FACULTÉ DES SCIENCES ET TECHNOLOGIES | Université de Lille

**L1 MIASH/SESI/PEIP - 2022–2023**

**Algorithmes et Programmation**

# TD : Recursivite

**Exercice 1** *Que calculent les fonctions* `est_pair` *et* `est_impair` *?*

Voici la déclaration en PYTHON de deux fonctions :

```
def est_pair(n):
    return n == 0 or not est_impair(n)

def est_impair(n):
    return not (n == 0 or est_pair(n))
```

**Question 1** Que pensez-vous des expressions calculées par chacune des deux fonctions dans les deux cas de base et récursif ?

---

**Exercice 2** *Encore la fonction* `est_pair`

Voici la déclaration en PYTHON de la fonction `est_pair` :

```
def est_pair(n):
    if n == 0:
        res = True
    else:
        res = est_pair(n - 2)
    return res
```

Que pensez-vous de cette fonction ?

**Exercice 3** *Somme de deux entiers*

**Question 1** Proposez un algorithme récursif de calcul de la somme de deux entiers naturels $a$ et $b$ en supposant que les seules opérations de base dont vous disposez sont
— l'ajout de 1 à un entier $a$ : $a+1$
— le retrait de 1 à un entier $a$ : $a-1$
— et les comparaisons à 0 d'un entier $a$ : $a=0$, $a>0$ et $a<0$.

---

**Question 2** Comment étendre cette fonction aux entiers de signe quelconque ?

---

**Exercice 4** *Produit de deux entiers*

**Question 1** Proposez un algorithme récursif de calcul du produit de deux entiers naturels $a$ et $b$ en supposant que les seules opérations de base dont vous disposez sont
— la somme de deux entiers $a$ et $b$ : $a+b$
— le retrait de 1 à un entier $a$ : $a-1$
— et la comparaison à 0 d'un entier $a$ : $a=0$.

---

## Exercice 5 *Puissance entière d'un nombre réel*

**Question 1** Proposez un algorithme récursif de calcul de la puissance $n$-ième ($n \in \mathbb{N}$) d'un nombre réel $a$ en supposant que les seules opérations de base dont vous disposez sont
— le produit de deux réels $a$ et $b$ : $a \times b$
— le retrait de 1 à un entier $a$ : $a - 1$
— et la comparaison à 0 d'un entier $a$ : $a = 0$.

---

**Question 2**

En utilisant des élévations au carré et des multiplications, exprimez $a^n$ en fonction de $a^k$ et $a$ lorsque $n = 2k$, puis lorsque $n = 2k + 1$.

En déduire un autre algorithme récursif pour calculer $a^n$.

Donnez, en fonction de $n$, un encadrement du nombre de multiplications effectuées par cet algorithme. Comparez avec l'algorithme précédent.

---

## Exercice 6 *Nombre de chiffres d'un entier*

Réalisez une fonction récursive qui calcule le nombre de chiffres de l'écriture décimale d'un nombre entier passé en paramètre.

```
>>> nbre_chiffres(0)
1
>>> nbre_chiffres(2020)
4
```

## Exercice 7 *Divisibilité par trois*

Il est bien connu qu'un nombre entier est divisible par trois si et seulement si la somme de ses chiffres l'est aussi. Voilà donc un bel algorithme récursif.

**Question 1** Étudiez le (ou les) cas de base nécessaire(s). Qu'est ce qui garantit l'arrêt de cet algorithme ?

---

**Question 2** Programmez une fonction récursive de calcul de la somme des chiffres d'un entier et un prédicat récursif pour tester la divisibilité par trois.

```
>>> somme_chiffres(9876)
30
>>> est_divisible_par_trois(9876)
True
>>> somme_chiffres(9877)
31
>>> est_divisible_par_trois(9877)
False
```

---

## Exercice 8 *Algorithme d'Euclide*

L'algorithme d'Euclide permet de calculer le pgcd de deux nombres entiers, c'est–à–dire le plus grand entier positif divisant ces deux nombres, par des divisions successives.

Voici le déroulement de cet algorithme pour le calcul du pgcd de $a = 119$ et $b = 544$

$$\begin{array}{rcrcrcr} 119 & = & 544 & \times & 0 & + & 119 \\ 544 & = & 119 & \times & 4 & + & 68 \\ 119 & = & 68 & \times & 1 & + & 51 \\ 68 & = & 51 & \times & 1 & + & \boxed{17} \\ 51 & = & \boxed{17} & \times & 3 & + & 0 \end{array}$$

Le pgcd de 119 et 544 est le dernier reste non nul, c'est–à–dire 17.

