TP Informatique n° 2
Révisions sur les structures itératives
(illustrations sur les suites et les séries)

I. Les structures de contrôle

En informatique, on appelle **structure de contrôle** une commande qui contrôle l'ordre dans lequel les différentes instructions d'un programme sont exécutées.
On peut distinguer plusieurs types de structures de contrôle :

a) les **structures séquentielles** : les différentes commandes sont exécutées les unes à la suite des autres *i.e.* dans un ordre séquentiel.

b) les **structures conditionnelles** : qui permettent d'introduire des branchements conditionnels dans le programme. Un programme peut comporter plusieurs branches. Chacune d'elle est associée à une condition. La branche exécutée est la première dont la condition est réalisée.

c) les **structures itératives** : qui permettent d'effectuer la répétition (*i.e.* l'itération) de commandes. Ces répétitions peuvent s'effectuer :

- × un nombre de fois fixé explicitement par le programmeur.
  On utilise alors une boucle `for`.
- × ou peuvent avoir lieu tant qu'une condition n'est pas réalisée.
  On utilise alors une boucle `while`.

II. La boucle `for`

II.1. Syntaxe générale

Une boucle for permet de réaliser la répétition d'un bloc d'instructions.
En **Python**, la syntaxe classique de cette commande est la suivante.

```
for i in range(n):
    instruction
```

▶ Écrire un programme permettant d'afficher un à un les éléments de `range(5)`.

```
1 for i in range(5):
2     print(i)
```

▶ Quel est le résultat de l'exécution de ce programme ? De manière générale que produit l'instruction `range(n)` ?

- On obtient l'affichage de 0, 1, 2, 3, 4.
- L'instruction `range(n)` permet de produire la liste des entiers de 0 à $n-1$.

▶ En procédant de même, décrire ce que produit les instructions `range(3,9)`, puis `range(3,9,2)` et `range(9,3,-1)`.

- Lorsque l'on applique la procédure précédente avec l'instruction `range(3,9)`, on obtient l'affichage des entiers successifs de 3 à 8.
- Avec l'instruction `range(3,9,2)`, on obtient l'affichage des entiers de 3 à 8 (au maximum) avec un pas de 2.
- Avec l'instruction `range(9,3,-1)`, on obtient l'affichage des entiers de 9 à 4 (au minimum) avec un pas de -1.

▶ Une spécificité du langage **Python** est qu'il est possible d'itérer sur de nombreux objets. En utilisant le procédé précédent, écrire une itération sur la chaîne de caractère `"ma chaîne"` permettant d'afficher un à un les éléments de cet objet. Qu'obtient-on ?

```
1 for elt in "ma chaîne":
2     print(elt)
```

On obient l'affichage successif de chaque lettre de cette chaîne de caractère.

II.2. Calcul du $m^{\text{ème}}$ élément d'une suite

On considère la suite $(u_n)_{n\in\mathbb{N}^*}$ définie par :

$$\begin{cases} \forall n \in \mathbb{N}^*,\ u_{n+1} = 2\,u_n + n + 1 \\ u_1 = 1 \end{cases}$$

▶ Écrire un programme qui :

× demande initialement à l'utilisateur d'entrer au clavier la valeur d'un entier `m`,

× affiche la valeur du $\texttt{m}^{\text{ème}}$ élément de la suite (u_n).

```
1 m = int(input("Prière d'entrer un entier m : "))
2 u = 1
3 for i in range(1,m)
4     u = 2 * u + i + 1
5
6 print(u)
```

▶ Calculer u_{12} et u_{20} à l'aide du programme précédent.

On obtient $u_{12} = 8178$ et $u_{20} = 2097130$.

- Modifier le programme précédent afin d'obtenir une fonction `emeSuiteU` qui :
  - × prend en paramètre une variable `m`,
  - × renvoie le $\texttt{m}^{\text{ème}}$ élément de la suite (u_n).

```
def emeSuiteU(m)
    u = 1
    for i in range(m-1):
        u = 2 * u + i + 1
    return(u)
```

- Calculer u_7 et u_{15} à l'aide de la fonction précédente.

On obtient $u_7 = 247$ et $u_{15} = 65519$.

- Selon vous, quels sont les avantages de la représentation sous forme de programme avec dialogue utilisateur ? Sous forme de fonction ?

- D'un point de vue utilisateur, la version avec dialogue utilisateur, plus ludique, peut être plus appréciée.
- D'un point de vue algorithmique, il faut privilégier la version sous forme de fonction. L'avantage est que le calcul réalisé par `emeSuiteU` peut facilement être utilisé ailleurs (notamment dans une autre fonction) : il suffit pour ce faire d'écrire l'appel `emeSuiteU(m)` (avec `m` choisi correctement).

II.3. Calcul des m premiers éléments d'une suite

- Écrire un programme qui :
  - × demande initialement à l'utilisateur d'entrer au clavier la valeur d'un entier `m`,
  - × affiche la liste des des `m` premiers éléments de la suite (u_n).

```
m = int(input("Prière d'entrer un entier m : "))
l = [1]
for i in range(1,m):
    l = l + [2 * l[i-1] + i + 1]
    # ou l.append(2 * l[i-1] + i + 1)

print(l)
```

- Calculer les 5 premiers éléments de la suite à l'aide du programme précédent.

