Feuille de cours 3 (informatique) : boucles for

Lorsqu'une opération doit être répétée plusieurs fois dans un programme, on utilise une boucle. Les boucles sont des structures de contrôle fondamentales en informatique. Elles offrent une grande richesse dans l'écriture de programmes. Elles sont divisées en deux types : boucles `for` et boucles `while`. La boucle `for`, ou boucle indexée, s'utilise quand on sait déjà combien de fois on veut répéter les instructions.

1 Syntaxe des boucles `for`

La boucle `for` permet de répéter un bloc d'instructions un certain nombre de fois, sous la condition que ce nombre de fois soit *connu à l'avance*.

Pour cela, on utilise un *compteur* qui va parcourir une liste de valeurs. La syntaxe est la suivante :

```
for compteur in liste_de_valeurs :
    # Vos instructions a repeter dans la boucle
# Vos instructions hors de la boucle
```

où `liste_de_valeurs` est un objet dont on peut parcourir les éléments. Nous verrons cette année plusieurs objets de ce type : listes, chaînes de caractères... On dit que ces éléments sont des *itérables*.

Voici un premier exemple avec une liste de valeurs obtenue avec la fonction `range` (cf ci-dessous), indiquez ce que renvoie Python :

```
for x in range(1,6) :
    print(x)
print("c'est fini")
```

Remarque 1
On utilise ici la fonction `print` qui permet d'afficher dans la console la valeur d'une variable (ou aussi une constante, comme ici la chaîne de caractères `"c'est fini"`). **Attention à ne pas confondre `print` et `return`** :

- La commande `return` (qui n'est pas une fonction, donc ne nécessite pas l'emploi de parenthèses) s'emploie exclusivement dans les fonctions : elle permet de *renvoyer* le résultat `f(x)` d'une fonction f évaluée en x ; on peut ensuite utiliser `f(x)` comme une variable *contenant* la valeur $f(x)$.
- À l'inverse la fonction `print` (qu'on utilisera avec des parenthèses) sert simplement à faire *afficher* une valeur. Impossible de réutiliser cette valeur ensuite.

Pour ne pas créer de confusion, on n'utilisera pas de `print` dans des fonctions – sauf exception.

Remarque 2
- Les deux points et l'indentation sont fondamentaux.
- Le nom du compteur, ici `x`, est libre : c'est une variable soumise aux mêmes règles que les autres notamment pour son nom (ni espace ni tiret, ne pas commencer par un chiffre).

La boucle `for` affecte au compteur une valeur. Ainsi dans l'exemple `x` vaut successivement :

Voici un autre exemple utilisant pour itérable une *chaîne de caractères* (signalée par des guillemets). Que vaut successivement `x` ?

```
1 for x in "BCPST 1B" :
2     print(x)
```

Dans les boucles for, le plus souvent on souhaite répéter une opération un nombre (entier) N de fois, et éventuellement suivre à quelle étape $k \in [\![1, N]\!]$ en est l'algorithme. Il faut donc que le compteur parcourt la liste des entiers de 1 à N. Il existe une fonction spécialement adaptée à la manipulation de telles listes d'entiers : la fonction `range`.

2 La fonction `range`

La fonction **`range`** renvoie des listes d'entiers, qui pourront être parcourues par le compteur d'une boucle `for`. La syntaxe est la suivante : pour `a`, `b` et `r` des entiers :
- `range(a,b)` est la liste de tous les entiers de `a` à `b-1`
- `range(a, b, r)` est la liste de tous les termes de la suite arithmétique de premier terme `a` et de raison `r`, jusqu'à la dernière valeur *strictement* plus petite que `b` i.e.

- `range(a)` =

Quelques remarques :
- en informatique, comme pour les étages d'un immeuble, on commence à compter à zéro
- lorsque deux valeurs sont spécifiées dans la fonction `range`, la première est incluse et la deuxième est exclue
- les arguments de la fonction `range` doivent être entiers
- si `n` est un entier, alors `range(n)` contient `n` éléments
- si `a` et `b` sont des entiers tels que `a` $\leqslant$ `b`, alors `range(a, b)` contient `b` $-$ `a` éléments.

