

Asie – 2025 – sujet1 - Correction

Exercice 1 (6 points)

1) Valeurs de x et y après exécution de programme1

Programme :

```
x = 10
y = 10
while x > 0:
    x = x - 1
    y = y + 1
```

Analyse :

- x diminue de 1 à chaque tour.
- y augmente de 1 à chaque tour.
- La boucle s'exécute tant que $x > 0$.
- Il y a donc 10 itérations.

Valeurs finales :

- **x = 0**
 - **y = 20**
-

2) Pourquoi tout programme Python peut être vu comme une chaîne de caractères ?

Un programme Python est un texte écrit dans un fichier.
Or un texte est une suite de caractères.

Ainsi, tout programme peut être représenté comme une chaîne de caractères, puis interprété ou exécuté par Python (via exec).

3) Terminaison des programmes 3, 4, 5 et 6

- programme3 : x prend les valeurs 10, 8, 6, 4, 2, 0 ; la condition $x \neq 0$ devient fausse : **programme3 termine.**
- programme4 : x vaut 10, 12, 14... et reste toujours strictement positif : **programme4 ne termine pas.**
- programme5 : $x = 10$ n'est pas strictement négatif, la boucle n'est jamais exécutée : **programme5 termine.**
- programme6 : x prend les valeurs 10, 6, 2, -2, -6... et ne vaut jamais 0 : **programme6 ne termine pas.**

👉 *Commentaire : programme3 et programme6 ont la même condition $x \neq 0$: tout dépend du pas (2 divise 10, 4 ne divise pas 10).*

4) Analyse de arret_essai1

```
def arret_essai1(programme):  
    exec(programme)  
    return True
```

Ce programme :

- exécute le programme donné
- renvoie True si exec se termine

Problème :

Si le programme ne termine pas, alors arret_essai1 ne termine pas non plus.

Donc cette fonction **ne résout pas le problème** demandé car elle ne s'arrête pas dans tous les cas.

5) Principe de l'algorithme de Boyer-Moore

L'algorithme de Boyer-Moore est un algorithme efficace de recherche de motif dans une chaîne.

Principe :

- Il compare le motif au texte de droite à gauche.
- En cas de désaccord, il saute plusieurs caractères grâce à des tables de décalage.

- Cela permet souvent d'éviter de comparer tous les caractères.

Il est plus efficace qu'une recherche naïve.

6) Fonction `arret_essai2`

```
def arret_essai2(programme):  
    return not recherche('while', programme)
```

7) Montrer les limites de `arret_essai2`

Cas 1 : renvoie `True` mais le programme ne termine pas

Exemple :

```
l = [1]  
for x in l:  
    l.append(x)
```

Il n'y a pas de "while" mais la boucle `for` ne termine pas : à chaque tour on ajoute un élément à la liste parcourue, qui n'est donc jamais épuisée.

Donc `arret_essai2` renvoie **True** alors que le programme **ne termine pas**.

👉 *Commentaire : une fonction récursive sans cas de base convient aussi (une fonction f qui s'appelle elle-même), à condition d'appeler f dans le programme : la seule définition de f termine immédiatement. En pratique, Python l'interrompt par une `RecursionError` (environ 1000 appels).*

Cas 2 : renvoie `False` mais le programme termine

Exemple :

```
x=1  
while x!=1:  
    print("NSI for ever!")
```

Le mot "while" est présent, donc `arret_essai2` renvoie **False**.

Mais la boucle ne s'exécute jamais, donc le programme termine. C'est aussi le cas du **programme5**.

8) Fonction `terminaison_inverse`

On suppose que la fonction `arret` existe.

```
def terminaison_inverse(programme):  
    if arret(programme):          # programme termine  
        boucle_infinie()         # alors on ne termine pas  
    # sinon on termine aussitôt
```

Comportement :

- Si programme termine → `terminaison_inverse` boucle infiniment.
- Si programme ne termine pas → `terminaison_inverse` termine.

9) Étude du programme paradoxal

On définit :

```
programme_paradoxal = "terminaison_inverse(programme_paradoxal)"
```

On analyse :

- Si `programme_paradoxal` termine,
alors `arret` renvoie `True`,
donc `terminaison_inverse` boucle infiniment,
donc `programme_paradoxal` ne termine pas.

Contradiction.

- Si `programme_paradoxal` ne termine pas,
alors `arret` renvoie `False`,
donc `terminaison_inverse` termine,
donc `programme_paradoxal` termine.

Nouvelle contradiction.

Conclusion : **On obtient un paradoxe !**

10) Conclusion sur la fonction arrêt

La fonction arrêt ne peut pas exister.

Il est impossible d'écrire un programme qui détermine pour tout programme s'il termine ou non.

C'est le problème de l'arrêt (Halting Problem), démontré indécidable par Alan Turing.

11) Cette impossibilité est-elle due à Python ?

Non.

Ce résultat est indépendant du langage.

Tout langage de programmation suffisamment expressif (Turing-complet) possède cette limitation.

Ce n'est pas une limite technique de Python, mais une limite fondamentale de l'informatique théorique.
