TP 15 : Preuves de programmes

Exercice 1.

(a) Écrire une fonction **`nocc(x,L)`** qui calcule le nombre de fois où x apparait dans la liste L. *On utilisera une boucle while et on donnera un invariant de boucle permettant de démontrer que cette fonction est correcte.*

(b) Écrire une fonction **`majoritaire(L)`** qui détermine un élément x qui soit majoritaire dans L (autrement dit qui apparait plus souvent dans L que tout autre élément de L). *On utilisera une boucle while et on donnera un invariant de boucle permettant de démontrer que cette fonction est correcte.*

Exercice 2 *Calcul rapide de puissances.* On donne l'algorithme suivant :

(a) Détailler les valeurs successives prises par les variables x, y et n lorsque l'on appelle cet algorithme avec $A = 2$ et $N = 21$. Mettre en évidence le résultat renvoyé.

(b) Programmer cet algorithme sous la forme d'une fonction **`puiss_rapide(A,N)`** et vérifier que le fonctionnement est bien celui attendu.

(c) Démontrer que la fonction termine pour toute valeur de A et tout entier $N \geqslant 0$.

(d) Justifier que la propriété $A^N = y \times x^n$ est un invariant de boucle. Quelle est alors la valeur de y au point 4 ?

(e) Pour calculer la complexité, on va considérer un entier p tel que $2^p \leqslant N < 2^{p+1}$. Justifier que le corps de la boucle *while* est répété au plus $p+1$ fois. En déduire un ordre de grandeur du temps d'exécution de **`puiss_rapide(A,N)`** en fonction de p puis en fonction de N.

```
algorithme puiss_rapide(A,N)
  A : nombre
  N : nombre entier positif
  Résultat : la valeur de A^N
début algorithme
  x ← A
  n ← N
  y ← 1
  # Point 1 (ceci est un commentaire)
  tant que n > 0 faire
    # Point 2
    si n est impair alors
      y ← yx
    fin si
    x ← x^2
    n ← ⌊n/2⌋ (Partie entière)
    # Point 3
  fin tant que
  # Point 4
  renvoyer y
fin algorithme
```

Exercice 3 *Évaluation d'un polynôme, méthode de* HORNER. On considère un polynôme A noté :

$$A = \sum_{k=0}^{n} a_k X^k$$

et que l'on représente en PYTHON par la liste **`A`** $= [a_0, \ldots, a_n]$ (on supposera que cette liste est non vide).

(a) Écrire une fonction **`evaluer(A,x)`** qui calcule la valeur en x du polynôme représenté par la liste A. On utilisera une boucle *while* dont l'invariant de boucle pourra s'écrire $y = a_0 + a_1 x + \cdots + a_i x^i$.

(b) Reprendre la question précédente mais en faisant en sorte que l'invariant de boucle puisse s'écrire $y = x^i a_n + x^{i-1} a_{n-1} + \cdots + x a_{n-i+1} + a_{n-i}$ (la méthode de calcul de $A(x)$ obtenue ici s'appelle la *méthode Horner*).

Corrections

Ex 1.

```
def nocc(x,L):
    """
    Calcule le nombre d'apparitions de x dans L
    """
    n=0
    i=0
    while i<len(L): # len(L)-i est >=0 et strict. déc.
        if L[i]==x:
            n=n+1
        i=i+1
    # Invariant de boucle : n=nocc(x,L[0..i])
    return n
def majoritaire(L):
    """
    Détermine _un_ élément majoritaire de L
    """
    xmaj=L[0]
    nmaj=nocc(xmaj,L)
    i=1
    while i<len(L): # len(L)-i est >=0 et strict. déc.
        if nocc(L[i],L)>nmaj:
            xmaj=L[i]
            nmaj=nocc(xmaj,L)
        i=i+1
    # Invariant de boucle : xmaj est l'élément de L[0:i]
    # qui apparait le plus souvent dans L
    return xmaj
```

Ex 2. Les valeurs successives de x, y et n pour $A = 2$ et $N = 21$:

x	~~2~~	~~4~~	~~16~~	~~256~~	~~65536~~	4294967296
y	~~1~~	~~2~~	~~32~~	2097152		
n	~~21~~	~~10~~	~~5~~	~~2~~	~~1~~	0

