TP4 – Ecriture et analyse d'un programme

A – Travaux dirigés

Exercice 1 : Syracuse

On définit la suite de Syracuse de la manière suivante. Le premier terme est un entier naturel non nul n. Si un terme est pair, le suivant vaut sa moitié, si un terme est impair, le suivant vaut le triple plus un.

Dans le programme suivant rajouter des assertions afin de vérifier que n est strictement positif et une autre qui permettra d'éviter des problèmes si on utilise un flottant dont la valeur n'est pas entière.

Exercice 2 : Pair ou impair

La fonction qui suit prend en paramètre un entier n et doit envoyer True si n est un nombre pair et False sinon. Cependant cela ne fonctionne pas comme prévu. Modifier le programme et expliquer.

```
def pair(n):
    """n est un entier et renvoie True si n est pair, False sinon"""
    if n%2:
        return True
    else:
        return False
```

Exercice 3 : Tuple

Quel est le type du résultat renvoyé par la fonction f. Quelle est la valeur de c ?

```
Entrée [20]: def f(x):
                 return 2*x,3*x,4*x

             a,b,c=f(2)
             print(c)
```

Exercice 4 : Invariant de boucle

Dans la fonction suivante, les valeurs des variables a et b sont des entiers naturels :

```
Entrée [1]: def f(a,b):
                s,k=a,b
                while k>0:
                    s=s+1
                    k=k-1
                return s
```

Quelle affirmation est fausse ?

- La propriété « $s + k = a + b$ » est un invariant de la boucle while.
- La valeur finale de k est 1.
- La propriété « $k \geq 0$ » est un invariant de la boucle while.
- Le résultat renvoyé est égal à la somme a+b.

Exercice 5 : Division euclidienne

La fonction div_euclid doit renvoyer le quotient et le reste de la division euclidienne de m par n où m et n sont deux entiers naturels avec n non nul. Le code de cette fonction comporte une erreur.

Entrée [2]:
```
def div_euclid(m,n):
    """m et n sont deux entiers naturels, n est non nul, renvoie le quotient
    et le reste de la division de m par n - une propriété invariante est
    m==n*q+r"""
    assert n!=0
    q,r=0,m
    assert m==n*q+r
    while r>n:
        r=r-n
        q=q+1
        assert m==n*q+r
    return q,r

div_euclid(25,3)
```

Out[2]: (8, 1)

1°) Prouver la terminaison de l'algorithme.

2°) Prouver que $m = n \times q + r$ est un invariant de la boucle. Peut-on en déduire que l'algorithme est correct ? Si ce n'est pas le cas, corriger le code.

3°) Ecrire un jeu de tests pour cette fonction.

Exercice 6 : Algorithme de Hörner

L'objectif est de comparer deux algorithmes permettant d'évaluer la valeur de P(x) pour une valeur de x donnée avec : $P(x) = a_n x_n + a_{n-1}x_{n-1} + \ldots + a_2 x_2 + a_1 x_1 + a_0$. La liste des coefficients est notée $a = [a_0, a_1, \ldots, a_n]$;

Le programme ci-dessous correspondant à l'algorithme 1 :

Entrée [1]:
```
def polynome(x,a):
    p=a[0]
    for i in range(1,len(a)):
        puissance=1
        for j in range (1,i+1):
            puissance=puissance*x
        p=p+a[i]*puissance
    return p
```

Entrée [2]:
```
a=[1,2,1]
polynome(2,a)
```

Out[2]: 9

1°) Donnez la complexité de l'algorithme 1.

2°) On va écrire un algorithme 2, appelé *algorithme de Hörner*, basé sur une écriture différente de P(x) :

$P(x) = a_0 + x\big(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n)\ldots)\big)$. Donc ici, la liste des coefficients est parcourue dans l'ordre inverse ; la variable p est initialisée avec a_n et nous n'utilisons qu'une seule boucle dans laquelle est effectuée une multiplication par x et l'addition du coefficient précédent. Écrire le programme et déterminer sa complexité.