On obtient : $[1, 4, 11, 26, 57]$.

- Modifier ce programme afin d'obtenir une fonction `premSuiteU` qui :
  - × prend en paramètre une variable `m`,
  - × renvoie la liste des `m` premiers éléments de la suite (u_n).

```
def premSuiteU(m):
    l = [1]
    for i in range(1,m):
        l = l + [2 * l[i-1] + i + 1]
        # ou l.append(2 * l[i-1] + i + 1)
    return(l)
```

II.4. Calcul des sommes partielles d'ordre n

On considère maintenant les suites $(u_n)_{n\in\mathbb{N}}$ et (v_n) suivantes :

$$\forall n \in \mathbb{N},\ u_n = \frac{n^2}{3^n} \qquad \text{et} \qquad \begin{cases} v_0 = 1 \\ \forall n \in \mathbb{N},\ v_{n+1} = \dfrac{2\,v_n}{\mathrm{e}^{v_n} + \mathrm{e}^{-v_n}} \end{cases}$$

On notera par la suite $S_n = \sum_{k=0}^{n} u_k$ et $T_n = \sum_{k=0}^{n} v_k$ les sommes partielles d'ordre n des séries $\sum u_n$ et $\sum v_n$.

II.4.a) Calcul de S_n

- Écrire une fonction `calculSn` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie la valeur de $S_{\texttt{n}}$.

```
def calculSn(n):
    S = 0
    for i in range(n+1):
        S = S + i**2 / (3**i)
    return(S)
```

- Que vaut S_0 ? S_5 ? S_{10} ? S_{100} ? S_{1000} ?

On trouve : $S_0 = 0$, $S_5 \simeq 1.4115$, $S_{10} \simeq 1.4989$, $S_{100} \simeq 1.5$ et $S_{1000} \simeq 1.5$.

II.4.b) Calcul de T_n

Il s'agit ici d'illustrer le cas où la suite (u_n) est donnée sous forme récurrente.

- Écrire une fonction `calculTn` qui :
  - × prend en paramètre un entier `n`,
  - × renvoie la valeur de T_n.

```python
import math

def calculTn(n):
    T = 1
    v = 1
    for i in range(n):
        v = 2 * v / (math.exp(v) + math.exp(-v))
        T = T + v
    return(T)
```

- Que vaut T_0 ? T_{100} ? T_{10000} ?

On trouve : $T_0 = 1$, $T_{100} \simeq 18.18$, $T_{10000} \simeq 197.58$.

II.5. Calcul des n premières sommes partielles

II.5.a) Calcul des n premières sommes partielles de $\sum u_n$

- Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre S_n et S_{n+1} ?

$$S_{n+1} = S_n + \frac{(n+1)^2}{3^{n+1}}$$

- En tirant profit de l'égalité précédente, écrire une fonction `premS` qui :
  - × prend en paramètre une variable `n`,
  - × renvoie la liste des `n` premières valeurs de la suite (S_n).

  On ne devra pas effectuer d'appel à `calculSn`.

```python
def premS(n):
    lS = [0]
    for i in range(n):
        lS = lS + [lS[i] + (i+1)**2 / (3**(i+1))]
        # ou lS.append(lS[i] + (i+1)**2 / (3**(i+1)))
    return(lS)
```

- Quelles sont les 5 premières valeurs de la suite (S_n) ?

On trouve approximativement : 0.33, 0.77, 1.11, 1.3, 1.4.

II.5.b) Calcul des n premières sommes partielles de $\sum v_n$

▶ Soit $n \in \mathbb{N}$. Quel lien y a-t-il entre T_n et T_{n+1} ?

$$T_{n+1} = T_n + v_{n+1}$$

▶ En tirant profit de l'égalité précédente, écrire une fonction `premT` qui :

× prend en paramètre une variable `n`,

× renvoie la liste des `n` premières valeurs de la suite (S_n).

On ne devra pas effectuer d'appel à `calculTn`.

```
def premT(n):
    lT = [1]
    v = 1
    for i range(n)
        v = 2 * v / (math.exp(v) + math.exp(-v))
        lT = lT + [lT[i] + v]
        # ou lT.append(lT[i] + v)
    return(lT)
```

III. La boucle `while`

III.1. Calcul du premier entier tel qu'une condition est vérifiée

Le problème qui nous intéresse ici est le suivant.

Problème.

Données :

- Une suite (u_n) et une suite (d_n) telle que $d_n \underset{n\to+\infty}{\longrightarrow} 0$.
- L'existence d'un réel α tel que : $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant d_n$.

But :

1) Déterminer un indice N tel que le terme u_N vérifie : $|u_N - \alpha| \leqslant 10^{-4}$.

2) En déduire une valeur approchée de α à 10^{-4} près.

