

Quatrième partie

Itérations

1 Introduction

Une itération est une répétition de calculs.

C'est un élément indispensable en algorithmique : il permet de décomposer un problème en une répétition de sous-problèmes plus simples.

Il existe deux grands types d'itérations :

- les itérations bornées : le nombre d'itérations est connu à l'avance. En Python, cela est mis en œuvre (principalement) sur les séquences (données composées, vues plus tard!).
- les itérations non bornées : le nombre d'itérations n'est pas connu. Il dépend des données et des calculs réalisés : utilisation d'une condition pour savoir s'il faut continuer ou non. En Python, cela est mis en œuvre à l'aide de l'instruction composée `while`.

2 Ré-affectations

Il est possible d'affecter autant de fois que nécessaire une variable. Ceci n'est pas le cas dans tous les langages, l'assignation unique ayant de bonnes propriétés pour analyser le code.

Exemple 2.1. Sur le portable :

```
valeur = 23
print (valeur ,end="")
valeur = 234678098
print (valeur)
```

Il y a une temporalité liée à l'affectation : la valeur des variables ne peut être connue sans indication de « temps ».

Exemple 2.2. `x = 23`

```
y = x # x et y ont même valeur
x = 35 # plus maintenant
x = x+1
```

Retour sur l'affectation multiple :

```
x,y = 23, 89
```

Comment échanger les valeurs contenues dans les deux variables **a** et **b**?

Après échange avec les étudiants, montrer sur le portable :

Exemple 2.3. `x,y = 23, 3456`

```
x,y = y, x
print ("x =",x, "et y =", y)
```

Faire, si le temps, un exemple avec trois variables.

3 Mise en œuvre de l'itération

Schéma usuel en algorithmique :

```
~~~~tantque (condition)

~~~~~corps de l'itération
```

Fonctionnement d'une telle instruction :

la condition est évaluée (type booléen!). Deux possibilités :

faux : le corps est ignoré, et, s'il y en a une, on exécute l'instruction suivant ce corps ;

vrai : le corps est exécuté (cela constitue une itération), et à la fin on reprend sur l'évaluation de la condition.

En Python, utilisation du mot clef **while** : c'est une instruction composée.

Exemple 3.1. Exemple sur le portable (exemple3_1.py) :

```
a=1
while (a<10):
    print (a, a**2, a**3)
    a = a+1
```

Poser la question de ce qui se produit si la dernière instruction est oubliée. Idem si on met `a = a - 1`.

3.1 Construction d'une itération

Pour construire une itération, il est important de procéder dans l'ordre suivant :

1. mettre en évidence les actions à réaliser pour passer d'une étape à la suivante. Cela donnera le corps de la boucle ;
2. trouver les conditions d'arrêt des calculs. Cela donnera la condition de l'itération.
3. trouver les valeurs initiales permettant la réalisation du calcul. Cela donnera les initialisations des variables utilisées dans l'itération.

3.2 Itération correcte

Conditions à respecter :

- les variables de la condition doivent être initialisées (s'il n'y a pas de variable dans la condition, il faut se poser la question de l'utilité de l'itération!!!).
- au moins une instruction du corps de la boucle doit modifier la valeur d'au moins une des variables de la condition.
- cette modification doit se faire de telle sorte qu'au bout d'un nombre fini d'itérations, la condition ne soit plus vérifiée (sinon, le programme boucle!).

C'est un point difficile mais fondamental pour une bonne programmation.

3.3 Calcul des éléments de la suite de Fibonacci

La suite de Fibonacci (XII^e) est définie par :

$$\text{fib}(0) = 0 \quad (1)$$

$$\text{fib}(1) = 1 \quad (2)$$

$$\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1) \quad (3)$$

Comment calculer les **10** premiers termes de cette suite à l'aide d'une itération ?

Il « suffit » d'appliquer la formule précédente 8 fois de suite (les deux premiers étant connus).