Exercice 1
Compléter :
- `range(2,7)` contient :
- `range(1,7,2)` contient :
- Pour $n \in \mathbb{N}$, `range(` `)` et `range(` `)` contiennent : $0,\ 1,\ 2,\ 3,\ \ldots,\ n$
- `range(` `)` contient : $5,\ 2,\ -1,\ -4$.

Attention, insistons encore une fois sur un point source d'erreur : `range(a,b)` contient les entiers de l'intervalle

Exercice 2
Écrire un programme qui affiche tous les entiers pairs entre 0 et $2n$. (On proposera 3 solutions.)

3 Sommes et produits

La boucle for permet de calculer des objets définis par récurrence comme typiquement les sommes. Pour calculer une somme s'écrivant mathématiquement $\sum_{k=p}^{q} f(k)$, on commence par initialiser une variable S à 0 puis on lui ajoute successivement les termes $f(k)$ pour chaque valeur de $k \in [\![p, q]\!]$ (= `range(  )`) :

Exercice 3

1. Écrire une fonction Python `somme1` prenant en argument un entier `n` et renvoyant la valeur de $\sum_{k=1}^{n} k^3$.
2. Écrire une fonction Python `somme2` prenant en argument un entier `n` et renvoyant la valeur de $\sum_{k=0}^{n} \frac{1}{2k+1}$.

Suivi des variables : Considérons la fonction `somme` prenant en argument `n` et renvoyant $S_{\texttt{n}} = \sum_{k=0}^{\texttt{n}} f(k)$ (où f est une fonction fixée, préalablement définie dans Python).
Elle comporte 2 variables dont les valeurs évoluent : S et k. Écrivons les valeurs que prennent ces variables dans un tableau au cours de l'algorithme afin de s'assurer que `somme(n)` renvoie bien $S_{\texttt{n}}$:

Pour finir, on peut calculer de même des produits du type

$$f(p) \times f(p+1) \times \cdots \times f(q-1) \times f(q) = \prod_{k=p}^{q} f(k).$$

Que doit-on changer par rapport à l'algorithme de calcul d'une somme ?

Exercice 4

1. Écrire une fonction `prod1` prenant en argument `n` et renvoyant $\prod_{k=2}^{\texttt{n}} \frac{1+k}{2}$. Que renvoie `prod1(1)` ?
2. Écrire une foncton `prod2` prenant en argument `n` et renvoyant le produit de tous les entiers impairs compris entre 1 et $2\texttt{n}+1$ inclus i.e. $1 \times 3 \times 5 \times \cdots \times (2\texttt{n}-1) \times (2\texttt{n}+1)$.

4 Suites récurrentes

Un exemple important d'utilisation de boucles `for` est le calcul des termes d'une suite définie par récurrence. Dans ce cas, la boucle `for` est précédée d'une étape *d'initialisation*, puis on effectue à l'intérieur de la boucle des étapes *d'update*. On peut le voir comme une généralisation de l'algorithme de calcul d'une somme où l'initialisation consiste à faire `S=0` et l'update `S=S+f(k)`.

4.1 Cas où $u_{n+1} = f(u_n)$

Exercice 5
Écrire une fonction Python `suite1` prenant en argument un entier naturel `n` et renvoyant le terme u_n de la suite $(u_n)_{n\geq 0}$ définie par $u_0 = 1$ et : $\forall n \in \mathbb{N},\ u_{n+1} = u_n^2 + 3$.
On commencera par écrire les opérations *mathématiques* que Python doit faire pour obtenir u_n.

