INFO-H-100 - Programmation
TP 16 — La récursivité

Lorsque l'exercice demande d'écrire une fonction, écrivez la fonction demandée et testez-la avec plusieurs valeurs pertinentes.

Ex. 1. Pour chaque exemple donné ci-dessous, qu'affiche l'appel `mystere(5)` ?

```
def mystere1(n):
    if n >= 1:
        mystere1(n-1)
        print(n,end=" ")
```

```
def mystere2(n):
    if n >= 1:
        print(n,end=" ")
        mystere2(n-1)
```

```
def mystere3(n):
    if n >= 1:
        print(n,end=" ")
        mystere3(n+1)
```

```
def mystere4(n):
    if n >= 1:
        mystere4(n+1)
        print(n,end=" ")
```

Ex. 2. Ecrivez une fonction récursive qui renvoie la factorielle d'un nombre entier positif.

Ex. 3. Le $n^{\text{ème}}$ nombre de Fibonacci est défini de la manière suivante : $\forall n \geq 2 : F_n = F_{n-1} + F_{n-2}$ avec $F_0 = 0$ et $F_1 = 1$. Ecrivez une fonction récursive qui renvoie la valeur du $n^{\text{ème}}$ nombre de Fibonacci. Développez l'exécution de `fib(5)`. Combien d'additions doit effectuer votre algorithme pour arriver au résultat ?

Ex. 4. Pour tous entiers naturels n et k, les coefficients binomiaux $\binom{n}{k}$ donnent le nombre de sous-ensembles différents à k éléments que l'on peut former à partir d'un ensemble contenant n éléments.

Ils sont définis récursivement par :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \quad \text{pour tous les entiers } n, k > 0,$$

Les conditions initiales de cette définition récursive sont : $\binom{n}{0} = 1$ si $n \geq 0$, et $\binom{0}{k} = 0$ si $k > 0$. Ecrivez une fonction récursive `binom(n, k)` en vous basant sur cette propriété. Faites bien attention aux conditions de terminaison.

Ex. 5. Soit n, un nombre entier positif, écrivez une fonction récursive qui calcule a^n pour a donné.

Ex. 6. Servez-vous du fait que, par exemple, $x^{24} = (x^{12})^2$ pour diminuer dramatiquement le nombre de multiplications et d'appels récursifs dans la solution de l'exercice précédent.

Ex. 7. Si $p > 0$ et $n > 1$, p est une puissance de n ssi p égal 1 ou p est divisible par n et p/n est une puissance de n. Ecrivez une fonction récursive `isPower(p, n)`.

Ex. 8. L'algorithme d'Euclide permet de calculer très rapidement le pgcd de deux nombres naturels. Il peut être défini par les formules suivantes : $pgcd(a, 0) = a$ et $pgcd(a, b) = pgcd(b, a \ modulo \ b)$. Ecrivez une fonction récursive `pgcd(a, b)` qui calcule le pgcd de `a` et `b`.

Ex. 9. La parité d'un nombre n, positif ou nul peut être définie de la façon suivante : n est pair, ssi n est nul ou $n - 1$ est impair. n est impair ssi n est différent de 0 et $n - 1$ est pair. Transcrivez ces définitions sous la forme de fonctions récursives. Attention : Du fait des limitations de Python dans le traitement des fonctions récursives, ces définitions ne s'appliquent que pour un petit n ($n < 900$)

Ex. 10. Écrivez une version récursive de la fonction `findDicho` qui détermine par dichotomie si, oui ou non, une valeur donnée existe dans une liste triée. Utilisez une fonction auxiliaire qui reçoit les limites dans lesquelles la recherche doit être effectuée.

Ex. 11. Ecrire une fonction `my_range(a,b)` qui produit une liste ordonnée contenant tous les entiers dans $[a, b[$.

