Rappels d'algorithmique – Partie 2

La boucle bornée

Définition : Lorsque l'on doit répéter un nombre de fois *connu à l'avance* la même tâche, on utilise une boucle bornée de la forme « Pour.. allant de... à ».

Dans un algorithme, cette structure est codée de la façon suivante :

```
Pour variable allant de valeur_depart à valeur_fin faire
    tâche 1
    tâche 2
    ...
Fin pour
```

La variable utilisée dans la boucle est appelée compteur. À chaque passage dans la boucle, sa valeur est automatiquement augmentée de 1.

Remarques :

1. En Python, une boucle bornée se code de la façon suivante :

```
for i in range(a,b) :

    tâche 1
    tâche 2
    ...
```

   Attention ! Dans ce cas, la valeur de fin du compteur i sera $b-1$, pas b. Sous python, la dernière boucle n'est pas exécutée.

2. En Python, on peut aussi utiliser l'instruction simplifiée `in range(b)`, qui correspond à `in range(0,b)`, c'est-à-dire que le compteur prend toutes les valeurs entières entre 0 et $b-1$.

Exemple :

L'algorithme 1 (et le programme Python de l'algorithme 2) affiche la table de multiplication (de 0 à 10) d'un nombre entier donné.

Algorithme 1 Table de multiplication

```
n ⟵ valeur saisie
Pour i allant de 0 à 10 faire
    m ⟵ n × i
    Afficher n ,« x », i, « = », m
Fin Pour
```

Algorithme 2 Programme Python : table de multiplication

```
n = int(input())
for i in range (0,11) :
    m = n*i
    print(n," x ",i," = ",m)
```

La boucle non bornée

Définition : Dans l'exécution d'un algorithme, on peut être amené à réaliser plusieurs fois la même tâche, sans savoir à priori combien de fois cette tâche doit être réalisée. On répète alors les instructions tant qu'une condition est remplie.

On utilise alors une boucle non bornée qui est codée de la façon suivante dans un algorithme :

```
Tant que condition  faire
    tâche 1
    tâche 2
    ...
Fin Tantque
```

Remarque : En Python, une fonction se code de la façon suivante :

```
while condition :
    tâche 1
    tâche 2
    ...
```

Exemple : L'algorithme 3 (et le programme Python donné à l'algorithme 4) affiche la plus petite puissance de 2 supérieure à 10 000.

Algorithme 3 Puissance de 2 supérieure à 10 000

```
puissance ⟵1
Tant que puissance ≤ 10000 faire
    puissance ⟵ puissance × 2
Fin Tantque
Afficher(puissance)
```

Algorithme 4 Programme Python : Puissance de 2 supérieure à 10 000

```
puissance = 1
while puissance <= 10000 :
    puissance = puissance*2
print(puissance)
```

Exercices

Exercice 1

Soit (u_n) la suite définie par $u_n = 3 + \dfrac{1}{n+1}$

1. Écrire un programme Python qui affiche les 15 premiers termes de la suite sous la forme suivante :

```
u(0) = ...
u(1) = ...
...
```

2. Que peut-on conjecturer pour le sens de variations de cette suite ?
   Prouver ensuite ce résultat.
3. On admet que cette suite admet comme limite 3.
   Écrire une fonction Python `seuil_u(e)` qui renvoie le plus petit entier n tel que $3 < u_n \leq 3 + e$.
   Tester cette fonction avec des valeurs de `e` de plus en plus petites.

Exercice 2

On considère la suite (v_n) définie par :

$$\begin{cases} v_0 = 1 \\ v_{n+1} = -\frac{2}{3}v_n + 4 \end{cases}$$

1. Écrire la fonction Python `v(n)` qui renvoie la valeur du terme de rang n de la suite v_n.
2. En testant la fonction `v(n)` pour différentes valeurs de n, conjecturer le sens de variation et la limite l de cette suite.
3. En vous inspirant de l'exercice 1, écrire une fonction Python `seuil_v(e)` qui renvoie le plus petit entier n tel que $l - e < v_n < l + e$.
   Tester cette fonction avec des valeurs de `e` de plus en plus petites.

Exercice 3

On considère la suite (w_n) définie par :

$$\begin{cases} w_0 = 1 \\ w_{n+1} = 3w_n - 2 \end{cases}$$

1. Écrire la fonction Python `w(n)` qui renvoie la valeur du terme de rang n de la suite w_n.
2. En testant la fonction `w(n)` pour différentes valeurs de n, conjecturer le sens de variation et la limite de cette suite.
3. En vous inspirant de l'exercice 2, écrire une fonction Python `seuil_w(M)` qui renvoie le plus petit entier n tel que $w_n \geq M$.
   Tester cette fonction avec des valeurs de `M` de plus en plus grandes.