Suivi des variables : Au niveau de $(*)$, les variables en jeu prennent les valeurs suivantes :

Remarque 3

- La variable `u`, même si elle ne change pas de nom, prend successivement plusieurs valeurs u_k de la suite (u_n). Tout comme la variable `k` change elle aussi de valeur dans la boucle. Il n'est pas utile d'appeler cette variable `u_n` (et encore moins u_n !).
- Que contient `range(1,1)` ?
  Que renvoie donc `suite1(0)` ?

Attention, l'étape de "suivi des variables" est indispensable à comprendre pour identifier comment choisir les arguments de la fonction `range`. En effet, ces arguments changent lorsque la suite est initialisée à $n = 0$ ou $n = 1$, et/ou lorsque la relation de récurrence est donnée sous la forme $u_{n+1} = f(u_n)$ ou $u_n = f(u_{n-1})$... Il faut s'adapter à chaque cas en s'assurant que Python effectue bien les opérations *mathématiques* voulues.

Exercice 6

1. Écrire une fonction `suite2` prenant en argument un entier naturel `n` et renvoyant le terme $u_{\texttt{n}}$ de la suite $(u_n)_{n\geq 0}$ définie par $u_0 = 2$ et : $\forall n \in \mathbb{N},\ u_{n+1} = \dfrac{1}{u_n} + u_n$.
2. Écrire une fonction `suite3` prenant en argument un entier $\texttt{n} \in \mathbb{N}^*$ et renvoyant le terme $u_{\texttt{n}}$ de la suite $(u_n)_{n\geq 1}$ définie par $u_1 = 2$ et : $\forall n \in \mathbb{N}^*,\ u_{n+1} = \dfrac{u_n + 1}{2u_n + 3}$.

Dans le cas où la suite (u_n) est définie par une relation du type $u_{n+1} = f(u_n)$, la variable `k` intervenant dans la boucle `for` n'a pas d'importance au sens où par exemple les fonctions suivantes effectuent les mêmes opérations et renvoient donc les mêmes résultats :

```
1 def suite1(n):
2     u = 1
3     for k in range(1,n+1):
4         u = u**2 + 3
5     return u
```

```
1 def suite1(n):
2     u = 1
3     for k in range
4         u = u**2 + 3
5     return u
```

Dans ce cas, seul compte alors le *nombre* de tours de boucles effectués, i.e. le nombre de valeurs prises par `k`.

4.2 Cas où $u_{n+1} = f(u_n, n)$

La variable `k` joue en revanche un rôle important lorsque la relation de récurrence permettant de passer de u_k à u_{k+1} *fait intervenir* `k` !

Exercice 7
Écrire une fonction Python `suite4` prenant en argument un entier naturel `n` et renvoyant le terme u_{n} de la suite $(u_n)_{n\geq 0}$ définie par $u_0 = 6$ et : $\forall n \in \mathbb{N},\ u_{n+1} = u_n + (n+3)^2$.
On commencera par écrire les opérations *mathématiques* nécessaires.

Suivi des variables : Au niveau de $(*)$, les variables en jeu prennent les valeurs suivantes :

Remarque 4
Attention : même si la relation de récurrence est écrite sous la forme $\forall n \in \mathbb{N},\ u_{n+1} = f(u_n, n)$ il faut bien utiliser `k` dans la boucle `for`. En fait on utilise que : $\forall k \in \mathbb{N},\ u_{k+1} = f(u_k, k)$ (ce qui est rigoureusement identique !).

Exercice 8

1. Écrire une fonction Python `suite5` prenant en argument un entier naturel `n` et renvoyant le terme u_n de la suite $(u_n)_{n\geq 0}$ définie par $u_0 = 3$ et : $\forall n \in \mathbb{N},\ u_{n+1} = \dfrac{u_n + 3n}{2n+1}$.
2. Écrire une fonction Python `suite6` prenant en argument un entier naturel `n` et renvoyant le terme u_n de la suite $(u_n)_{n\geq 1}$ définie par $u_1 = 6$ et : $\forall n \in \mathbb{N}^*,\ u_{n+1} = (u_n + n + 1)^2$.

