Compte Rendu de TP

TP 9 : Récursivité

Module : Programmation Python

Pr. : Ouafae EL BOUHADI

Étudiant : Boulahrouf Hamza

Cycle d'Ingénieur en Géoinformation

Année universitaire 2025/2026

Introduction

Ce compte rendu présente les travaux réalisés dans le cadre du TP 9 de **Programmation Python**, consacré à la **récursivité**. L'objectif de ce TP est de comprendre et d'appliquer les principes fondamentaux de la récursivité à travers plusieurs exercices, portant sur :

— Le calcul de la suite de Fibonacci ;
— Le comptage d'appels récursifs ;
— La somme des n premiers termes de Fibonacci ;
— Le comptage des voyelles dans une chaîne ;
— L'inversion d'une chaîne de caractères ;
— Le compte à rebours et la recherche dans une liste.

Chaque exercice met en évidence la puissance de la récursivité dans la résolution de problèmes de programmation.

Exercice 1 : Suite de Fibonacci

1. Suite de Fibonacci

Objectif

Écrire une fonction récursive qui renvoie le n^{me} nombre de Fibonacci.

Code principal

```python
def Fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return Fibonacci(n-1) + Fibonacci(n-2)

#test
print(f"F({8}) =", Fibonacci(8))
```

Résultat obtenu

```
F(8) = 21
```

Analyse

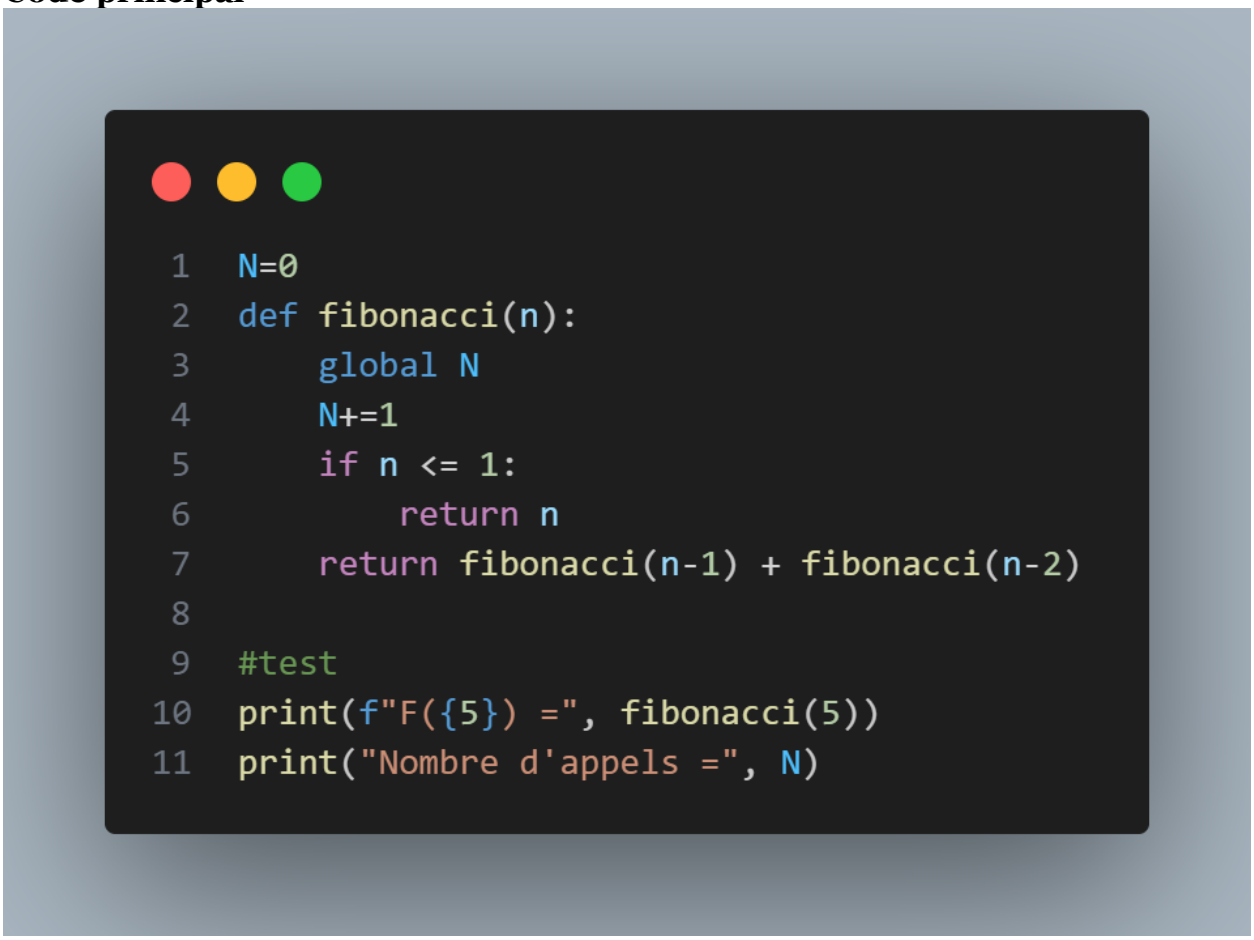
La fonction illustre le mécanisme récursif en appelant deux fois la fonction elle-même pour chaque valeur de $n > 1$. C'est une implémentation simple mais coûteuse en calculs.

