Math93.com

# Cours - NSI
# La récursivité

## 1 Introduction et définitions

### 1.1 Définition

**Définition 1** (Fonction récursive)

Une **fonction récursive** est une fonction qui fait appel à elle-même.

### 1.2 Un premier exemple : la fonction puissance

On peut par exemple implémenter de façon récursive la fonction puissance. On rappelle que pour $a$ réel non nul et $n$ entier naturel non nul on a :

$$\begin{cases} a^0 = 1 \\ a^n = a \times a^{n-1} \end{cases}$$

Notons alors $deux_puissance(n)$ la fonction d'argument l'entier $n$ et qui renvoie $2^n$.
On peut programmer cette fonction de façon récursive :

```
def deux_puissance(n):
    if n==0:
        return 1
    else:
        return 2*deux_puissance(n-1)
```

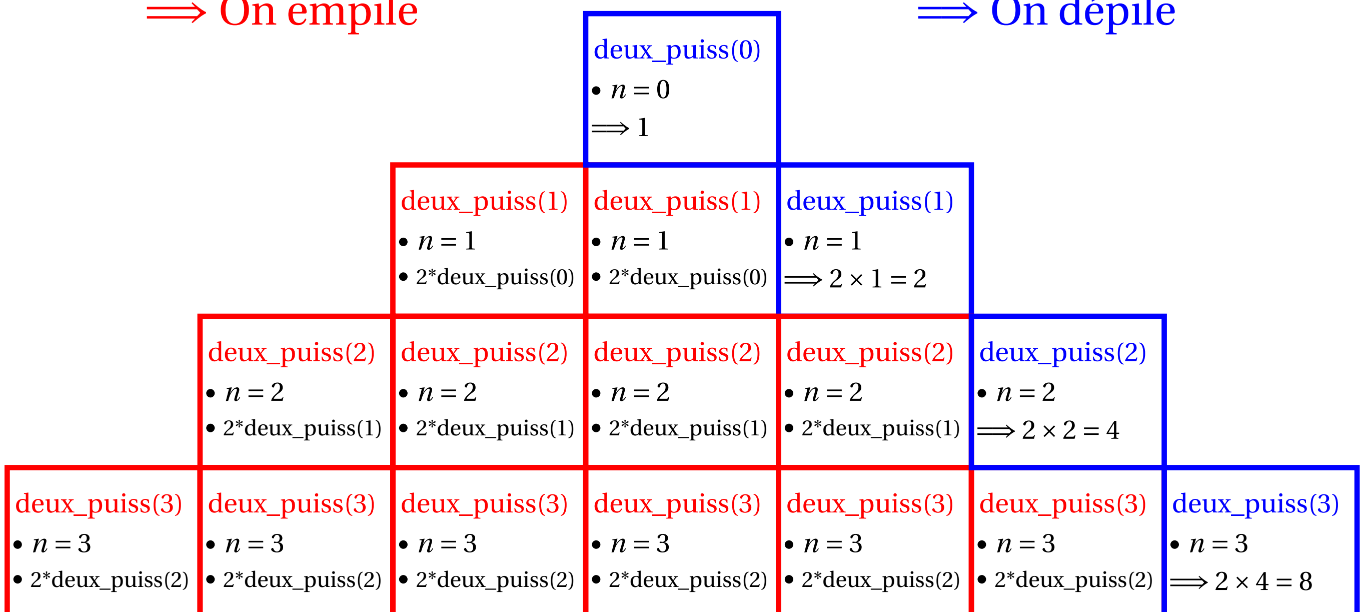
```
# Dans la console PYTHON
>>> deux_puissance(3)
8
```

### 1.3 Pile d'exécution

```
def deux_puiss(n):
    if n==0: # Cas de base
        return 1
    else: # Appel récursif (appel interne)
        return 2*deux_puiss(n-1)
```

Voyons comment va fonctionner cette fonction $deux_puiss(n)$.

1. Lors de l'appel une **structure de Pile** est utilisée.
2. L'idée est que comme pour une pile d'assiettes, on peut :
   - Soit empiler un objet en haut de la pile;
   - Soit retirer un objet du haut.
3. La pile d'exécution est limitée par défaut à 1 000 sous python (voir 997 sous repl?), au delà on a une erreur "*RecursionError : maximum recursion depth exceeded in comparison*".




On obtient bien le résultat attendu et observé :

```
# Dans la console PYTHON
>>> deux_puiss(3)
8
```

### 1.4 Écriture d'une fonction récursive

Pour écrire une fonction récursive :

1. On détermine un **cas de base**, c'est à dire une valeur de l'argument pour laquelle le problème se résout immédiatement (renvoie une valeur).
2. Traiter attentivement le **cas récursif** du passage des valeurs renvoyées par l'appel précédent à l'appel suivant.