Le pgcd n'est pas défini lorsque les deux nombres sont nuls.

**Question 1** Exprimez de manière récursive cet algorithme. Vous pourrez supposer que les deux entiers $a$ et $b$ sont positifs ou nuls, et que l'un au moins de ces deux entiers n'est pas nul.

---

**Question 2** Codez cet algorithme en PYTHON en utilisant l'opérateur modulo.

---

## Exercice 9 *Coefficient binomial*

Les coefficients binomiaux sont définis pour les entiers naturels $n \geq p \geq 0$ par $\binom{n}{p} = \frac{n!}{p!(n-p)!}$. Une relation de récurrence valable pour $n > p > 0$ est

$$\binom{n+1}{p+1} = \binom{n}{p} + \binom{n}{p+1}$$

**Question 1** Réalisez une fonction effectuant un calcul récursif des coefficients binomiaux qui n'utilise que l'addition des entiers.

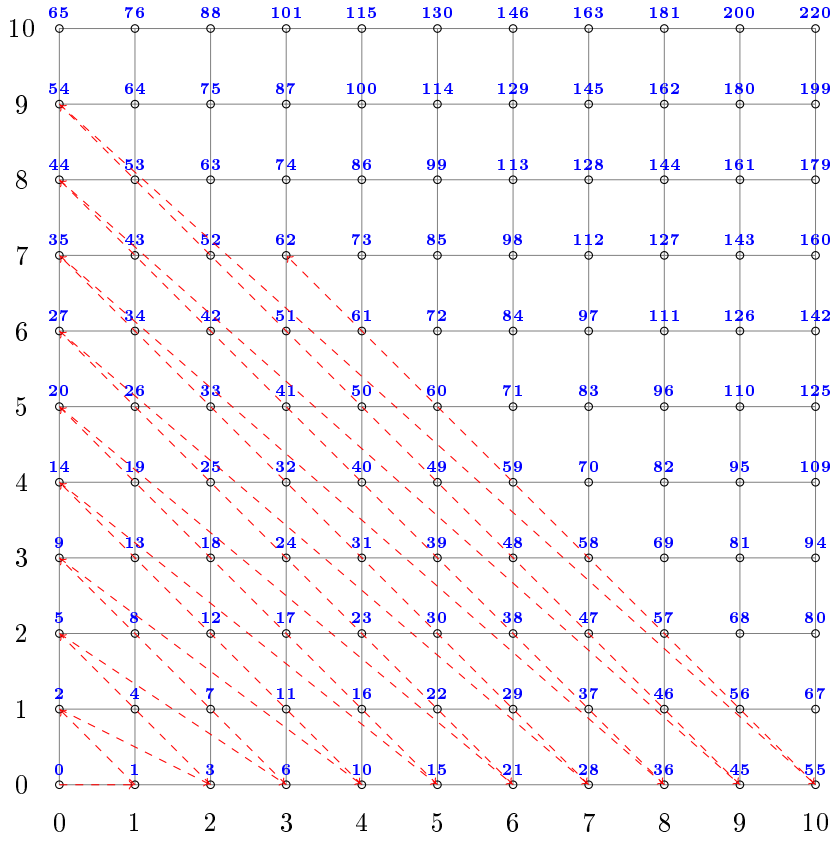
---

**Question 2** Tracez le calcul du coefficient binomial pour $n = 5$ et $p = 2$.

---

## Exercice 10 *Numérotation de Cantor*

Les points à coordonnées entières peuvent être numérotés comme la figure ci-dessous le suggère.




**Question 1** Programmez une fonction récursive qui prend un point à coordonnées entières en paramètre et renvoie le numéro de ce point.

---

**Question 2** Qu'est-ce qui garantit l'arrêt de cet algorithme ?

---

**Question 3** Programmez la fonction réciproque qui à partir d'un numéro (entier naturel) renvoie le point à coordonnées entières ayant ce numéro.

---

**Exercice 11** *Palindromes*

Un *palindrome* est un mot dont les lettres lues de gauche à droite sont les mêmes que celles lues de droite à gauche. Les mots `radar`, `elle`, `été`, `ici` sont des palindromes.

