



Programmation récursive



1. Introduction

Une fonction est dite récursive si elle comporte, dans son corps, au moins un appel à elle-même.

Exemple :

Code python	Résultat affiché dans la console
<pre>def affichage(n) : if n==0: print(n, "Terminé"); else: print(n, "hello") affichage(n-1) affichage(3)</pre>	<pre>3 hello 2 hello 1 hello 0 Terminé</pre>

Q1. La fonction affichage est-elle récursive ?

La fonction affichage est récursive car elle s'appelle elle-même.

Q2. Dans l'exemple ci-dessus, combien de fois la fonction affichage s'appelle-t-elle elle-même ?

La fonction affichage s'appelle 3 fois.

Q3. Critiquer la fonction « ouVaton » ci-dessous.

Code python	Résultat affiché dans la console
<pre>def ouVaton(n) : print(n) ouVaton(n-1) ouVaton(10)</pre>	<pre>10 9 ... -966 RuntimeError: maximum recursion depth exceeded</pre>

La fonction s'appelle elle-même indéfiniment sans s'arrêter. Cela provoque une erreur d'exécution au bout d'un certain nombre d'appels récursifs.

Voir Annexe : Pile d'exécution

Une fonction récursive doit **obligatoirement s'arrêter**. La condition terminale (if) se situe toujours au **début** de toute fonction récursive.

La condition terminale (appelée aussi point d'arrêt) ne sert à rien si elle ne devient jamais vraie.

Une fonction récursive doit comporter :

<pre>def affichage(n): if n==0: print(n, "Terminé"); else: print(n, "hello") affichage(n-1) affichage(3)</pre>	<ul style="list-style-type: none"> • Un cas d'arrêt dans lequel aucun autre appel n'est effectué • Un cas général dans lequel un ou plusieurs autres appels sont effectués
---	--

Il faut faire attention aux préconditions lors d'un appel de fonction récursive.

Q4. Que se passe-t-il si l'on appelle la fonction précédente avec « `affichage(-5)` » ?

La condition terminale ne sert à rien car n ne sera jamais égal à zéro puisque l'on commence l'appel de la fonction avec un nombre négatif. Lors de chaque appel récursif, les valeurs de n vont décroître.

Q5. Modifier la condition terminale quelle que soit la valeur de n en entrée de la fonction.

Code python	Résultat console pour <code>affichage(5)</code>	Résultat console pour <code>affichage(-3)</code>
<pre>def affichage(n): if n : print(n, "Terminé"); else: print(n, "hello") affichage(n-1)</pre>	5 hello 4 hello 3 hello 2 hello 1 hello 0 Terminé	-3 Terminé

2. Boucle « Pour » traduite en programmation récursive

2.1. Incrémentation

Programmation itérative	Programmation récursive	Résultat affiché dans la console
<pre>def afficheINC(max): for n in range(0,max+1): print(n) afficheINC(5)</pre>	<pre>def afficheINC_REC(n): if n==0: print(n) else: afficheINC_REC(n-1) print(n) afficheINC_REC(5)</pre>	0 1 2 3 4 5

2.2. Décrémentation

Programmation itérative	Programmation récursive	Résultat affiché dans la console
<pre>def afficheDEC(max): for n in range(max,-1,-1): print(n) afficheDEC(5)</pre>	<pre>def afficheDEC_REC(n): if n==0: print(n) else: print(n) afficheDEC_REC(n-1) afficheDEC_REC(5)</pre>	5 4 3 2 1 0

La plupart des traitements itératifs simples sont traduisibles sous forme récursive. L'inverse est plus difficile. Il arrive souvent qu'un problème ait une solution récursive simple alors qu'il est très difficile d'en trouver une solution itérative.

Q6. Quelle est la différence entre les deux programmes récursifs suivants ?

Programmation récursive (incrémentation)	Programmation récursive (décrémentation)
<pre>def afficheINC_REC(n): if n==0: print(n) else: afficheINC_REC(n-1) print(n) afficheINC_REC(5)</pre>	<pre>def afficheDEC_REC(n): if n==0: print(n) else: print(n) afficheDEC_REC(n-1) afficheDEC_REC(5)</pre>

Dans le cas de la décrémentation, la position du print est positionnée avant l'appel récursif. L'affichage se fait au fur et à mesure des appels récursifs.

Dans le cas de l'incrémentation, la position du print est positionnée après l'appel récursif. L'affichage se fait en remontant.