2.Compteur d'appels récursifs

Objectif

Ajouter un compteur global pour mesurer le nombre total dappels récursifs effectués.

Code principal



```python
N=0
def fibonacci(n):
    global N
    N+=1
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

#test
print(f"F({5}) =", fibonacci(5))
print("Nombre d'appels =", N)
```

Résultat obtenu

```
F(5) = 5
Nombre d'appels = 15
```

Analyse

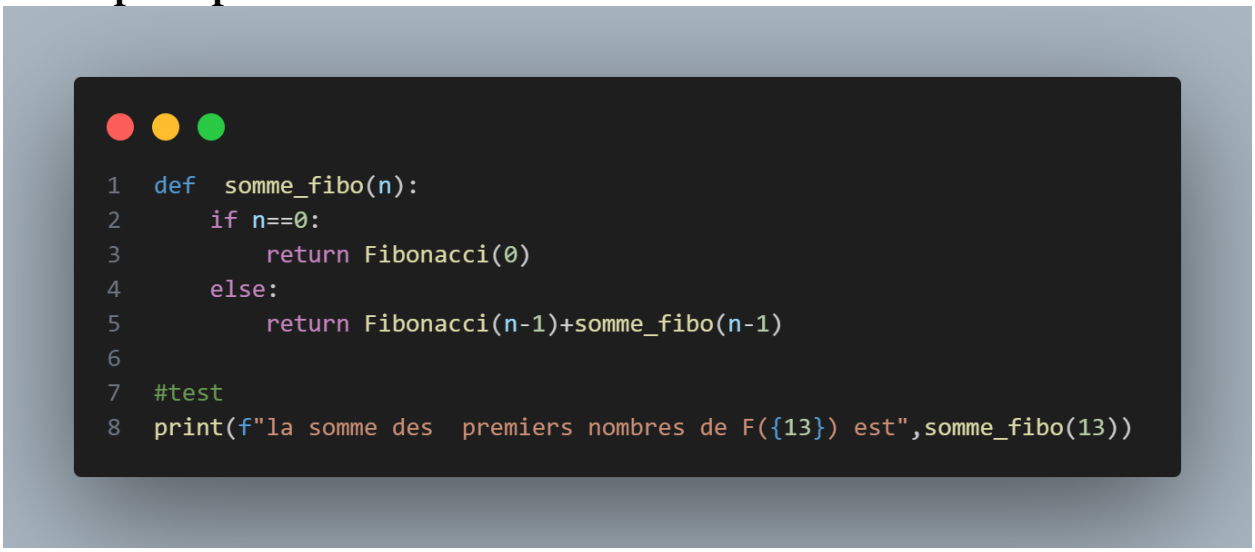
La croissance du nombre dappels récursifs est exponentielle. Cette approche permet de visualiser la lourdeur dune récursion non optimisée.

3.Somme des n premiers nombres de Fibonacci

Objectif

Écrire une fonction récursive qui retourne la somme des n premiers termes de Fibonacci.

Code principal



```python
def  somme_fibo(n):
    if n==0:
        return Fibonacci(0)
    else:
        return Fibonacci(n-1)+somme_fibo(n-1)

#test
print(f"la somme des  premiers nombres de F({13}) est",somme_fibo(13))
```

Résultat obtenu

```
376
```

Analyse

Chaque appel calcule un terme de Fibonacci et lajoute au résultat précédent. La fonction démontre la composition de fonctions récursives entre elles.

Exercice 2 : Compter les voyelles dans une chaîne

Objectif

Écrire une fonction récursive qui compte le nombre de voyelles dans une chaîne donnée.

