Prérequis : utilisation des boucles bornées (POUR - for) et des boucles non bornées (TANT QUE - while).

1 Mise en évidence du problème

1. Le programme suivant s'arrête-t-il jamais ? parfois ? toujours ?
   Algo 1 :

   N ← 4
   pour *i* **de** *1* **à** *N*
   faire
   | **afficher** (i)
   fin pour

```
N = 4
for i in range(1, N+1):
  print(i)
```

2. Le programme suivant s'arrête-t-il jamais ? parfois ? toujours ?
   Algo 2 :

   N ← 10
   mini ← 0
   tant que $N > mini$
   faire
   | **afficher** (N)
   | N ← N + 1
   fin tq

```
N = 10
mini = 0
while N > mini:
  print(N)
  N = N + 1
```

3. Le programme suivant s'arrête-t-il jamais ? parfois ? toujours ?
   Algo 3 :

   N ← 10
   maxi ← 15
   tant que $N < maxi$
   faire
   | **afficher** (N)
   | N ← N + 1
   fin tq

```
N = 10
maxi = 15
while N < maxi:
  print(N)
  N = N + 1
```

4. Le programme suivant s'arrête-t-il jamais ? parfois ? toujours ?
   Algo 4 :

   reponse ← 'non'
   tant que *reponse != 'oui'* **faire**
   | reponse ← **saisir** ('oui ou non ?')
   fin tq

```
reponse = 'non'
while reponse != 'oui':
  reponse = input('oui  ou non ?')
```

2 Variant de boucle

À l'aide des exemples précédents, on met en évidence un problème qui peut survenir lors de la construction d'algorithme avec des boucles : il est possible que la boucle ne s'arrête pas et donc que le **programme** qui le met en œuvre **ne termine pas**.

Les exemples sont assez simples mais certains algorithmes peuvent être plus complexes. Néanmoins, le principe reste toujours le même pour *prouver la terminaison* d'un algorithme : il faut pouvoir exhiber un **variant de boucle** : c'est une **grandeur positive qui décroît à chaque itération de la boucle**.

En effet, si on peut trouver une telle grandeur, alors l'algorithme va forcément terminer lorsque le variant va cesser d'être positif.

3 Retour sur les algorithme d'introduction

Pour chaque algorithme, chercher un variant de boucle qui prouve sa terminaison (si possible !).

1. Algo 1 : le variant de boucle est $N - i$.

itération n°	i	N - i
1	1	3
2	2	2
3	3	1
4	4	0

   Lorsque le variant s'annule, la boucle s'arrête.
2. Algo 2 : il n'existe pas de variant de boucle : en l'occurrence l'algorithme ne termine jamais. La grandeur $N - mini$ croît à chaque itération et reste toujours positive...
3. Algo 3 : le variant de boucle est $maxi - N$.

itération n°	(début) N	(début) $maxi - N$	(fin) N	(fin) $maxi - N$
1	10	5	11	4
2	11	4	12	3
3	12	3	13	2
4	13	2	14	1
5	14	1	15	0

4. Algo 4 : il n'existe pas de variant de boucle. Cependant cet algorithme **peut** terminer, mais ce n'est pas une garantie ! C'est à priori un mauvais algorithme.
   On peut proposer une version améliorée de cet algorithme pour qu'il termine toujours.
   Algo 4 :

   reponse ← 'non'
   tentative ← 1
   maxi ← 10
   tant que *(reponse != 'oui'* **ET** *tentative < maxi)* **faire**
   | reponse ← **saisir** ('oui ou non ?')
   | tentative ← tentative + 1
   fin tq

```
reponse = 'non'
tentative, maxi = 1, 10
while (reponse != 'oui' and tentative < maxi):
  reponse = input('oui  ou non ?')
  tentative = tentative + 1
```

Le variant de boucle est $maxi - tentative$.