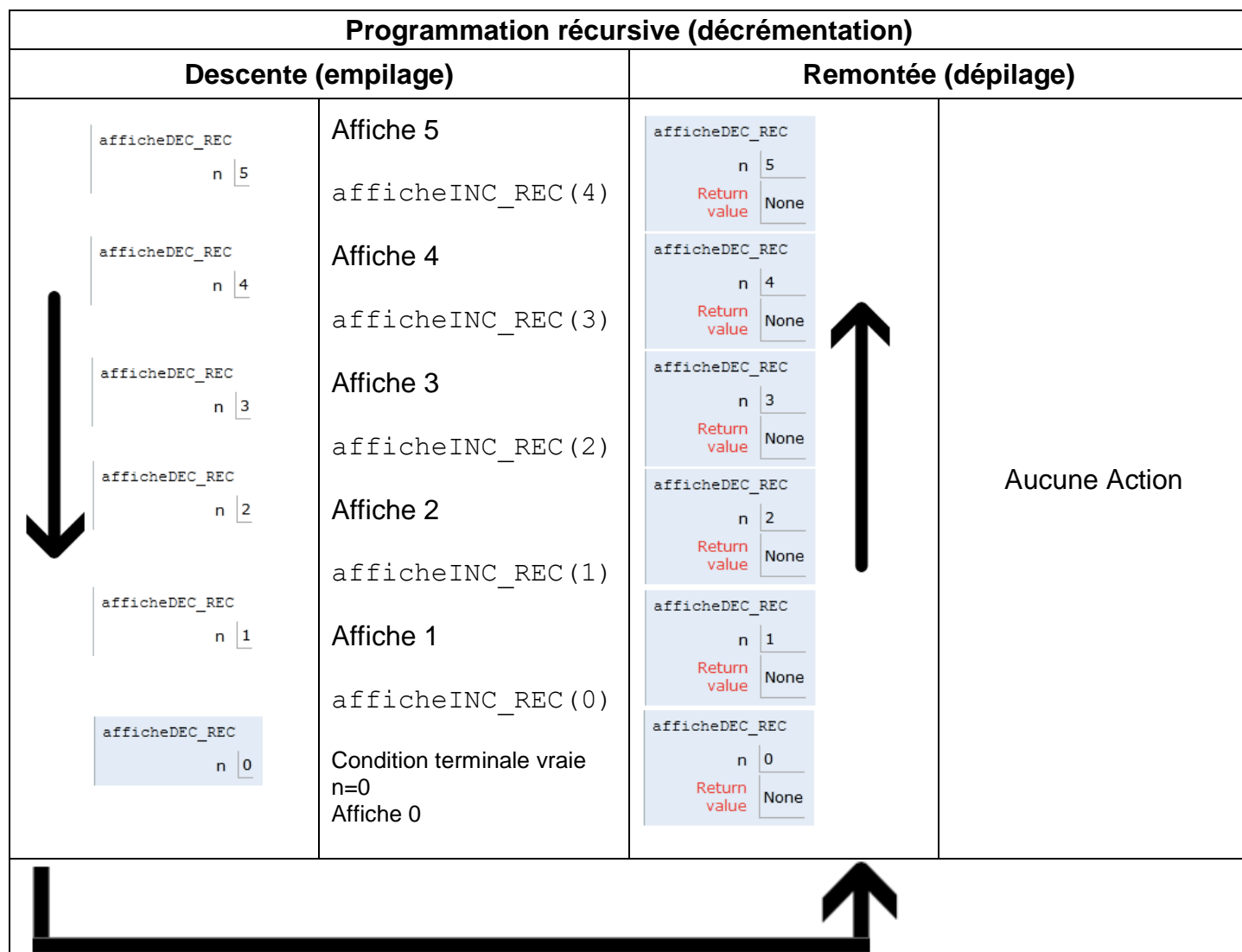
3. Exécution pas à pas d'un programme récursif

Q7. Visualiser l'exécution des 2 programmes précédents dans [python tutor](#). Bien voir la différence lors de l'affichage des valeurs de n (en descendant ou en remontant).

