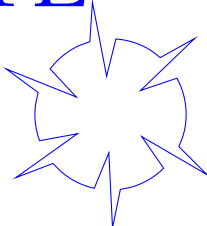


CHAPITRE

# 3 RÉCURSIVITÉ



## Table des matières

---

1	Définition .....	1
2	Étude d'un exemple. ....	2
3	Écrire une fonction récursive. ....	3

## 1 Définition

---



En informatique, on dit qu'une fonction est récursive lorsque celle fonction s'appelle elle-même.

Une fonction récursive permet donc comme une boucle de répéter des instructions. Souvent une fonction peut d'ailleurs se programmer de façon **itérative** (avec des boucles) ou de façon **récursive** (en s'appelant elle-même).

### Exemple n°1

Calcul de  $a^n$  où  $a$  est un réel et  $n$  un entier

```
1  !-- Version Iterative --#
2  def puissance_iter(a, n):
3      p = 1
4      for k in range(n):
5          p = p*a
6      return p
```

```
1  #----- Version recursive ----- #
2  def puissance_recur(a, n):
3      if n == 0:
4          return 1
5      else:
6          return a*puissance_recur(a, n - 1)
```

L'explication repose sur la définition mathématique de la puissance :

$$a^n = a \times a^{n-1}$$

qui en python se traduit finalement par :

```
puissance(a,n) = a*puissance(a, n - 1)
```

Mais dans tout processus récurrent, il faut une initialisation(voir cours de maths...).



Toute fonction récursive doit avoir un **cas d'arrêt**

Sinon, l'appel récursif est infini...

### Exercice n°1

Quel est le cas d'arrêt de la fonction récursive précédente ?

### Exercice n°2

Combien d'appel récursif a-t-on dans le calcul de `puissance_recur(3, 4)` précédent ?

## 2 Étude d'un exemple.

---

On considère la fonction suivante :

```
1 def mystere(elt, liste):
2     if liste == []:
3         return 0
4     first = liste.pop(0)
5     if elt == first:
6         return 1 + mystere(elt, liste)
7     else:
8         return mystere(elt, liste)
```

### Exercice n°3

- ❶ Pourquoi cette fonction est réursive ?
- ❷ Quel est son cas d'arrêt ?
- ❸ Pourquoi est-on certain que les appels réursifs se terminent ?
- ❹ Quel est son rôle ?

## 3 Écrire une fonction réursive.

Disons-le !



Ce n'est pas toujours facile d'écrire une fonction réursive !

En fin d'année, vous serez plus à l'aise avec ce concept...  
Voici une situation où la notion de réursivité prend tout son sens.



Un palindrome est un mot ou une phrase dont l'ordre des lettres reste le même qu'on les lise de gauche à droite ou de droite à gauche.

Les mots ou phrases suivantes :

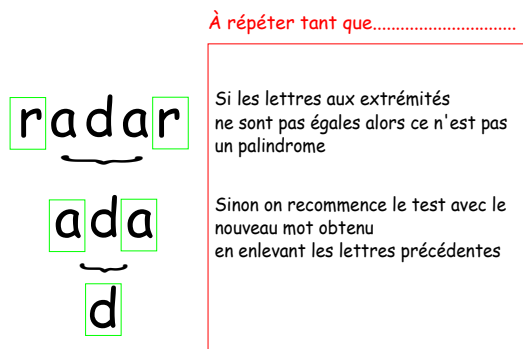
radar, kayak, ou « Ésope reste ici et se repose » ,

sont des palindromes.



Comment programmer une fonction Python qui permet de savoir si un mot est un palindrome ?

Il suffit d'appliquer un processus qui compare les lettres aux extrémités :



L'appel est donc réursif !



Quel est le cas d'arrêt ?

Si le mot testé contient 0 ou 1 lettres c'est alors un palindrome : on retourne alors dans ce cas un booléen affecté à la valeur **True**.

Finalement, la structure algorithmique de la fonction récursive python se présente ainsi :

```
1 def palindrome(mot):
2     if .....:#cas d'arret
3         return True
4     if .....:#test d'égalité
5         return False
6     else:
7         return palindrome(.....)
```

On rappelle ici un résultat important :



Un **return** arrête l'exécution d'une fonction ou d'une boucle!

Donc si la fonction retourne **False** parce qu'elle a trouvé des extrémités différentes, son exécution s'arrête!

#### Exercice n°4

Compléter les parties manquantes de la fonction `palindrome` puis testez-là!

#### Exercice n°5

Écrire la version itérative de la fonction.

On posera les assertions suivantes :

```
1 assert palindrome('radar') == True
2 assert palindrome('ici') == True
3 assert palindrome('raidir') == False
```