Description du fonctionnement d'un programme

On peut décrire le fonctionnement d'un programme avec plusieurs niveaux de rigueur et de précision. Dans tous les cas, on précisera le rôle du programme, la nature des variables utilisées, l'évolution de leurs valeurs lors des boucles. Pour décrire rigoureusement, et justifier, le fonctionnement d'un programme, on écrit une preuve de fonctionnement.

Deux types de preuves

- Preuve de terminaison : preuve qu'une boucle while se termine
- Preuve de correction : preuve qu'un algorithme donne le résultat attendu

La combinaison des deux permet d'affirmer qu'un algorithme fonctionne.

Preuve de terminaison

On s'appuiera souvent sur les propriétés suivantes :

- Une suite d'entiers naturels strictement décroissante est finie.
- Une suite d'entiers naturels strictement croissante tend vers $+\infty$ (et sera supérieur à n'importe quel nombre A au bout d'un certain rang).

On appelle parfois « variant » de boucle une expression du type entier naturel strictement décroissante.

Remarque : On ne sait pas toujours prouver qu'une boucle s'arrête. Un des exemples les plus célèbres est la conjecture de Syracuse, problème non résolu à ce jour, qui mobilisa tant les mathématiciens durant les années 1960, en pleine guerre froide, qu'une plaisanterie courut selon laquelle il faisait partie d'un complot soviétique visant à ralentir la recherche américaine.

Preuve de correction

Là aussi, la difficulté vient des boucles. Pour le reste, il suffit de suivre l'algorithme ligne par ligne.

On s'appuie sur un invariant de boucle : une proposition qui reste vraie à chaque exécution de la boucle. On l'exprime en général en fonction du nombre n d'exécutions de la boucle et on utilise une preuve par récurrence. Plusieurs rédactions sont possibles. Le plus souvent, nous ajouterons un indice aux noms de variables affectées par la boucle : ainsi, nous noterons a_n la valeur de la variable a lors de la n-ième exécution de la boucle.

Exercices

1. Prouver que le code suivant affiche la somme des entiers de 1 à 100.

```
s = 0
for i in range(1,101):
    s+=i
print(s)
```

2. Après avoir deviné la fonction du programme suivant, prouver que la boucle s'arrête et que le programme remplit bien son rôle.

```
def surprise(a,n):
    r = 1
    while n>0:
        if (n%2):
            r = r * a
            n = n - 1
        else:
            a = a * a
            n = n / 2
    return r
```

3. *Recherche par dichotomie du zéro d'une fonction continue et monotone*

On considère le programme suivant :

```
def f(x):
    return x**3-2
```

```
a,b=0,2
while (b-a)>0.000000001:
    x=(a+b)/2
    if f(x)<0:
        a=x
    else:
        b=x
print(x)
```

On note $a_0 = 0$, $b_0 = 2$ et pour tout entier $n \geq 1$, a_n et b_n les valeurs respectives de a et b après la n-ième exécution de la boucle while.
Montrez $\mathcal{P}(n)$ pour tout entier naturel n:

« $a_n \leq a_{n+1} \leq b_{n+1} \leq b_n$, $b_{n+1} - a_{n+1} = \frac{b_n - a_n}{2}$, $f(a_n) \leq 0$ et $f(b_n) \geq 0$ »

En déduire la preuve de terminaison et de correction du programme.

4. Le programme suivant s'arrête-t-il toujours ?

```
seuil = float(input("Entrer un seuil : "))
s = 0
k = 1
while s<seuil:
    s+=1/k
    k+=1
print(k)
```

5. *Tri par sélection*

Prouver que la liste T retournée par la fonction suivante contient les termes de la liste L classés dans l'ordre croissant.

```
def tri_selection(L):
    T=[]
    for i in range(0,len(L)):
        min = L[0]
        for x in L:
            if x<min:
                min = x
        T.append(min)      # niveau d'indentation : 8 espaces
        L.remove(min)
    return T
```

6. *Tri à bulles*

```
def tribulles(lst):
    pastrie = True
    while pastrie:
        pastrie = False
        for i in range(len(lst)-1):
            if lst[i] > lst[i+1]:
                lst[i], lst[i+1] = lst[i+1], lst[i]
                pastrie = True
    return lst
```

Expliquer en français le principe de cet algorithme de tri.
Prouver sa correction et sa terminaison

7. Décrire le fonctionnement de la fonction suivante (les plus vaillants pourront tenter de rédiger une preuve).

```
def test_pal(a):
    pal = True
    l = len(a)-1
    i = 0
    while i<l and pal==True:
        if a[i]!= a[l]:
            pal = False
        i+=1
        l-=1
    return pal
```

8. *Le Redoutable Crible d'Eratosthène*

Prouver que eratosthene(N) retourne la liste P des nombre premiers inférieurs ou égaux à N.

```
def eratosthene(N) :
    P = []
    L = [True]*(N+1)
    for i in range(2,int(N**0.5)+1) :
        if L[i] :
            L[2*i::i] = [False]*(N//i-1)
    for i in range(2,N+1) :
        if L[i] :
            P.append(i)
    return P
```