Programmation récursive (décrémentation)

```
def afficheDEC_REC(n) :
    if n==0:
        print(n)
    else:
        print(n)
        afficheDEC_REC(n-1)

afficheDEC_REC(5)
```



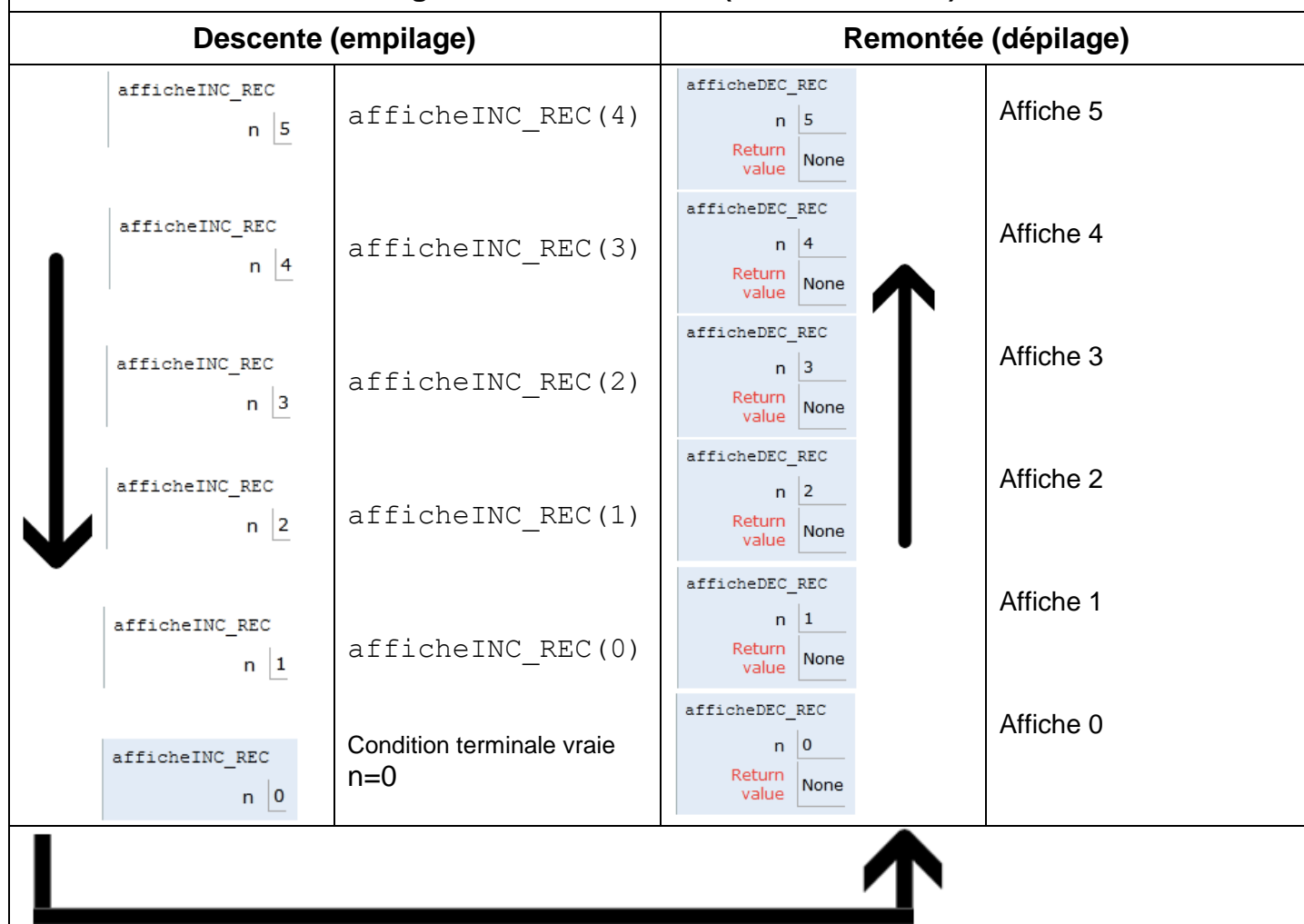
La récursivité est **terminale**, l’affichage ou les calculs sont effectués au fur et à mesure que l’on **descend (empilage)**.

Programmation récursive (incrémentation)

```
def afficheINC_REC(n):
    if n==0:
        print(n)
    else:
        afficheINC_REC(n-1)
        print(n)

afficheINC_REC(5)
```

Programmation récursive (décrémentation)



La récursivité est **non terminale**, l'affichage ou les calculs sont effectués en **remontant (dépilage)**.

4. Lien avec les mathématiques (suite arithmétique)

Soit la suite arithmétique définie par :

$$\begin{cases} u_0 = 3 \\ u_{n+1} = u_n + 2 \end{cases}$$

Q8. Compléter le tableau suivant :




u_0	u_1	u_2	u_3	u_4
3	5	7	9	11

On peut réécrire la suite par :

$$\begin{cases} u_0 = 3 : \text{Condition terminale} \\ u_n = u_{n-1} + 2 : \text{Cas général} \end{cases}$$

Code python	Résultat affiché dans la console
<pre>def u(n): if n == 0: return 3 else: return u(n-1) + 2 print(u(4))</pre>	11

Q9. Visualiser l'exécution du programme dans [python tutor](#).