Pour calculer la valeur d'un rang donné, il faut les valeurs des deux rangs précédents.

On va noter **a** la valeur de rang inférieur, **b** l'autre valeur et **c** le rang de **b**.

Au départ, pour le rang 1, nous aurons `a=0`, `b=1` et `c=1`.

Faire le tableau pour les premières itération (mais pas la première!), en déduire les expressions de remplacement des 3 variables, et proposer la condition. Donner à la fin les valeurs initiales, et montrer qu'il faut un premier affichage pour `fib(0)` !

a	b	c
0	1	1
1	1	2
1	2	3
2	3	4
3	5	5
5	8	6
...
b	a+b	c+1

Exemple 3.2. Exemple sur le portable (fichier fibo.py) :

Spécification 1.

fib : vide → vide
→ affiche les 20 premiers termes de la suite

```
def fib ():
    "affiche les 20 premiers termes ...."
    a,b,c = 0, 1 , 1
    print ("Au rang 0 : 0")
    while (c <20 ):
        print ("Au rang",c ,":", b)
        a,b,c = b, a+b,c+1
```

Si le temps, faire la version récursive (cf fiboRec.py)!

3.4 Calcul de la racine carrée

Souvent, il n'est pas possible de déterminer à l'avance le nombre d'itérations à effectuer. Voici un exemple très simple, tiré du calcul numérique, qui illustre cette situation.

Le calcul de la racine carrée d'un nombre a peut être obtenue comme limite de la suite :

$$x_n = \frac{1}{2} \times (x_{n-1} + \frac{a}{x_{n-1}})$$

en partant de la valeur initiale $x_0 = a$

Bien reprendre la construction d'une itération. D'abord avec une seule variable, puis avec deux pour trouver la condition d'arrêt. Il faut que nous garantissions que ce calcul se termine. Appelons **eps** la précision avec laquelle nous voulons atteindre notre résultat. À la fin de chaque étape, on teste si la différence entre la nouvelle valeur et la précédente est supérieure à cette précision. Si oui, on continue, sinon on arrête. Pour finir, trouver les valeurs initiales.

4 Les chaînes de caractères

Elles constituent la première **séquence** de Python (une séquence est une collection ordonnées d'éléments).

Différentes actions sont possibles sur les chaînes :

- accès à un élément de la chaîne : **ch[5]** ;
- longueur d'une chaîne : **len(ch)** ;
- concaténation de chaînes : **ch = ch1 + ch2** ;
- conversion en nombre : **int(ch)**, **float(ch)**

N.B. : les chaînes ne sont pas modifiables

Exemple 4.1. Exemple sur le portable :

```

prenom, nom = 'Laurent', "Perraudau"
print (prenom[0]) (faire aussi avec 10!!)
len(prenom)
monMon = prenom+" "+nom
ch='567'
print (ch + 12)
print (int(ch) + 12)
prenom[0] = T
monNom = monNom + monNom

```

Faire exemple sur portable

4.1 Itération sur les chaînes

Nous allons chercher, dans un premier temps, à afficher une chaîne de caractères à l'envers.

Spécification 2.

$$\begin{aligned}
 \textit{afficheEnvers} : \textit{chaîne} &\rightarrow \textit{vide} \\
 \textit{ch} &\rightarrow \textit{affichage de ch à l'envers}
 \end{aligned}$$

Construire au tableau la base de la fonction avant de montrer le fichier `afficheEnvers.py`

Nous allons chercher, dans un second temps, à construire une chaîne de caractères inverse de celle passée en paramètre.

Spécification 3.

$$\begin{aligned}
 \textit{renverse} : \textit{chaîne} &\rightarrow \textit{chaîne} \\
 \textit{ch} &\rightarrow \textit{rend le contenu de ch renversé}
 \end{aligned}$$

En utilisant la fonction précédente, construire au tableau la base de la fonction avant de montrer le fichier `renverse.py`