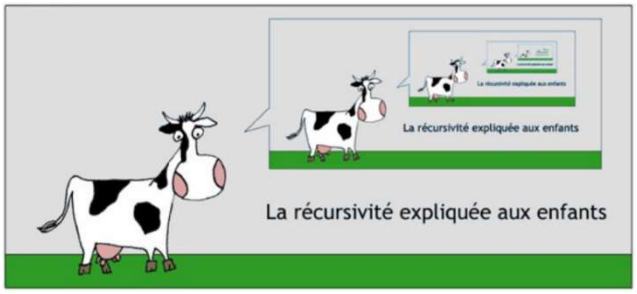
La récursivité

1. La récursivité, c'est quoi ?



En mathématiques, en informatique, en biologie, mais aussi dans notre quotidien, nous faisons souvent face à des situations où un problème doit être résolu en utilisant une méthode de résolution qui est répétée plusieurs fois :

- dans l'itération, cette méthode est appliquée par paliers de façon séquentielle ;
- dans la récursivité, la méthode s'appelle elle-même.

La récursivité se retrouve dans beaucoup de domaines.

1.1. Définition récursive

Une définition récursive est une définition dans laquelle intervient le nom qu'on est en train de définir.

Exemples

Une personne A est un **descendant** d'une autre personne B si et seulement si :

- soit A est un enfant (fils ou fille) de B ;
- soit A est un enfant (fils ou fille) d'un **descendant** de B.

Un **entier naturel positif ou nul** est :

- Soit l'entier 0 ;
- Soit le successeur d'un **entier naturel positif ou nul**.

1.2. Arts

Image contenant une image similaire.
Selfie pris dans un miroir.

1.3. Linguistique

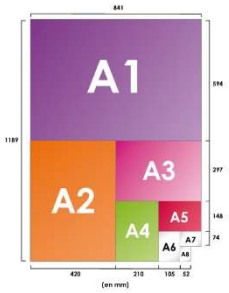
Dictionnaire : chaque mot du dictionnaire est défini par d'autres mots eux-mêmes définis par d'autres mots dans ce même dictionnaire.

1.4. Nature

Chou romanesco.
Fougère.

1.5. Format A4

La norme DIN-A détermine la taille du papier. Le format A0 qui est le plus grand format normalisé (1 mètre carré de surface) se décline jusqu'au format A10. La longueur du format inférieur est systématiquement égale à la largeur du format supérieur. Le format inférieur est donc obtenu en pliant le format supérieur en deux dans sa largeur.
Quel que soit le format, on trouve toujours le rapport $\sqrt{2}$ entre longueur et largeur.



1.6. Mathématiques

Suites récurrentes.
Fractales.

2. Fonctions récursives

En informatique, une fonction qui s'appelle elle-même est dite **récursive**.

Les fonctions récursives, permettent, pour certains types de problèmes :

- d'écrire plus facilement des programmes ;
  - l'écriture se déduit de la définition de la fonction ;
- de vérifier plus facilement que les programmes sont corrects ;
  - preuve par induction.

3. Exemple d'une fonction récursive : affichage de nombres

On veut écrire un programme qui :

- étant donné un entier n > 0,
- affiche les nombres de 1 à n par ordre croissant.

3.1. En programmation itérative

```
# avec une boucle while
n = int(input("Donner un entier : "))
i = 1
while (i <= n):
    print("i : ",i)
    i = i + 1

# avec une boucle for
n = int(input("Donner un entier : "))
for i in range(1, n+1):
    print("i : ", i)
```

✍ Tester le bon fonctionnement des programmes ci-dessus.

3.2. En programmation récursive

Avec une fonction récursive de paramètre n :

- ce qu'on sait faire facilement : quand le paramètre vaut 1,
  - on affiche 1 et c'est fini ;
- quand n > 1,
  - on affiche les entiers de 1 à n – 1,
  - puis on affiche n.

Le premier cas est appelé **cas trivial** ou **cas d'arrêt de la récursivité** :

- on s'arrête quand n vaut 1.

Le deuxième cas est appelé le **cas récursif** :

- on ramène le problème à un sous-problème plus simple, c'est-à-dire de taille plus petite :
  - on fait un appel récursif à la fonction avec n – 1,
    - on affichera donc les entiers de 1 à n – 1 ;
  - on affiche ensuite n.

✍ Tester le bon fonctionnement du programme récursif ci-après.