**Exemple**

Avec l'exemple classique de la fonction puissance de deux qui retourne $2^n$.
Cette fonction peut-être définie par une fonction récursive car :

1. Un **cas de base** est :
$$2^0 = 1$$
2. **Égalité de récurrence** :
$$2^n = 2 \times 2^{n-1}, \; pour\, n > 0$$

```python
def deux_puissance(n):
    if n==0: # Cas de base
        return 1
    else: # Appel récursif (appel interne)
        return 2*deux_puissance(n-1)
```

## 2 Preuves de terminaison et de correction

Remarque : Cette partie peut être omise dans un premier temps.

**Méthode**

Pour prouver qu'un algorithme récursif fonctionne on doit prouver qu'il vérifie deux propriétés :

1. **Terminaison** : l'algorithme doit se terminer.
2. **Correction (partielle)** : si l'algorithme se termine, il doit renvoyer ce que l'on souhaite. Pour prouver cette **correction partielle** il faut montrer que si les **appels internes** renvoient la bonne valeur, alors la fonction aussi, c'est le même principe qu'une démonstration par récurrence en mathématiques.

Nous pouvons démontrer que l'algorithme $deux_puissance(n)$ est valide. Pour cela nous allons prouver par récurrence que $deux_puissance(n)$ renvoie vraiment $2^n$, pour simplifier nous écrirons cette assertion : $deux_puissance(n) = 2^n$.

```python
def deux_puissance(n):
    if n==0: # Cas de base
        return 1
    else: # Appel récursif (appel interne)
        return 2*deux_puissance(n-1)
```

1. **Correction**.
   — **Initialisation (cas de base)** : pour $n = 0$ on a bien

   $$deux_puissance(0) = 1 \quad \text{et} \quad 2^0 = 1$$

   — **Conservation** : (on suppose que les appels internes renvoient bien ce qu'il faut).
   Si on suppose que pour $n$ fixé les **appels internes récursifs sont valides** soit :

   $$deux_puissance(n-1) = 2^{n-1}$$

   alors puisque notre relation de récurrence est :

   $$deux_puissance(n) = 2 \times deux_puissance(n-1)$$

   On obtient bien en utilisant notre hypothèse de récurrence (c.a.d en supposant nos appels internes valides) :

   $$deux_puissance(n) = 2 \times 2^{n-1} = 2^n$$

2. **Terminaison**.
   L'algorithme se termine car à chaque tour de boucle $n$ diminue de 1 et on fini par arriver au return du cas terminal lorsque $n = 0$ si on a fourni initialement un argument positif pour $n$.

# 3 Exercices sur la récursivité

## 3.1 Fonction $a^n$

**Exercice 1**

1. Écrire une fonction récursive $puissance(a, n)$ qui renvoie $a^n$.
2. Démontrer la terminaison et la correction de votre algorithme récursif.

## 3.2 Factorielle

On note $n!$ (se lit « factoriel $n$ ») le nombre $1 \times 2 \times 3 \times \cdots \times n$, pour tout entier naturel $n > 0$. Par convention on définit :

$$\begin{cases} 0! = 1 \\ n! = 1 \times 2 \times 3 \times \cdots \times n \quad , n \in \mathbb{N}^* \end{cases}$$

**Remarque historique**

La **notation factorielle** est introduite par le mathématicien Christian KRAMP (1760-1826) en 1808 dans *Éléments d'arithmétique universelle* (1808).

**Exercice 2**

1. Calculer 1! , 2! , 3! , 4! et 5!.
2. Soit $n \in \mathbb{N}^*$, exprimer $n!$ en fonction de $(n-1)!$.
3. Écrire une fonction **fact(n)** et **fact_recur(n)** qui renvoient $n!$, avec $n \geq 0$ :
   (a) En utilisant une boucle ;
   (b) Avec une fonction récursive.
4. Démontrer la terminaison et la correction de votre algorithme récursif.

## 3.3 Une somme

**Exercice 3**

Soit $(S_n)$ la suite définie pour $n$ entier non nul par :

$$S_n = 1 + 2 + 3 + \cdots + n$$

1. Calculer $S_1$, $S_2$, $S_3$, $S_4$ et $S_5$.
2. Soit $n \in \mathbb{N}^*$, exprimer $S_n$ en fonction de $S_{n-1}$.
3. Écrire des fonctions **s(n)** et **s_recur(n)** qui renvoient $S_n$, avec $n \geq 0$ :
   (a) En utilisant une boucle ;
   (b) Avec une fonction récursive.
4. Démontrer la terminaison et la correction de votre algorithme récursif.

