Terminaison et correction des algorithmes

Activité – Première NSI

① Un algorithme s'arrête-t-il toujours ?

Exercice 1 *Deux programmes suspects*

Voici deux programmes Python :

Programme A :

```
def programme_a(n):
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
    return n
```

Programme B :

```
def programme_b(n):
    while n > 0:
        n = n + 1
    return n
```

1. Exécuter à la main le programme A pour $n = 6$. Noter les valeurs successives de n.

   .....

   .....

2. Exécuter à la main le programme B pour $n = 3$. Que se passe-t-il ?

   .....

3. Parmi ces deux programmes, lequel semble poser problème ? Expliquer.

   .....

   .....

Remarque 1. Quand un algorithme contient une boucle `while`, rien ne garantit a priori qu'il s'arrêtera. On doit le **prouver**. C'est la question de la **terminaison**. En revanche, une boucle `for` se termine toujours car le nombre d'itérations est fixé à l'avance.

② Tracer un algorithme

Exercice 2 *La division euclidienne*

Considérons l'algorithme suivant :

```python
def division(a, b):
    q = 0
    r = a
    while r >= b:
        q = q + 1
        r = r - b
    return (q, r)
```

1. Compléter la trace de cet algorithme pour $a = 22$ et $b = 3$:

Étape	`q`	`r`	`r >= b` ?
Init.	0	22	Vrai
Tour 1			
Tour 2			
Tour 3			
Tour 4			
Tour 5			
Tour 6			
Tour 7			

2. Vérifier que le résultat renvoyé est correct : $22 = 3 \times q + r$ avec $0 \leqslant r < 3$.

.....

3. Observer la colonne `r` : comment évolue cette valeur à chaque tour ?

.....

③ Prouver la terminaison : le variant de boucle

Définition 1 **(Variant de boucle)** Un **variant de boucle** est une quantité :

— qui est un **entier positif ou nul** à l'entrée de chaque tour de boucle ;

— qui **décroît strictement** à chaque itération.

Un entier positif qui décroît strictement finit nécessairement par atteindre 0 ou devenir négatif : la boucle se termine forcément.

Exercice 3 *Variant de la division euclidienne*

1. Dans l'algorithme de division, on propose r comme variant de boucle. Justifier que r est bien un entier positif à l'entrée de chaque tour.

.....

.....

2. Justifier que r décroît strictement à chaque tour (on rappelle que $b > 0$).

.....

.....

3. En déduire que l'algorithme de division se termine toujours.

.....

Exercice 4 *Trouver le variant*

Pour chacun des algorithmes suivants, proposer un variant de boucle et justifier brièvement que la boucle se termine.

1.

```
def compte_a_rebours(n):
    while n > 0:
        n = n - 1
    return n
```

Variant proposé : .....

Justification : .....

.....

2.

```
def mystere(a, b):
    # a et b entiers, b > 0
    resultat = 0
    compteur = 0
    while compteur < b:
        resultat = resultat + a
        compteur = compteur + 1
    return resultat
```

Variant proposé : .....

Justification : .....

.....

④ Prouver la correction : l'invariant de boucle

Définition 2 **(Invariant de boucle)** Un **invariant de boucle** est une propriété qui :

— est vérifiée **avant** la première itération (**initialisation**) ;

— **reste vraie** après chaque itération (**conservation**).

Un invariant bien choisi, combiné à la condition d'arrêt de la boucle, permet de montrer que l'algorithme produit le **résultat attendu**.

Exercice 5 *Invariant de la division euclidienne*

On propose l'invariant suivant : « à chaque début de tour, $a = b \times q + r$ avec $r \geqslant 0$ ».

1. **Initialisation.** Avant la boucle, on a $q = 0$ et $r = a$. Vérifier que l'invariant est vrai.

.....

.....

2. **Conservation.** Supposons qu'au début d'un tour, $a = b \times q + r$. Après les affectations $q \leftarrow q + 1$ et $r \leftarrow r - b$, calculer $b \times (q + 1) + (r - b)$ et vérifier que l'on retrouve a.

.....

.....

.....

3. **Conclusion.** À la sortie de la boucle, la condition `r >= b` est fausse, donc $r < b$. Combiner cela avec l'invariant pour conclure que le résultat est correct.

.....

.....

⑤ Application : preuves sur des algorithmes connus

Exercice 6 *L'exponentiation*

```
def puissance(a, n):
    resultat = 1
    compteur = 0
    while compteur < n:
        resultat = resultat * a
        compteur = compteur + 1
    return resultat
```

1. Écrire la trace pour $a = 2$ et $n = 4$:

Étape	`resultat`	`compteur`
Init.	1	0
Tour 1		
Tour 2		
Tour 3		
Tour 4		

2. **Terminaison.** Proposer un variant de boucle et justifier.

.....

.....

3. **Correction.** On propose l'invariant : « `resultat` $= a^{\texttt{compteur}}$ ».

   a) Vérifier l'initialisation.

   .....

   b) Vérifier la conservation : si `resultat` $= a^{\texttt{compteur}}$ au début d'un tour, que vaut `resultat` $\times a$?

   .....

   .....

   c) Conclure en utilisant la condition d'arrêt.

   .....

Exercice 7 *La recherche séquentielle*

```python
def recherche(liste, valeur):
    for i in range(len(liste)):
        if liste[i] == valeur:
            return True
    return False
```

1. La terminaison est-elle garantie ? Justifier.

   .....

   .....

2. On propose l'invariant suivant : « après le tour i, si la fonction n'a pas encore renvoyé `True`, alors `valeur` n'apparaît pas dans `liste[0]`, ..., `liste[i]` ». Vérifier l'initialisation et la conservation.

   .....

   .....

   .....

3. En déduire que si la fonction renvoie `False`, alors `valeur` n'est pas dans la liste.

   .....

6 Synthèse

Exercice 8 *Bilan*

Compléter le tableau récapitulatif :

Question	**Outil**	**Méthode**
L'algorithme s'arrête-t-il ?		
L'algorithme donne-t-il le bon résultat ?		