Ceci correspond au calcul « à la main » :

$$\begin{aligned}
2^{21} &= 2^{21} \times 1 \\
&= 2^{2\times 10+1} = (2^2)^{10} \times 2 = 4^{10} \times 2 \\
&= 4^{2\times 5} \times 2 = (4^2)^5 \times 2 = 16^5 \times 2 \\
&= 16^{2\times 2+1} \times 2 = (16^2)^2 \times 16 \times 2 = 256^2 \times 32 \\
&= 65536^1 \times 32 \\
&= (65536^2)^0 \times 65536 \times 32 = 4294967296^0 \times 2097152 \\
&= 2097152
\end{aligned}$$

```
def puiss_rapide(A,N):
    """
    Calcul rapide de A^N
    A : int ou float
    N entier >=0
    """
    x=A
    n=N
    y=1
    # Point 1
    while n>0:
        # Point 2
        if n%2==1:
            y=y*x
        x=x**2
        n=n//2
        # Point 3
    # Point 4
    print(x)
    return y
print(puiss_rapide(2,21))
```

```
4294967296 2097152
```

La quantité n est entière, positive et strictement décroissante à chaque tour de boucle. Par conséquent, la fonction *termine*. La propriété $y \times x^n = A^N$ est un invariant de boucle. En effet :

- Cette propriété est vraie au point 1 car $A^N = x^n = y \times x^n$ puisque $x = A$, $n = N$ et $y = 1$;
- Si on suppose cette propriété vraie au point 2, alors il faut distinguer deux cas :
  - — Si n est pair, notons $n = 2n'$, alors n' est la nouvelle valeur de n au point 3, $x' = x^2$ est la nouvelle valeur de x en 3 et $y' = y$. Ainsi :

$$y' \times x'^{n'} = y \times (x^2)^{n'} = y \times x^{2n'} = y \times x^n = A^N$$

  - — Si n est impair, notons $n = 2n' + 1$, alors n' est la nouvelle valeur de n au point 3, $x' = x^2$ est la nouvelle valeur de x en 3 et $y' = xy$. Ainsi :

$$y' \times x'^{n'} = xy \times (x^2)^{n'} = xy \times x^{2n'} = y \times x^{2n'+1} = y \times x^n = A^N$$

Dans tous les cas, la propriété reste vraie au point 3.

Nécessairement, cette propriété est vraie à la sortie de la boucle. Or, quand la boucle est terminée, on a $n = 0$ donc $x^n = 1$ et $A^N = y \times x^n = y$. Le résultat retourné est donc bien égal à A^N. La fonction est *correcte*. Calcul de la *complexité* (en fonction de N). Soit $p \in \mathbb{N}$ tel que $2^p \leqslant N < 2^{p+1}$, la boucle *while* est repétée $p+1$ fois (on divise à chaque fois par 2 jusqu'à obtenir 0). Or en appliquant la fonction ln :

$$p\ln(2) \leqslant \ln(N) \leqslant (p+1)\ln(2)$$

Donc la boucle est effectuée au plus $\dfrac{\ln(N)}{\ln(2)} + 1$ fois et ainsi :

$$\mathrm{T}_{\texttt{puiss\_rapide}}(N) = \mathrm{O}(\ln(N))$$

Maintenant que le fonctionnement du programme est compris, on peut l'écrire de manière plus concise : on utilise directement A et N (qui peuvent être considérées comme des variables locales de la fonction) à la place de x et n :

```
def puiss_rapide(A,N):
    """
    Calcul rapide de A^N
    A de type int ou float
    N entier >=0
    """
    y=1
    while N>0:
        if N%2==1:
            y=y*A
        A=A**2
        N=N//2
    return y
print(puiss_rapide(2,21))
```

```
2097152
```

Ex 3. (a) Vu l'invariant de boucle, on doit définir des variables i et y. On prendra initialement $y = A[0]$ et $i = 0$ de sorte que l'invariant est vrai. À chaque étape de la boucle, on ajoute à y la quantité $A[i+1]x^{i+1}$ et on augmente i de 1 de sorte que l'invariant est maintenu. On doit arrêter le calcul lorsque $i = n$, la boucle doit donc continuer tant que $i < n$. Notons qu'on a ici $n = len(A) - 1$.

```
def evaluer(A,x):
    n = len(A)-1
    y = A[0]
    i = 0
    while i<n:
        y = y+A[i+1]*x**(i+1)
        i = i+1
    return y
print(evaluer([1,-2,3,-4],3))
```

```
-86
```

(b) On a toujours des variables i et y. Initialement, $y = A[n]$ et $i = 0$ de sorte que l'invariant est vrai. Dans la boucle, on multiplie y par x et on ajoute $A[n-i-1]$ puis on augmente i de 1. Ici également, on doit arrêter le calcul lorsque $i = n$.

```
def evaluer(A,x):
    n = len(A)-1
    y = A[n]
    i = 0
    while i<n:
        y = x*y+A[n-i-1]
        i = i+1
    return y
print(evaluer([1,-2,3,-4],3))
```

Cette méthode de calcul est intéressante car elle ne nécessite pas de calculer des puissances et utilise simplement n multiplications. C'est la méthode de HORNER pour évaluer un polynôme.