Code principal

```python
def compter_voyelles(chaine):
    voyelles = "aeiouyAEIOUY"
    if chaine == "":
        return 0
    else:
        if chaine[0] in voyelles :
            pr = 1
        else :
            pr = 0
        return pr + compter_voyelles(chaine[1:])

#test
print("Le nombre de voyelles :", compter_voyelles("Bonjour"))
```

Résultat obtenu

```
Le nombre de voyelles : 3
```

Analyse

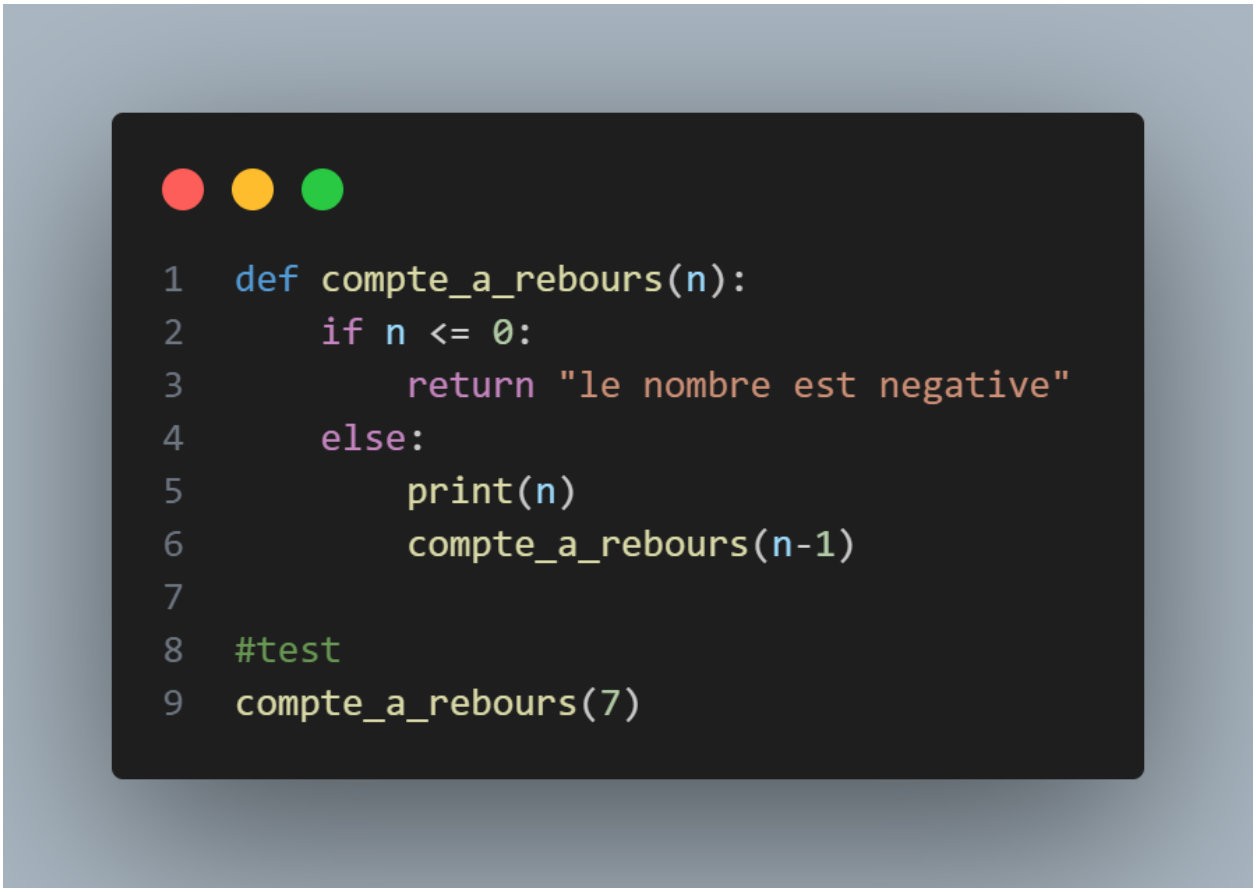
Le traitement s'effectue caractère par caractère jusqu'à atteindre une chaîne vide. C'est un exemple simple et efficace d'appel récursif sur une structure textuelle.

Exercice 3 : Inverser une chaîne

Objectif

Écrire une fonction récursive qui renvoie la chaîne inversée.

