5 Boucles *while*

◇ Une boucle *while* repète une liste d'intructions tant qu'une certaine condition est satisfaite. Une telle boucle est décrite par 2 éléments :

- Une *condition c* ;
- Une *liste d'intructions*.

◇ Dans un algorithme, on écrit :

```
tant que c
    Liste d'instructions
fin tant que
```

⚠ Il est facile de produire ainsi des programmes qui ne s'arrêtent pas. Lorsque l'on utilise une boucle *while*, il faut avoir un argument permettant de justifier que la condition *c* deviendra nécessairement fausse.

Exemple Python. Un emprunt de 100 000 € a été effectué. Chaque mois la somme restant à rembourser est augmentée avec un taux d'intérêt de 0.3% et on rembourse une mensualité de 500 €. Combien de temps faut-il pour rembourser l'emprunt ?

```
dette=100000
mois=0
while dette>0:
    dette=dette+0.3/100*dette-500
    mois=mois+1
print u"L'emprunt est remboursé en", mois, "mois"
```

```
L'emprunt est remboursé en 306 mois
```

□

⚠ Les intructions à répéter sont repérées par leur indentation. On notera que la dernière instruction de l'exemple ci-dessus ne fait par conséquent pas partie de la boucle (elle n'est exécutée qu'une seule fois).

Application aux approximations

Exemple Python. On considère les suites (u_n) et (v_n) définies par :

$$u_0 = 1 \text{ et } v_0 = 2$$
$$\forall n \in \mathbb{N},\ u_{n+1} = \frac{u_n + v_n}{2} \text{ et } v_{n+1} = \sqrt{u_n v_n}$$

On admet que ces suites sont adjacentes et on note ℓ leur limite commune. Écrire une fonction qui calcule une valeur approchée de ℓ à 10^{-p} près. □

```
from math import *
def limite_commune(p):
    u=1
    v=2
    while abs(v-u)>10**(-p):
        t=float(u+v)/2
        v=(u*v)**0.5
        u=t
    return u
print limite_commune(4)
```

```
1.45679104815
```

Introduction à la preuve de programme

◇ En présence de boucle *while*, il est possible de réaliser des programmes qui bouclent indéfiniment. Il est donc important d'avoir les moyens de justifier qu'un programme réalise bien ce que l'on attend de lui. On considère le programme ci-contre qui calcule A^N pour une variable A de type *float* et N de type *int* avec $N \geqslant 0$. Il y a deux choses à démontrer :

- La *terminaison*, c'est à dire le fait que la boucle ne tourne pas indéfiniment ;
- La *correction*, c'est à dire le fait que le résultat x est égal à A^N.

```
def puiss(A,N):
    x=1
    n=N
    # Point 1
    while n>0:
        # Point 2
        x=x*A
        n=n-1
        # Point 3
    # Point 4
    return x
print puiss(2,5)
```

```
32
```

◇ *Terminaison* : à chaque étape de la boucle, n décroit scritement. Par ailleurs, n reste entier et positif. Or, il n'existe pas de suites infinies d'entiers positifs qui soit strictement décroissante. Cette suite est donc *finie* et la boucle termine.

◇ *Correction* :

- Au point 1 du programme, $x \cdot A^n = A^N$ (car $x = 1$ et $n = N$) ;
- Supposons que l'on soit au point 2 du programme et que $x \cdot A^n = A^N$. Au point 3, la nouvelle valeur de x est $x' = xA$ et la nouvelle valeur de n est $n' = n - 1$ et ainsi $x' \cdot A^{n'} = xA \cdot A^{n-1} = x \cdot A^n = A^N$.

Considérons la propriété P : $x \cdot A^n = A^N$. D'après les deux points précédents :

- P est vraie au point 1 ;
- Si P est vraie au point 2, alors P est vraie au point 3.

Ainsi, la propriété P est vraie tout au long du déroulement de la boucle : on dit que P est un *invariant de la boucle*. Conséquence : lorsque l'on sort de la boucle (point 4), ce qui arrive nécessairement (cf. *terminaison*), la propriété P est toujours vraie. De plus, au point 4, on a $n = 0$ (car on vient de sortir de la boucle) donc $A^N = x \cdot A^n = x$ (car $n = 0$).