Programmation récursive (décrémentation)				
Descente (empilage)		Remontée (dépileage)		
	<div>u</div> <div>n 4</div>	u(3)	<div>u</div> <div>n 4</div> <div>Return value 11</div>	Retourner 9+2
	<div>u</div> <div>n 3</div>	u(2)	<div>u</div> <div>n 3</div> <div>Return value 9</div>	Retourner 7+2
	<div>u</div> <div>n 2</div>	u(1)	<div>u</div> <div>n 2</div> <div>Return value 7</div>	Retourner 5+2
	<div>u</div> <div>n 1</div>	u(0)	<div>u</div> <div>n 1</div> <div>Return value 5</div>	Retourner 3+2
	<div>u</div> <div>n 0</div>	Condition terminale vraie n=0	<div>u</div> <div>n 0</div> <div>Return value 3</div>	Retourner 3
				

Code python	Autre représentation graphique
<pre>def u(n): if n == 0: return 3 else: return u(n-1) + 2 print(u(4))</pre>	

Q10. La récursivité est-elle **terminale** ou **non terminale** ?

La récursivité est non terminale, les calculs sont effectués en remontant (dépileage).

5. Exercice

Soit la suite géométrique définie par :

$$\begin{cases} u_0 = 5 \\ u_{n+1} = 2u_n \end{cases}$$

Q11. Compléter le tableau suivant :

u_0	u_1	u_2	u_3	u_4
5	10	20	40	80

Q12. Donner le code python en utilisant la programmation récursive pour $u(4)$.

Code python	Résultat affiché dans la console
<pre>def u(n): if n == 0: return 5 else: return 2*u(n-1) print(u(4))</pre>	<p>80</p>

Q13. Visualiser l'exécution du programme dans [python tutor](https://python-tutor.com/)

6. Conclusion

Les algorithmes récursifs ne se limitent évidemment pas au calcul de suites récurrentes et de fonctions sur les entiers naturels. Ils permettent de travailler sur des structures de données définies récursivement comme les chaînes de caractères, les listes ou les arbres.

Exemples de programmes utilisant la programmation récursive :

- Calcul du factoriel d'un nombre ;
- Calcul d'une somme dans une liste ;
- Calcul du PGCD ;
- Conversion d'un nombre décimal en binaire ;
- Inverser les caractères dans une liste
- Recherche d'un nombre dans un arbre binaire

Nous verrons ces programmes tout au long de l'année.

Annexe

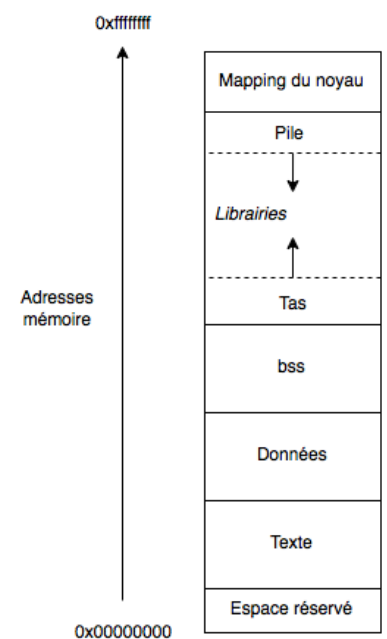
La Pile d'exécution (call stack) du programme en cours est un emplacement mémoire destiné à mémoriser les paramètres, les variables locales, ainsi qu'à garder la trace de l'endroit où chaque fonction active doit retourner à la fin de son exécution.

Elle fonctionne selon le principe LIFO (Last-In-First-Out) : dernier entré premier sorti.

Plus on empile des valeurs dans la « stack », plus les adresses diminuent.

Attention ! La pile a une taille fixée, une mauvaise utilisation de la récursivité peut entraîner un débordement de pile et un message d'erreur s'affiche.

RuntimeError: maximum recursion depth exceeded



En Python il est possible de redéfinir la taille de la pile avec la méthode `sys.setrecursionlimit` :

```
import sys

taille = sys.getrecursionlimit()
print('Taille de la pile =', taille)

NouvelleTaille = 500
sys.setrecursionlimit(NouvelleTaille)
```

Taille de la pile = 1000
taille de la pile par défaut