Code principal



```python
def compte_a_rebours(n):
    if n <= 0:
        return "le nombre est negative"
    else:
        print(n)
        compte_a_rebours(n-1)

#test
compte_a_rebours(7)
```

Résultat obtenu

```
radaR
```

Analyse

Chaque appel retire le premier caractère et le replace à la fin au retour de la récursion. C'est un exemple typique du mécanisme de pile dans l'exécution récursive.

Exercice 4 : Compte à rebours

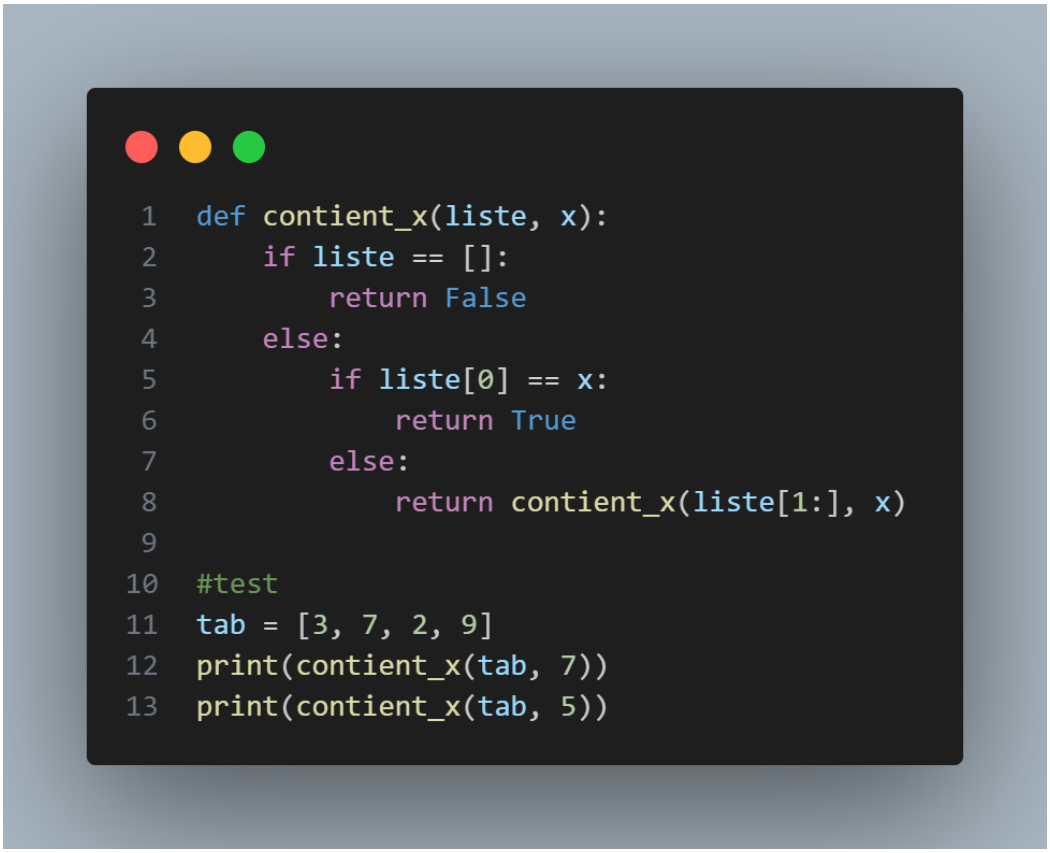
```python
def inverse_chaine(s):
    if s == "":
        return ""
    else:
        return inverse_chaine(s[1:]) + s[0]

#test
print(inverse_chaine("Radar"))
```

Résultat obtenu

```
5
4
3
2
1
```

Exercice 5 : Vérifier la présence d'un élément dans une liste



```python
def contient_x(liste, x):
    if liste == []:
        return False
    else:
        if liste[0] == x:
            return True
        else:
            return contient_x(liste[1:], x)

#test
tab = [3, 7, 2, 9]
print(contient_x(tab, 7))
print(contient_x(tab, 5))
```

Résultat obtenu

```
True False
```

Conclusion

Ce TP a permis d'explorer les bases et la puissance de la récursivité en Python. Les différents exercices ont mis en évidence son application dans des situations variées : cal- culs numériques, traitement de chaînes et de listes. La récursivité constitue un concept fondamental pour la programmation algorithmique et la compréhension de nombreux algorithmes récursifs comme les parcours d'arbres ou la recherche dichotomique.