**Question 1** Réalisez un prédicat qui teste si un mot est un palindrome.

---

**Exercice 12** *Occurrences dans une chaîne*

**Question 1**

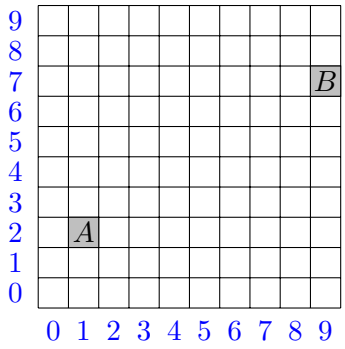
Réalisez une fonction `nb_occurrences` à deux paramètres : une chaîne de caractères de longueur quelconque et une chaîne de 1 caractère qui renvoie le nombre d'occurrences du caractère (second paramètre) dans la chaîne (premier paramètre).

Par exemple, `nb_occurrences ('abcdebdb','z')` doit renvoyer 0 et `nb_occurrences ('abcdebdb','b')` doit renvoyer 3.

---

**Exercice 13** *Tracer un segment*

En informatique graphique, tous les points ont des coordonnées entières. Soit donc deux points $A$ de coordonnées $(x_A, y_A)$ et $B$ de coordonnées $(x_B, y_B)$. Proposez un algorithme récursif pour tracer à l'écran le segment $[A, B]$. La commande primitive que vous utiliserez est la commande `plot(x, y)` qui dessine le pixel de coordonnées $(x, y)$.




**Exercice 14** *Coloriage*

Supposons données les coordonnées (entières) $(x, y)$ d'un pixel situé à l'intérieur d'une région du plan délimitée par une courbe fermée. Les points sur la courbe délimitant la région sont de couleur noire, et ceux à l'intérieur sont blancs. On souhaite donner la couleur rouge à tous les points à l'intérieur de la région.

Proposez un algorithme récursif pour effectuer ce coloriage. (Vous pourrez utiliser deux fonctions `get_color(x, y)` qui renvoie la couleur du pixel de coordonnées $(x, y)$ et `set_color(x, y, color)` qui fixe la couleur du pixel de coordonnées $(x, y)$.)

**Exercice 15** *Somme des éléments d'une liste*

Donnez une version récursive du calcul de la somme des éléments d'une liste de nombres.

**Exercice 16** *Inversion des éléments d'une liste*

Donnez un algorithme récursif de l'inversion de l'ordre des éléments d'une liste.

**Exercice 17** *Alterner les éléments de deux listes*

Réalisez une fonction nommée `alterne` paramétrée par deux listes `l1` et `l2` qui renvoie une liste dont les éléments sont ceux de ces deux listes dans un ordre alterné.

```
>>> alterne([1,3,5,7,9], [2,4,6])
[1, 2, 3, 4, 5, 6, 7, 9]
```

**Exercice 18** *Retour sur les anagrammes*

On rappelle que deux mots sont *anagrammes* l'un de l'autre si on peut former l'un des mots avec les lettres de l'autre. Ainsi, `égratigna` et `gagnerait` sont des anagrammes.

Proposez une version récursive du prédicat nommé `sont_anagrammes` qui détermine si deux mots sont des anagrammes (vous pouvez négliger les différences entre lettres capitales ou non, accentuées ou non).

Indication : utilisez la méthode `index` des chaînes de caractères.

```
>>> sont_anagrammes('egratigna','gagnerait')
True
>>> sont_anagrammes('ensabler','ebranlees')
False
```

**Exercice 19** *cosinus, sinus*

Écrire une fonction récursive `cos_sin_nx` qui prend en paramètre un entier $n$ et un couple de valeurs réelles représentant les cosinus et sinus d'un angle $\theta$ et qui a pour résultat le couple de valeurs $(\cos n\theta, \sin n\theta)$.

Vous pouvez utiliser les formules trigonométriques suivantes :

$$\cos(nx) = \cos((n-1)x)\cos(x) - \sin((n-1)x)\sin(x)$$
$$\sin(nx) = \sin((n-1)x)\cos(x) + \cos((n-1)x)\sin(x)$$

Ainsi avec $n = 3$ et $\theta = \frac{\pi}{4}$, l'appel `cos_sin_nx(n, (cos(theta), sin(theta)))` renvoie le couple de flottants (-0.707..., 0.707...) (valeur approchée de $(\cos\frac{3\pi}{4}, \sin\frac{3\pi}{4})$).

**Vous n'utiliserez pas** les fonctions `cos` et `sin` définies dans le module `math` de PYTHON.