parcourir les entiers dans le sens décroissant (voir script de la figure 6).

I.C exemples fondamentaux

```python
for j in range(3,3):
    print("j vaut", j)
# Aucun affichage sur la console.
```

FIGURE 5 – Exemple de boucle `for`.

```python
for j in range(10,3,-2):
    print("j vaut", j)
# Affichage sur la console.
j vaut 10
j vaut 8
j vaut 6
j vaut 4
```

FIGURE 6 – Exemple de boucle `for`.

Factorielle On souhaite écrire une fonction `factorielle` qui prend en argument un entier `n`.
On rappelle que $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$.

```python
def factorielle(n):
    res = 1
    for i in range(1,n+1):
        res *= i
    return res
print(factorielle(3))
```

FIGURE 7 – Exemple de la fonction `factorielle`.

Solution Analysons ce programme de la figure 6 :

- n valant 3 dans cet exemple, le range produit la liste de valeurs $[\![1;3]\!]$.
- la variable locale `res` sert à stocker la valeur du produit. On lui donne comme valeur initiale l'élément neutre de la multiplication, donc 1.

Remarque le premier tour de boucle (pour i=1) ne sert à rien, puisqu'on y multiplie… par 1. Regardons si on peut remplacer la ligne 4 par `for i in range(2, n+1) :`. Si `n` vaut 2 ou plus alors ça ne change rien. S'il vaut 1, on se retrouve avec `range(2,2)`, autrement dit une liste vide. Il n'y a alors aucun tour de boucle effectué et la fonction renvoie 1, ce qui est correct. Donc cette petite modification est possible (même si le gain de performance est très mineur).

Tirage de deux dés différents : boucles imbriquées On jette deux dés : un dé à 10 faces (résultat de 0 à 9) pour générer un chiffre des dizaines, et un dé à 6 faces (résultat de 1 à 6) pour générer le chiffre des unités. Le code suivant affiche toutes les valeurs possibles de ce tirage (voir figure 8). On appelle une telle structure des **boucles imbriquées**[a]. La boucle sur `d10` est appelée **boucle extérieure** et celle sur `d6` **boucle intérieure**.

a. *nested loops* en anglais

II Astuces

II.A Récurrence

```
for d10 in range(0,10) :
    for d6 in range(1,7) :
        print(d10*10 + d6)
```

FIGURE 8 – Boucles imbriquées.

Récurrence d'ordre 1 Une récurrence d'ordre 1 est une récurrence dans laquelle u_n se calcule avec la connaissance de u_{n-1} seulement. Prenons, par exemple, $u_n = 2u_{n-1} + 1$ avec $u_0 = 1$. On souhaite calculer u_5. Bien sûr, on veut utiliser une boucle `for`, mais on ne veut pas créer 6 variables pour stocker u_0 à u_6 ! On se reportera à la figure 9.

```
def recurence(n):
    u_ancien = 1
    for i in range(n):
        u_nouveau = 2*u_ancien+1
        u_ancien = u_nouveau
    return u_nouveau
```

FIGURE 9 – Récurrence.

Analyse La logique est de créer un "roulement" sur deux variables seulement, puisque la relation de récurrence ne met en jeu que deux éléments de la suite $(()\, u_n)$ à un instant donné. Ceci est décortiqué dans la table 1.

Récurrence Boucle	n itération	u_{n-1} `u_ancien`	u_n `u_nouveau`
	1	1	3
	2	3	7
	3	7	15
	4	15	31
	5	31	63

TABLE 1 – Récurrence.

Remarque En réalité, on a pas besoin de stocker deux valeurs à tout instant. Le script de la figure 10 est beaucoup plus efficace.

II.B Vérifier les valeurs extrêmes de la variable de boucle

Sachant que `range(p,n)` commence à `p` mais s'arrête en fait à `n-1`, on a parfois des hésitations sur les arguments à lui donner.

Idée `range(n)` ? `range(n-1)` ? `range(n+1)` ? `range(1,n)` ? Quand vous avez un doute, vérifiez deux choses :

- Nombre de tours de boucle : `range(p,n)` va donner `n-p` tours, est-ce la valeur attendue ?
- Valeurs extrêmes : vérifiez les instructions quand la variable de boucle prend sa valeur minimale `p` puis sa valeur maximale `n-1`.