III.2. Un exemple classique

On commence par illustrer le problème et sa résolution par l'étude d'une suite de type $u_{n+1} = f(u_n)$ dans le cadre de l'utilisation de l'inégalité des accroissements finis.

On considère la fonction $f : x \mapsto \mathrm{e}^{-\frac{x^2}{2}}$ et on définit la suite (u_n) par :

$$\begin{cases} u_0 = \frac{1}{2} \\ \forall n \in \mathbb{N},\ u_{n+1} = f(u_n) \end{cases}$$

Rappelons les différentes étapes de ce type d'étude. Les démonstrations sont laissées au lecteur.

1) En appliquant le théorème de la bijection à la fonction $g : x \mapsto f(x) - x$, on démontre que l'équation $f(x) = x$ admet une unique solution dans $[0, 1]$, que l'on note α.

2) Après avoir démontré que l'intervalle $[0, 1]$ est stable par f, on en déduit, par récurrence que : $\forall n \in \mathbb{N},\ u_n \in [0, 1]$.

3) ***a)*** Par étude de la fonction f', on démontre : $\forall x \in [0, 1],\ |f'(x)| \leqslant \dfrac{1}{\sqrt{\mathrm{e}}}$.

b) On est alors dans le cadre de l'application de l'IAF, qui permet de démontrer que :

$$\forall n \in \mathbb{N},\ |u_{n+1} - \alpha| \leqslant \frac{1}{\sqrt{\mathrm{e}}} |u_n - \alpha|$$

(on démontre en fait que $|f(u_n) - f(\alpha)| \leqslant \dfrac{1}{\sqrt{e}} |u_n - \alpha|$*)*

c) On en déduit que : $\forall n \in \mathbb{N},\ |u_n - \alpha| \leqslant \left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n$.

d) Comme $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \underset{n\to+\infty}{\longrightarrow} 0$, on en déduit que (u_n) est convergente, de limite α.

Le but est alors de calculer une valeur approchée de α à 10^{-4} près.

▶ Quelle condition permet d'assurer que $|u_n - \alpha| \leqslant 10^{-4}$?

Si $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$, on en déduit par transitivité que $|u_n - \alpha| \leqslant \left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

▶ Écrire un programme permettant d'afficher le premier entier n tel que $\left(\dfrac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

```
n = 0
while (1 / (math.sqrt(math.exp(1))))**n > 10**(-4):
    n = n+1

return("La valeur de n est : ", n)
```

on peut améliorer cette version en calculant au fur et à mesure $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n$:

```
n = 0
aux = 1
while aux > 10**(-4)
    aux = aux * 1/(math.sqrt(math.exp(1)))
    n = n+1

print("La valeur de n est : ", n)
```

▶ Compléter le programme précédent afin qu'il affiche une valeur approchée à 10^{-4} près de α.

```
n = 0
u = 1/2
while (1 / (math.sqrt(math.exp(1))))**n > 10**(-4)
    n = n+1
    u = math.exp(-u**2 / 2)

print("La valeur de n est : ", n)
print("alpha peut être approché par : ", u)
```

(on aurait pu, comme précédemment, calculer $(\frac{1}{\sqrt{e}})^n$ par multiplications successives)

▶ Déterminer une formule mathématique donnant le premier entier n tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$.

$$\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4} \Leftrightarrow n\ln\left(\frac{1}{\sqrt{\mathrm{e}}}\right) \leqslant -4\ln(10) \quad \textit{(par stricte croissance de la fonction } \ln\textit{)}$$

$$\Leftrightarrow -n\ln\left(\sqrt{\mathrm{e}}\right) \leqslant -4\ln(10)$$

$$\Leftrightarrow n \geqslant \frac{\cancel{-}4\ln(10)}{\cancel{-}\ln\left(\sqrt{\mathrm{e}}\right)} = \frac{4\ln(10)}{\ln(\mathrm{e}^{\frac{1}{2}})} = \frac{4\ln(10)}{\frac{1}{2}\ln(\mathrm{e})} = 8\ln(10)$$

Ainsi, le premier entier tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant 10^{-4}$ est le premier entier tel que : $n \geqslant 8\ln(10)$. L'entier cherché est donc $\lceil 8\ln(10)\rceil$.

▶ Comparer la valeur obtenue dans la quesiton précédente et celle affichée par le programme.

L'instruction `ceil(8 * log(10))` fournit bien le résultat 19 qui correspond à la valeur affichée par le programme.

▶ De même, donner la formule permettant d'obtenir le premier entier n tel que $\left(\frac{1}{\sqrt{\mathrm{e}}}\right)^n \leqslant \varepsilon$.

Par une étude similaire, on trouve : $\lceil -2\ln(\varepsilon)\rceil$.

▶ En déduire une fonction `calcApproch` qui prend en paramètre un réel `eps` et qui renvoie une valeur approchée de α à `eps` près à l'aide d'une boucle `for`.

```
def calcApproch(eps):
    u = 1/2
    n = math.ceil(-2 * math.log(eps))
    for i in range(n)
        u = math.exp(-u**2 / 2)

    return(u)
```