4.3 Cas des suites récurrentes doubles

Le cas des suites récurrentes doubles, c'est-à-dire définies par une relation du type $u_{n+2} = f(u_n, u_{n+1})$ peut sembler au premier abord différent de celui des suites récurrentes simples. Toutefois on peut toujours se ramener à ce dernier en considérant une variable "à deux dimensions".
Prenons un premier exemple : soit $(u_n)_{n\geq 0}$ la suite définie par

$$u_0 = 1,\ u_1 = 2 \quad \text{et} \quad \forall n \in \mathbb{N},\ u_{n+2} = 2u_{n+1} - 3u_n.$$

On peut reformuler cette relation du type $u_{n+2} = f(u_n, u_{n+1})$ en une relation du type $U_{n+1} = F(U_n)$ où $U_n = (u_n, u_{n+1})$ en l'écrivant :

Il s'agit alors de travailler sur le *couple* (u_n, u_{n+1}) au lieu de travailler simplement sur le réel u_n. On applique alors les mêmes règles "d'initialisation et d'update" que précédemment.

Exercice 9

1. Écrire une fonction `Fibo` prenant en argument un entier naturel `n` ≥ 1 et renvoyant le terme u_{n} de la suite de Fibonacci définie par $u_0 = 0,\ u_1 = 1$ et : $\forall n \in \mathbb{N},\ u_{n+2} = u_{n+1} + u_n$.
2. Écrire une fonction `suite7` prenant en argument un entier naturel `n` ≥ 1 et renvoyant le terme u_{n} de la suite définie par $u_0 = 2,\ u_1 = 3$ et : $\forall n \in \mathbb{N},\ u_{n+2} = \sqrt{u_n + u_{n+1}^2}$.

5 Entraînement

Exercice 10

1. Écrire une fonction `somme1` prenant en argument un entier `n` ≥ 5 et renvoyant $\sum_{k=5}^{\text{n}} \frac{\sqrt{k}}{k+1}$.
2. Écrire une fonction Python `somme2` prenant en argument un entier naturel `n` ≥ 3 et un réel `x` et renvoyant la valeur de $S_n(x) = \sum_{k=2}^{n-1} x^k$.
3. Écrire une fonction Python `prod1` prenant en argument un entier naturel `n` ≥ 1 et un réel `x` et renvoyant la valeur de $P_n(x) = \prod_{k=1}^{n+1} x^k$.

Exercice 11

1. Écrire une fonction `somme3` prenant en argument un entier `n` ≥ 3 et renvoyant $\displaystyle\sum_{k=2}^{n-1} \frac{1}{k+n}$.
2. Écrire une fonction `prod2` prenant en argument deux entiers positifs `n` et `m` et renvoyant la valeur de $\displaystyle\prod_{j=0}^{n} \frac{j^m + n}{m^2 + j + 1}$

Exercice 12

Pour chacune des suites (u_n) définies par récurrence ci-dessous, écrire une fonction Python prenant en argument un entier naturel `n` et renvoyant u_{n}. Vous testerez à la main que l'algorithme proposé renvoie le bon résultat pour les premières valeurs de `n`.

1. $u_0 = 2$ et $\forall n \geq 0,\ u_{n+1} = 2u_n - 1$
2. $u_0 = 2$ et $\forall n \geq 1,\ u_n = 3u_{n-1}^2$
3. $u_0 = u_1 = 4$ et $\forall n \geq 2,\ u_n = u_{n-1}^3$
4. $u_0 = 2$ et $\forall n \geq 0,\ u_{n+1} = n + u_n$
5. $u_0 = 2$ et $\forall n \geq 1,\ u_n = (n+1)u_{n-1}^2$
6. $u_0 = u_1 = 4$ et $\forall n \geq 2,\ u_n = u_{n-1}^2 + u_{n-2}^2$