III Exercices

III.A énoncés

Exercice 1 : Ecrire une fonction `suite` qui prend en argument une fonction f et deux entiers `n` et `u0` et qui renvoie le

```python
def recurence(n):
    u = 1
    for i in range(n):
        u = 2*u+1
    return u
```

FIGURE 10 – Récurrence. Version simplifiée.

n-ième terme de la suite récurente définie par :

$$u_n = f(u_{n-1})$$

Que vaut le 10-ème terme de la suite dans le cas où $u_0 = 1/2$ et $f : x \mapsto \frac{1}{2} \cdot \left(1 + x^2\right)$?

Exercice 2 : Écrire une fonction `sommeCarres(n)` qui renvoie la somme des carrés des entiers de 1 à n.

Exercice 3 : La suite des nombres de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour tout entier n. Écrire une fonction `fibo(n)` qui calcule F_n.

Exercice 4 : Écrire une fonction `estPremier(n)` qui renvoie `True` ou `False` selon que le nombre donné en entrée est ou non premier.
Un entier n est premier si et seulement si il n'est divisible par aucun entier compris entre 2 et $n - 1$.

Défi : Écrire cette fonction sans instruction conditionnelle `if`.

Exercice 5 : Ecrire une fonction `losange` qui prend en argument un entier impaire `n` et un caractère `ch` et qui affiche un losange dans la console. Ainsi, `losange(5,"#")` puis `losange(7,"@")` afficheront :

```
  #
 ###
#####
 ###
  #
```

```
   @
  @@@
 @@@@@
@@@@@@@
 @@@@@
  @@@
   @
```

III.B éléments de réponses

Réponse à l'exercice 1 On se reportera au script de la figure 11.

```python
def suite(f,u0, n):
    u = u0
    for i in range(n):
        u = f(u)
    return u
def f(x):
    return (1+x**2)/2
print(suite(f, 0.5, 10))
```

FIGURE 11 – script de l'exerice 1.

Réponse à l'exercice 2 On se reportera au script de la figure 12.

```python
def sommeCarres(n):
    somme = 0
    for i in range(n): # i varie de 0 à (n-1)
        p = i + 1 # pour aller de 1 à n
        somme = somme + p**2
    return somme
def sommeCarres(n):
    somme = 0
    for i in range(n+1): # i varie de 0 à (n)
        somme = somme + (i)**2
    return somme
```

FIGURE 12 – script de l'exerice 2.

Réponse à l'exercice 3 La récurrence utilise deux termes, on va donc conserver en mémoire deux termes à la fois. Cela revient à transformer la récurrence d'ordre 2 en une double récurrence d'ordre 1 : $(F_{n+1}, F_{n+2}) = (F_{n+1}, F_n + F_{n+1})$ est une fonction du couple (F_n,F_{n+1}). On se reportera au script de la figure 13.

```python
def fibo(n):
    f = 0
    f_suivant = 1
    for i in range(n):
        f_ancien = f
        f = f_suivant
        f_suivant = f_ancien + f_suivant
    return f
```

FIGURE 13 – script de l'exerice 3.

Réponse à l'exercice 4 On suit la définition : on crée un booléen avec la valeur `True` que l'on transformera en `False` si un entier est un facteur de n. On se reportera au script de la figure 14.

Réponse à l'exercice 5 On se reportera au script de la figure 15.

```python
# solution sans instruction if
def estPremier(n):
    resultat = True # on se suppose le nombre premier
    for p in range(2, n):
        resultat = resultat and not(n%p == 0)
    return resultat

# le plus simple est tout de même d'utiliser une instruction conditionnelle.
def estPremier(n):
    resultat = True
    for p in range(2,n):
        if n%p == 0:
            resultat = False
    return resultat

# On peut éviter de continuer à tester si le nombre admet un diviseur.
def premier(n):
    for p in range(2,n):
        if n%p == 0:
            return False
    return True
```

FIGURE 14 – script de l'exerice 4.

```python
def triangle1(n,ch):
    for i in range(1,n+1):
        print(i*ch)
triangle1(5,"*")
def triangle2(n,ch):
    for i in range(n):
        print((n-i)*ch)
def triangle2(n,ch):
    for i in range(n,0,-1):
        print(i*ch)
triangle2(5,"*")
def losange(n,ch):
    for i in range(n//2):
        print((n//2-i)*" "+(2*i+1)*ch+(n//2-i)*" ")
    for i in range(n//2,-1,-1):
        print((n//2-i)*" "+(2*i+1)*ch+(n//2-i)*" ")
losange(7,"*")
```

FIGURE 15 – script de l'exerice 5.