Exercice 4

On considère la suite (t_n) définie par :

$$\begin{cases} t_0 = 0 \\ t_{n+1} = t_n + 2n + 1 \end{cases}$$

1. Vérifier en calculant à la main que $t_1 = 1$ et $t_2 = 4$.
2. Écrire la fonction Python `t(n)` qui renvoie la valeur du terme de rang n de la suite t_n.
3. Tester la fonction `t(n)` pour différentes valeurs de n. Que peut-on conjecturer ?
4. Montrer cette conjecture.

TP : Longueur d'un arc de parabole

Calcul de la longueur d'un arc de parabole

On considère la fonction f définie sur $\mathbb{R}$ par $f(x)=x^2$. On note $\mathscr{P}$ la parabole représentant la fonction f dans le plan rapporté à un repère orthonormé.
Le but de ce TP est d'obtenir une valeur approchée de la longueur de l'arc de la parabole $\mathscr{P}$ sur l'intervalle $[0;1]$.

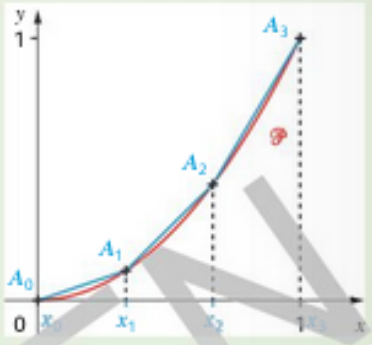
Objectif PYTHON
Comprendre, compléter et programmer un algorithme. Faire le lien entre langage naturel et le langage Python.

Description de la méthode

Soit n un entier naturel non nul. On découpe l'intervalle $[0;1]$ en n intervalles de même amplitude.
On note $x_0, x_1, \ldots, x_n$ les bornes de ces intervalles et $A_0, A_1, \ldots, A_n$ les points de la parabole $\mathscr{P}$ d'abscisses respectives $x_0, x_1, \ldots, x_n$.
On approche la longueur de l'arc de la parabole $\mathscr{P}$ sur l'intervalle $[0;1]$ par la somme L des longueurs des segments $[A_0A_1], [A_1A_2], \ldots, [A_{n-1}A_n]$:

$$L = A_0A_1 + A_1A_2 + \ldots + A_{n-1}A_n$$

La figure ci-contre illustre la situation dans le cas où $n=3$.



Partie A Comprendre et exécuter un algorithme

1. Dans cette question, on se place dans le cas où $n=3$.
a. Donner les valeurs de x_0, x_1, x_2 et x_3, ainsi que les coordonnées des points A_0, A_1, A_2 et A_3.
b. Calculer les valeurs exactes des longueurs A_0A_1, A_1A_2 et A_2A_3.

2. On considère l'algorithme ci-contre qui définit la fonction **Longueur(n)**, dans lequel la variable L est un nombre réel et la variable n un entier naturel non nul.

```
Fonction Longueur(n)
L ← 0
Pour k allant de 0 à n−1 Faire
    L ← L + A_k A_{k+1}
Fin Pour
Fin Fonction
```

a. Que permet de calculer la fonction **Longueur(n)** ?
b. Exécuter à la main la fonction **Longueur(n)** pour $n=3$. Donner une valeur approchée à 10^{-3} près de la valeur contenue dans la variable L à la fin de l'exécution de l'algorithme. Interpréter cette valeur.

Partie B Longueur entre deux points consécutifs de la ligne brisée

Dans cette partie, n est un entier naturel non nul.
1. Que valent x_0 et x_n ?
2. Quelle est l'amplitude de chacun des intervalles $[x_0;x_1], [x_1;x_2], \ldots, [x_{n-1};x_n]$?
3. Pour k entier compris entre 0 et n, exprimer x_k en fonction de k et de n.
4. Pour k entier compris entre 0 et n, déterminer les coordonnées des points A_k en fonction de k et de n.
5. Pour k entier compris entre 0 et $n-1$, montrer que :

$$A_kA_{k+1} = \sqrt{\frac{1}{n^2} + \left(f\left(\frac{k+1}{n}\right) - f\left(\frac{k}{n}\right)\right)^2}$$

Partie C Compléter et comprendre un algorithme

1. Écrire en Python une fonction `f(x)` qui renvoie l'image d'un nombre x par la fonction f.
2. En utilisant la fonction précédente, compléter le programme suivant qui permet de définir la fonction `Longueur(n)` donnée dans la partie A.

```
def Longueur(n):
    L=0
    for k in range(n):
        L=...
    return L
```

3. Programmer et exécuter la fonction `Longueur(n)` et compléter le tableau suivant en arrondissant les valeurs à 10^{-5}.

n	50	100	300	500	1 000
$L(n)$					

4. Quelle valeur approchée de la longueur de l'arc de la parabole $\mathscr{P}$ sur $[0;1]$ cette méthode permet-elle de donner ?