# Révisions PSI
# Retour sur la sup

*Lycée Châtelet - 357 Rue Marceline - 59500 Douai*

**Les 10 exercices seront à traiter et restitués <u>avant le 25 août</u> par mail (melissa.inglart@free.fr).**

**Ils feront l'objet d'un premier <u>DS lors de la première séance</u> d'informatique et des <u>points bonus</u> à ce DS seront affectés en fonction de la qualité de votre restitution.**

## Exercice 1 : définitions

En reprenant vos cours de l'année de sup, donner les définitions des termes suivants et décrire un exemple associé :

1. Variant de boucle
2. Invariant de boucle
3. Matrice d'adjacence d'un graphe
4. Pile et File (on précisera ici les structures de données associées à la place de la fourniture d'un exemple)

## Exercice 2 : moyenne pondérée

Écrire une fonction `moy_ponderee` qui prend en entrée trois entiers `n1`, `n2`, `n3`, trois poids `w1`, `w2`, `w3` et renvoie la moyenne arithmétique de `n1`, `n2`, `n3` pondérée par les coefficients `w1`, `w2`, `w3`. Vous devez vérifier que la somme des poids est non nulle à l'aide d'un assert `assert`. Par exemple :

`moy_ponderee(5, 6, 7, 1, 1, 1)` vaut `6.0`
`moy_ponderee(5, 3, 7, 6, -7, 2)` vaut `23.0`
`moy_ponderee(1, 2, 3, 4, 5, 6)` vaut `2.1333...`
`moy_ponderee(5, 6, 7, 1, 1, -2)` déclenche une erreur

## Exercice 3 : coefficients binomiaux

Écrire une fonction `binom` qui prend en entrée deux entiers `n, k` et renvoie $\binom{n}{k}$. Votre fonction doit renvoyer un `int` (pas un `float`). De plus, votre fonction déclenchera une erreur si $k < 0$ ou $n < 0$. Par exemple :

| `n` | 7 | 7 | 8 | 5 | -1 | 5 |
|---|---|---|---|---|---|---|
| `k` | 0 | 7 | 4 | 10 | 5 | -2 |
| `binom(n,k)` | 1 (et pas `1.0`) | 1 (et pas `1.0`) | 70 (et pas `70.0`) | 0 | Erreur | Erreur |

<u>On écrira une fonction récursive.</u>

## Exercice 4 : Listes

1. Étant donnés un entier `n` et une liste `L`, on souhaite écrire une fonction `appartient` qui renvoie `True` si `n` appartient à `L` et renvoie `False` sinon.
   Écrire la fonction `appartient` à l'aide d'une boucle `while`, mais sans utiliser l'instruction `break` ni de sortie anticipée de fonction (pas de `return` dans la boucle `while`). La boucle devra s'arrêter dès que l'élément a été trouvé. Par exemple, il ne doit y avoir qu'un seul tour de boucle pour `L = [1,2,3]` et `n = 1`. Proposer des exemples pour votre fonction.
2. Écrire une fonction qui renvoie le maximum d'une liste d'entiers. Votre fonction déclenchera une erreur si la liste est vide.

| L | [] | [1,2,3] | [-8,6,4,0] | [8,6,4,0] | [-4,-3,-8,-1] |
|---|---|---|---|---|---|
| maximum(L) | Erreur | 3 | 6 | 8 | -1 |

3. Écrire une fonction `est_croissante` qui prend en entrée une liste d'entiers `L` et renvoie `True` si `L` est croissante et `False` sinon. Vérifiez que votre fonction renvoie `True` pour les listes :

`[]` `[0]` `[1,2,3]` `[1,1,1]` `[-10,-3,0,0,4,8,8]`

et renvoie `False` pour les listes :

`[0,-1]` `[-3,4,5,4,5,6]` `[1,0,1,2,3,4]` `[-2,-1,0,1,2,3,2]`

## Exercice 5 : Listes en compréhension

Dans cet exercice, il est demandé d'utiliser des compréhensions de listes.

1. Écrire une fonction `mult_elem_liste` qui prend en paramètre un nombre $a$ et une liste d'entiers `L`, et renvoie la liste où chaque élément de `L` a été multiplié par $a$. Par exemple :

`mult_elem_liste(2, [1,2,3])` vaut `[2, 4, 6]`,
`mult_elem_liste(0.5, [])` vaut `[]`,
`mult_elem_liste(0.5, [10,8,6,-8,2,7])` vaut `[5.0, 4.0, 3.0, -4.0, 1.0, 3.5]`,
`mult_elem_liste(-4, [7,4,1,0,-7,-4,-1])` vaut `[-28, -16, -4, 0, 28, 16, 4]`.

2. Écrire une fonction `val_abs` qui prend en entrée une liste de nombres flottants `L` et renvoie la liste obtenue en prenant la valeur absolue de chaque élément de `L`. Par exemple :

`val_abs([])` vaut `[]`, `val_abs([1, 2, 3])` vaut `[1, 2, 3]`,
`val_abs([-5, 3, 0, 5])` vaut `[5, 3, 0, 5]`, `val_abs([-4])` vaut `[4]`.