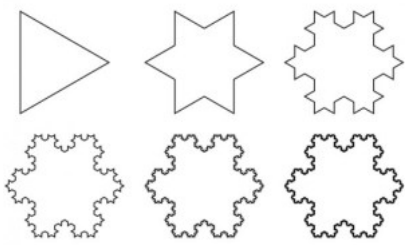
Ex. 12. Ecrivez la fonction `perms(n)` qui renvoie une liste de toutes les permutations de $1..n$. Par exemple

```
>>> perms(3)
[[3, 2, 1], [2, 3, 1], [2, 1, 3], [3, 1, 2], [1, 3, 2], [1, 2, 3]]
>>> perms(0)
[[]]
```

Exercices complémentaires (facultatifs)

Ex. 13. Choisissez quelques uns des exercices précédents et programmez leur variante itérative. Comparez-la en termes de lisibilité et de performance à la version récursive.

Ex. 14. Retournez à l'exercice 3 de la séance 13 (Turtle).
- Etudiez la fonction proposée sous l'aspect de la récursivité.
- Adaptez cette fonction pour tracer des flocons de Koch (Figure ci-dessous). Quelle est la différence d'implémentation entre les deux fonctions (Koch et arbre) ?



Ex. 15. Programmez la résolution du problème des tours de Hanoï.

INFO-H-100 - Programmation
TP 16 — La récursivité
Corrections

Solution de l'exercice 2:

```
def fact(n):
    if n < 2:
        res = 1
    else:
        res = n * fact(n-1)
    return res
```

Solution de l'exercice 3:

```
def fib1(n):
    """n >= 0"""
    if n == 0:
        res = 0
    elif n == 1:
        res = 1
    else:
        res = fib1(n-1) + fib1(n-2)
    return res
```

```
def fib2(n):
    def aux(n, a, b):
        if n == 0:
            res = a
        else:
            res = aux(n-1, b, a+b)
        return res
    return aux(n, 0, 1)
```

Solution de l'exercice 4:

```
def binomial(n, k):
    if k == 0:
        res = 1
    elif n == 0:
        res = 0
    else:
        res = binomial(n-1, k) + binomial(n-1, k-1)
    return res
```

Solution de l'exercice 5:

```
def exp(a, n):
    """ a exposant n"""
    if n == 0:
        res = 1
    else:
        res = a * exp(a, n-1)
    return res
```

Solution de l'exercice 6:

```
def exp2(a, n):
    """ a exposant n"""
    if n == 0:
        res = 1
    elif n % 2 == 1:
        res = a * exp2(a, n-1)
    else:
        res = exp2(a, n/2)**2
    return res
```

Solution de l'exercice 7:

```
def is_power(p, n):
    """p est-il une puissance de n?
       p > 0, n > 1
    """
    return p == 1 or (p % n == 0 and is_power(p / n, n))
```

Solution de l'exercice 8:

```
def gcd(a, b):
    if b == 0:
        res = a
    else:
        res = gcd(b, a % b)
    return res
```

Solution de l'exercice 9:

```
def is_even(n):
    return n == 0 or is_odd(n - 1)

def is_odd(n):
    return n != 0 and is_even(n - 1)
```

Solution de l'exercice 10:

```
def find_dicho(ls, val):
    """True ssi val apparait dans ls
    Pre: ls est triee
    """
    def aux(bi, bs):
        if bi == bs:
            res = False
        else:
            m = (bs + bi) // 2
            if val == ls[m]:
                res = True
            elif val < ls[m]:
                res = aux(bi, m)
            else:
                res = aux(m + 1, bs)
        return res
    return aux(0, len(ls))
```

Solution de l'exercice 11:

```
def my_range(a, b):
    if b <= a:
        res = []
    else:
        ls = my_range(a, b-1)
        ls.append(b-1)
        res = ls
    return res
```

Solution de l'exercice 12:

```
def perms(n):
    if n < 1:
        res = [[]]
    else:
        ls = perms(n-1)
        res = []
        for ss in ls:
            for pos in range(len(ss)+1):
                tmp = ss[:]
                tmp.insert(pos, n)
                res.append(tmp)
    return res
```