### 3.4 PGCD

**Exercice 4**

1. Écrire en python une fonction récursive **pgcd(a,b)** renvoyant le plus grand diviseur commun de deux nombres $a$ et $b$.
   Pour cela on utilisera le résultat mathématique suivant :

   « $pgcd(a,b) = pgcd(b,r)$ » , où $r$ reste de la division euclidienne de $a$ par $b$.

### 3.5 Une fonction mystère

*Attention il faut évidemment faire cet exercice sur feuille, vous pourrez ensuite vérifier sur votre éditeur Python vos résultats mais l'objectif est de s'entraîner à une évaluation écrite qui pourra proposer un exercice de ce type.*
On considère le programme suivant :

```python
# Dans l'éditeur PYTHON
def f(a,b):
    """
    In: a et b sont des entiers strictement positifs
    Out: ????
    """
    if b==1:
        return a
    else:
        return a+f(a,b-1)
```

**Exercice 5**

1. Quel résultat retourne $f(7,9)$ ?
2. En déduire la signification de la valeur renvoyée par cette fonction pour deux entiers naturels $a$ et $b$ quelconques.
3. Démontrer la terminaison et la correction de cet algorithme récursif.

### 3.6 La suite de Fibonacci

La suite de Fibonacci est une suite d'entiers dans laquelle chaque terme est la somme des deux termes qui le précèdent. Elle commence généralement par les termes 0 et 1 .

Elle doit son nom à **Leonardo Fibonacci** (v. 1175 à Pise - v. 1250) un mathématicien italien qui avait pour nom d'usage ń Leonardo Pisano ż ou ń Léonard de Pise ż

Fibonacci dans un problème récréatif posé dans l'ouvrage *Liber abaci* (1202), décrit la croissance d'une population de lapins :
« *Un homme met un couple de lapins dans un lieu isolé de tous les côtés par un mur. Combien de couples obtient-on en un an si chaque couple engendre tous les mois un nouveau couple à compter du troisième mois de son existence* ? »

*Leonardo Fibonacci (1175-1250)*

**Exercice 6**

On définit donc la suite $(F_n)$ pour $n \geq 2$ par :

$$\begin{cases} F_0 = 0\,;\; F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$

1. Calculer les 5 premiers termes de la suite.
2. Écrire des fonctions **F(n)** et **F_recur(n)** qui renvoient le terme de rang $n$, avec $n \geq 0$ :
   (a) En utilisant une boucle ;
   (b) Avec une fonction récursive.
3. On note $\phi(n) = \dfrac{F_{n+1}}{F_n}$. Écrire une fonction **phi(n)** qui renvoie le rapport $\dfrac{F_{n+1}}{F_n}$, avec $n \geq 1$.
   Conjecturer la limite de ce rapport.

### 3.7 Approximation de la racine carrée.

Le but de cet exercice est d'écrire une fonction qui calcule une valeur approchée de la racine carrée d'un nombre. Soit $x$ un nombre réel positif, une valeur approchée de $\sqrt{x}$ est donnée par le calcul des valeurs de la suite $(U_n)$ définie pour $n$ entier par :

$$U_n = \begin{cases} 1 & si\;\; n = 0 \\ \dfrac{U_{n-1} + \dfrac{x}{U_{n-1}}}{2} & si\;\; n > 0 \end{cases}$$

**Exercice 7**

1. Écrire une fonction qui, étant donne un nombre $x$ et un entier $n$, renvoie l'approximation au rang $n$ de $\sqrt{x}$.
2. Démontrer la terminaison et la correction de votre algorithme récursif.

### 3.8 D'autres fonctions mystères

**Exercice 8**

Que fait la fonction suivante ?

```python
# Dans l'éditeur PYTHON
def mystere(L,M=[]):
    """
    In: L est une liste
    Out: ????
    """
    if L==[]:
        return M
    a=L.pop(0)
    if a not in M:
        M.append(a)
    return mystere( L , M )
```

**Exercice 9**

Que fait la fonction suivante ?

```python
# Dans l'éditeur PYTHON
def mystere(L):
    """
    In: L est une liste
    Out: ????
    """
    if len(L)==1:
        return L[0]

    if L[0]<L[1]:
        L.pop(1)
    else:
        L.pop(0)
    return mystere( L )
```

### 3.9 Permutations

La notion de permutation exprime l'idée de réarrangement d'objets discernables. Une permutation d'objets distincts rangés dans un certain ordre correspond à un changement de l'ordre de succession de ces objets. Par exemple si $A = [1,2,3]$, l'ensemble des permutations de A est

$$[[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]$$

**Exercice 10**

Niveau avancé :

Écrire une fonction qui prend une liste A de taille $n$ et renvoie la liste des permutations de A.

↩ **Fin du cours** ↪