3.1 *Boucle bornée (« Pour »)*

Une boule bornée (ou boucle « Pour ») est une boucle dans laquelle des instructions identiques sont répétées un nombre de fois déterminé à l'avance.

Pour répéter 10 fois un bloc d'instructions, la syntaxe est la suivante.

en langage naturel :
Pour k variant de 0 à 9 :
bloc d'instructions
Fin pour

en Python :

```
for k in range(10) :
    instructions
```

Dans cette syntaxe, `k` est une variable dont le nom peut être choisi arbitrairement.

Remarque 1 Comme pour l'instruction conditionnelle, les deux points à la fin de la première ligne et l'indentation sont indispensables.

Il existe différentes syntaxes pour `for k in range(...) :`

- `for k in range(n) : :k` est une variable de type `int` qui prend successivement les valeurs `0`, `1`, ..., `n - 1`.
- `for k in range(a,b) : :k` est une variable de type `int` qui prend successivement les valeurs `a`, `a + 1`, ..., `b - 1`.
- `for k in range(a,b,p) : :k` est une variable de type `int` qui prend successivement les valeurs `a`, `a + p`, `a + 2p`, `a + 3p`, ..., jusqu'à atteindre la plus grande valeur possible strictement inférieure à `b` (ici, le pas est donc `p` et non pas `1` comme dans la syntaxe précédente).

Par exemple, la fonction suivante affiche tous les entiers pairs de `0` à `2n` pour un entier naturel `n` passé en argument.

```
def nb_pairs(n):
    for k in range (n +1):
        print(2*k)
```

Les boucles bornées sont particulièrement adaptées au calcul des termes d'une suite récurrente. Considérons, par exemple, la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbf{N}$, $u_{n+1} = 2u_n + n$. La fonction suivante renvoie la valeur de u_n pour un entier naturel n passé en argument.

```
def suite(n):
    u = 1
    for k in range(n):
        u = 2*u+k
    return u
```

3.2 *Boucle non bornée (« Tant que »)*

Une boule non bornée (ou boucle « Tant que ») est une boucle dans laquelle un bloc d'instructions est répété tant qu'une certain condition C est vraie. La boucle s'arrête dès que la condition C est fausse.

La syntaxe est la suivante.

en langage naturel :
Tant que (condition C)
bloc d'instructions
Fin Tant que

en Python :

```
while (condition C):
    instructions
```

Remarque 2 Comme pour une boucle bornée, les deux points à la fin de la première ligne et l'indentation sont indispensables. La condition C peut être n'importe quelle expression de type `bool`.

Ce peut être une comparaison du type `A == 3` ou `A > 0` mais ce peut aussi être le résultat renvoyé par une fonction si celui-ci est un booléen.

Les boucles non bornées sont particulièrement adaptées à la recherche de seuil, notamment pour les suites.

Considérons, par exemple, la suite (u_n) définie par $u_0 = 1$ et, pour tout $n \in \mathbf{N}$, $u_{n+1} = \dfrac{u_n}{n+1}$.

La fonction suivante renvoie la plus petite valeur de l'entier naturel n tel que $u_n \leqslant$ eps où eps est un flottant passé en argument.

```
def seuil (eps):
    u=1
    n=0
    while (u > eps):
        u = u/(n +1)
        n = n+1
    return n
```

La condition dans la boucle while est le contraire de ce qu'on cherche puisque le calcul continue tant qu'on n'a pas atteint de seuil voulu. On pourrait d'ailleurs remplacer la condition `(u > eps)` par `(not u <= eps)`. Dans cette fonction, n est une variable qui augmente de 1 à chaque tour de boucle, on dit qu'elle est incrémentée de 1 à chaque tour.

On l'a ici implémenté par `n = n+1`. Une autre syntaxe possible est `n += 1`.

3.3 *Exercices*

♦ **PYT.1** On considère l'algorithme suivant.

$S \leftarrow 0$
Pour i allant de 1 à 10
  $S \leftarrow S + i^2$
Fin pour

1) Quelle est la fonction de cet algorithme ?

2) Traduire cet algorithme en un programme Python puis implémenter ce programme.

♦ **PYT.2** La fonction suivante est censée calculer la somme des puissances de 3 de 3^4 jusqu'à 3^n où n est un entier supérieur ou égal à 4 passé en argument.

```
def somme_puissance_trois(n):
    S=0
    for j in range(4,n):
        S=S+3^j
    return S
```

1) Il y a plusieurs erreurs dans ce programme. Les trouver et les corriger.

2) Implémenter cette fonction corrigée.

♦ **PYT.3** Écrire une fonction qui prend deux entiers naturels non nuls n et p en argument et qui renvoie la somme des puissances p-ième des entiers de 1 à n, c'est-à-dire $\sum_{k=1}^{p} k^p$.