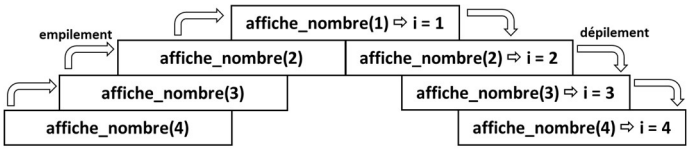
```
# avec une fonction récursive
def affiche_nombre(n):
    if n==1:
        print("i : ",1)
    else:
        affiche_nombre(n - 1)
        print("i : ",n)

affiche_nombre(4)
```

3.3. Comment ça marche ?

✍ Tester le programme récursif ci-dessus en utilisant le mode Debug de Thonny en pas à pas.

Le déroulement est le suivant :
appel à **affiche_nombre(4)**
  appel à **affiche_nombre(3)**
    appel à **affiche_nombre(2)**
      appel à **affiche_nombre(1)**
      i = 1
      retour à **affiche_nombre(2)**
    i = 2
    retour à **affiche_nombre(3)**
  i = 3
  retour à **affiche_nombre(4)**
i = 4



4. Autre exemple : calcul d'un entier a à la puissance n

Soient a > 0 et n > 0, la puissance $n^{ième}$ de a est définie par :

- pour n = 0 : $a^0 = 1$;
- pour n > 0 : $a^n = a \times a^{n-1}$.

Cette définition mathématique comporte :

- un point de départ (n = 0) pour lequel on a directement le résultat ($a^0 = 1$) ;
- un calcul de la puissance nième de a qui fait appel au calcul de la puissance $(n - 1)^{ième}$ de a.

C'est donc une définition récursive avec :

- un cas d'arrêt de la récursivité : n = 0 ;
- un cas de calcul récursif $an = a \times a^{n-1}$ dans lequel on fait diminuer la taille de problème (n - 1).

On constate donc que, pour tout n > 0, le calcul finira par s'arrêter quand n atteindra le cas d'arrêt et la valeur 0.

On se calque sur la définition mathématique pour écrire en Python la définition de la fonction **puissance** à 2 paramètres a (avec a > 0) et n (avec n> = 0) :

- un cas d'arrêt de la récursivité : **n = 0**
  - le résultat est 1
- un cas de calcul récursif : **n > 0**
  - pour calculer a^n, on multiplie n avec le résultat de calcul de a^{n-1}.

✍ Vérifier le bon fonctionnement du programme récursif ci-dessous en utilisant le mode Debug de Thonny en pas à pas .

```
# calcul de a à la puissance n, pour a>0 et n>=0
def puissance(a,n):
    if (n == 0):
        # cas d'arrêt
        return 1
    else:
        # n > 0 cas récursif
        return a * puissance(a,n-1)

puissance(2,4)
puissance(2,0)
```

5. Démarche pour écrire une fonction récursive

1. On commence par chercher le(s) cas simple(s), c'est-à-dire celui(ceux) qui ne nécessite(nt) pas de rappeler récursivement la fonction :
   - pour l'affichage, c'est le cas ou n = 1 (on affiche 1) ;
   - pour la puissance, c'est le cas où n = 0 (n sait que $a^0 = 1$).

2. On cherche ensuite le(s) sous-problèmes(s) récursif(s) (sous-problème de taille réduite par rapport au problème) pour rappeler la fonction récursive pour chaque sous-problème à résoudre :
   - pour l'affichage, c'est afficher les n - 1 entiers ;
   - pour la puissance, c'est calculer a^{n-1}.

3. Vérifier que la fonction se termine :
   - atteint-on au moins un cas d'arrêt ?

Problème de la terminaison
- Une fonction récursive doit obligatoirement avoir au moins un cas d'arrêt.
- Les appels récursifs doivent permettre d'atteindre ce(s) cas d'arrêt.

6. Danger de la récursivité

Lors de l'exécution d'un programme un emplacement mémoire (la pile d'exécution) est destiné à mémoriser les paramètres, les variables locales ainsi que les adresses de retour des fonctions en cours d'exécution.
A chaque appel récursif, les éléments cités ci-dessus sont mémoriser. Mais la pile a une taille fixée, une mauvaise utilisation de la récursivité peut entraîner un débordement de pile.

✍ Tester la fonction `affiche_nombre()` avec la valeur 1000 puis la valeur 10 000.

- Avec la valeur 1000, le programme s'exécute normalement.
- Avec la valeur 10 000, il y a un débordement de la pile (la profondeur maximum de récursivité est atteinte).

```
RecursionError: maximum recursion depth exceeded in comparison
```

Source : cours DIU Université de Toulouse