## Exercice 6 : Dictionnaire

Ecrire une fonction « histogramme » prenant en entrée une chaîne de caractères ne comportant que des lettres minuscules et qui renvoie un dictionnaire tel que ses clés soient les différentes lettres présentes dans la chaîne de caractères. On associera dans ce dictionnaire à chaque lettre son occurrence.

## Exercice 7 : Récursivité et dichotomie

Pour une liste triée :

**Recherche dichotomique.** On rappelle qu'il est possible de tester l'appartenance d'un élément `x` à une liste `L` en temps logarithmique grâce à une recherche par dichotomie. La fonction itérative correspondante est donnée ci-contre.

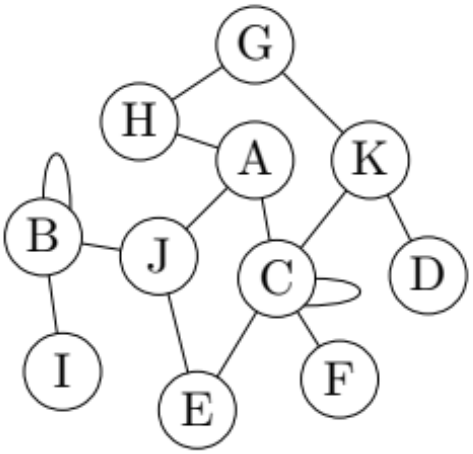
```
def appartient_dicho_it(L, e):
    i = 0
    j = len(L)-1
    while i <= j:
        m = (i+j)//2
        if L[m] == e:
            return True
        elif L[m] < e:
            i = m+1
        else:
            j = m-1
    return False
```

1. Convertir la fonction **appartient_dicho_it** en fonction récursive **appartient_dicho_rec.**

2. On justifiera la complexité de cet algorithme.

## Exercice 8 : Graphes

Pour le graphe ci-contre, lors d'un parcours en largeur ayant pour sommet initial A, quel est l'ordre dans le quel les sommets sont visités ?



*Quelques définitions complémentaires*

**Définition 3.** ***Un tri en place*** est un algorithme tel que la quantité de mémoire utilisée est constante (c'est à dire qu'elle ne dépend pas de la taille de `L`). Notez que dans la "quantité de mémoire utilisée", on ne compte pas la mémoire nécessaire pour stocker `L`. En particulier, un algorithme en place ne peut pas créer de nouvelle liste ou bien de dictionnaire, il doit donc nécessairement modifier la liste `L` directement :

**Définition 4.** ***Un tri par comparaisons*** choisit la manière de trier `L` en se basant uniquement sur le résultat de comparaisons entre éléments de la liste. Par exemple, le tri par insertion est un tri par comparaisons (exercice 1), mais ce n'est pas le cas du tri par comptage (exercice 4). Un tri par comparaisons permet de trier n'importe quel type d'objets sur lequel les opérateurs de comparaisons (`<`, `<=`, ...) sont définis. À l'inverse, le tri par comptage de l'exercice 4 ne permet de trier que des entiers.

**Définition 6.** Un ***tri stable*** est un tri dans lequel l'ordre de deux éléments égaux est le même dans la liste initiale et dans la liste triée.

## Exercice 9

Le principe du tri par sélection est de créer une liste `T` initialement vide et d'ajouter à la fin de `T` le plus petit élément de `L`, puis le deuxième plus petit élément de `L`, puis le troisième plus petit, et ainsi de suite jusqu'à ce que `T` contienne tous les éléments de `L`. Par exemple, voici l'évolution de `T` lors du tri de la liste `[5, 8, 1, 6, 5, 7]` :

`[]` → `[1]` → `[1,5]` → `[1,5,5]` → `[1,5,5,6]` → `[1,5,5,6,7]` → `[1,5,5,6,7,8]`

1. Écrire une fonction `tri_selection` qui trie une liste d'entiers à l'aide du tri par sélection. Si besoin, on pourra utiliser les indications ci-dessous.
2. Quelle est la complexité de votre fonction ?
3. Est-ce un tri en place ? Un tri stable ?

## Exercice 10

Le tri rapide est un algorithme récursif. Si la liste `L` à trier est vide, il suffit de renvoyer la liste vide. Sinon, on choisit l'un des éléments de `L` noté `e` qu'on appelle le pivot (on peut par exemple choisir `L[0]`). Soit `L0` la liste `L` privée de `e`, soit `L1` la liste des éléments de `L0` inférieurs ou égaux à `e` et `L2` la liste des éléments de `L0` strictement supérieurs à `e`. Soient `T1` et `T2` les listes obtenues lorsqu'on trie récursivement `L1` et `L2`. En utilisant `T1` et `T2`, on peut alors calculer une liste triée `T` contenant les mêmes éléments que `L`. L'arbre des appels récursifs lors du tri de la liste `[6,2,8,5,6,3,9,1]` est donné à la fin de l'énoncé.

1. Écrire une fonction `tri_rapide` qui trie une liste d'entiers à l'aide du tri rapide. Si besoin, on pourra utiliser les indications ci-dessous.
2. Quelle est la complexité de votre fonction dans le pire cas ?
3. Est-ce un tri en place ? Un tri stable ?