♦ **PYT.4**

1) Écrire une fonction `mult_7` qui renvoie le nombre de multiples de 7 compris entre 1 et n, où n est un entier naturel non nul passé en argument.

☞ **Indications exercice 4 :** Un entier n est un multiple de 7 si son reste dans la division euclidienne par 7 est nul.

2) Écrire une fonction `mult_7_pas_3_5` qui prend en argument un entier naturel n non nul et qui renvoie le nombre d'entiers compris entre 1 et n qui sont divisibles par 7 mais pas par 3 ni par 5.

♦ **PYT.5** On dit qu'un entier naturel n est parfait si la somme de ses diviseurs positifs est égale à $2n$. Par exemple, 6 est parfait car les diviseurs positifs de 6 sont 1, 2, 3 et 6 et $1+2+3+6=12=2\times 6$.

Écrire une fonction `est_parfait` qui renvoie le nombre d'entiers parfaits compris entre 1 et n, où n est un entier naturel non nul passé en argument.

♦ **PYT.6** On souhaite écrire une fonction `factorielle` prenant en argument un entier naturel n et renvoyant la valeur de $n!$.

1) Expliquer pourquoi la fonction suivante ne convient pas.

```
def factorielle(n):
    fact=1
    for i in range(n+1):
        fact = fact*i
    return fact
```

2) Corriger la fonction précédente pour obtenir le résultat voulu.

♦ **PYT.7**

1) Soit (u_n) la suite définie par $u_0=4$ et, pour tout $n\in\mathbf{N}$, $u_{n+1}=2-\dfrac{u_n}{2}$. Écrire une fonction `suite_u` qui renvoie le terme d'indice n de la suite (u_n) pour un entier naturel n passé en argument.

2) Soit (F_n) la suite définie par $F_0=0$, $F_1=1$ et, pour tout $n\in\mathbf{N}$, $F_{n+2}=F_{n+1}+F_n$. Écrire une fonction `suite_F` qui renvoie le terme d'indice n de la suite (F_n) pour un entier naturel n passé en argument.

♦ **PYT.8** On considère l'algorithme suivant.

$A \leftarrow 1$
Tant que $A < 10$
  Afficher A
  A = A+2
Fin Tant que

1) Quelle est la fonction de cet algorithme ?

2) Traduire cet algorithme en un programme Python puis implémenter ce programme.

♦ **PYT.9** On considère l'algorithme suivant.

$k \leftarrow 0$
Tant que $k^2 < 1000$
  $k \leftarrow k+1$
Fin Tant que

1) Quelle est la fonction de cet algorithme ?

2) Traduire cet algorithme en un programme Python puis implémenter ce programme.

♦ **PYT.10** La fonction suivante est censée calculer la somme des nombres impairs inférieurs à n pour un entier naturel n passé en argument.

1) Il y a plusieurs erreurs dans le script de cette fonction.
Les trouver et les corriger.

2) Implémenter cette fonction.

```
def somme_impairs(n):
    k = 1
    S = 0
    while (S < n)
        k += 2
        S = S+k
    return S
```

♦ **PYT.11** Sur un compte en banque, on place 1000 euros. Le compte rapporte2% d'intérêts par an. Ainsi, au bout d'un an, il y aura sur le compte $1000 + \frac{2}{100} \times 1000 = 1020$ euros.

Écrire une fonction `nb_années` qui renvoie le nombre d'années nécessaires pour que le montant sur le compte dépasse strictement P euros où P est un nombre strictement positif passé en argument. On suppose que le taux reste à 2% et qu'on n'ajoute rien d'autre sur le compte que les intérêts annuels.

♦ **PYT.12** Écrire une fonction `plus_gd_carré` qui renvoie le plus grand entier inférieur ou égal à n qui est le carré d'un entier, où n est un entier naturel passé en argument.

♦ **PYT.13** Écrire une fonction `comptage` qui renvoie le nombre d'entiers naturels non nuls k tels que k^k est inférieur ou égal à n où n est un entier naturel non nul passé en argument.

♦ **PYT.14** Pour $n \in \mathbf{N}$, on pose $S_n = \sum_{1 \leqslant i,j \leqslant n} |i-j|$. Écrire une fonction `somme_diff_abs` qui renvoie la valeur de S_n pour un entier naturel n non nul passé en argument. On pourra utiliser la fonction `valeur_absolue` de l'exercice 8.

♦ **PYT.15** On considère la fonction `mystere` suivante.

1) Déterminer, sans l'implémenter, la valeur renvoyée par `mystère(1234)`.

2) Si n est un entier naturel, que représente la valeur renvoyée par `mystere(n)` ?

3) En s'inspirant de la fonction `mystere`, écrire une fonction `nb_chiffres` qui renvoie le nombre de chiffres d'un entier naturel n passé en argument.

```
def mystere(n):
    s = 0
    while n > 0
        s = s+(n%10)
        n = n//10
    return s
```