# Un premier algorithme de tri

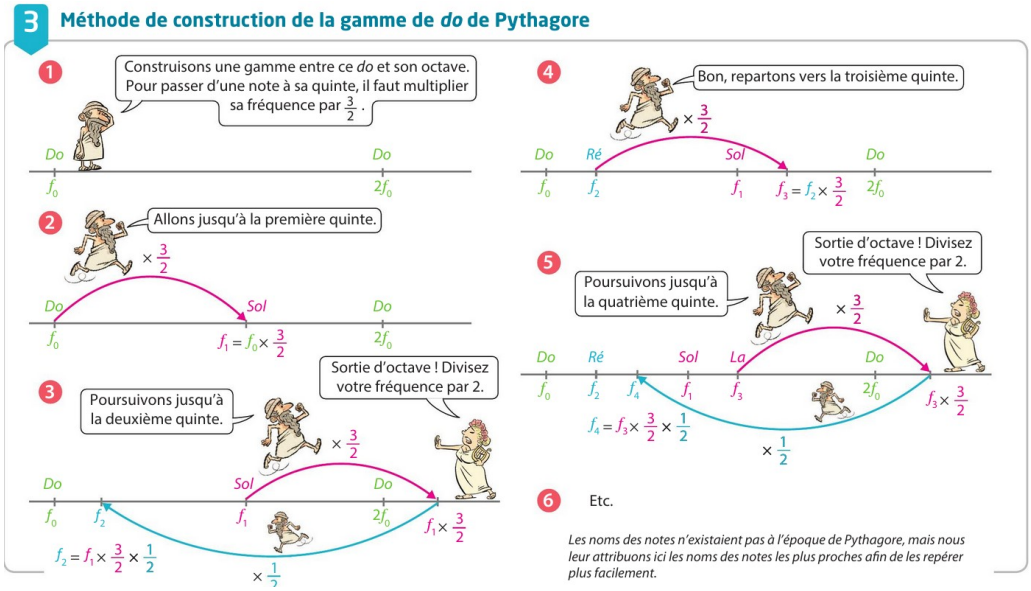
Dans ce TP, nous allons implémenter un algorithme de tri et le tester sur la gamme de Pythagore (découverte en enseignement scientifique) afin de trier les fréquences de cette gamme et de la comparer à la gamme tempérée.

Cet algorithme de tri nous permettra d'aborder des questions essentielles en algorithmique : Comment prouver qu'un algorithme se termine ? Comment estimer sa durée ?

## . Construction de la gamme de Pythagore

Une gamme est un ensemble de notes réparties sur une octave. La plus grande fréquence d'une gamme étant le double de la plus petite.

Les pythagoriciens construisent une gamme dont les notes sont séparées d'une quinte. La quinte est définie par le rapport de fréquence 3/2. La gamme de Pythagore se construit par quintes successives avec retour dans l'octave si la nouvelle fréquence est hors de l'octave d'étude.




*Extrait de la page 198 du manuel "Enseignement scientifique 1er", édition 2019, Hatier*

### . Exercice 1

1. Écrire une fonction `gamme_Pythagore` qui reçoit en entrée une fréquence `f` et renvoie la liste des fréquences des 12 notes (fréquence de départ comprise) de la gamme de Pythagore. Les fréquences seront arrondies au centième près en utilisant la fonction `round`.
2. Afficher la liste renvoyée par la fonction `gamme_Pythagore` pour la gamme de Do (f=261,6 Hz) à 12 notes. Quel est l'ordre des fréquences dans cette liste ?
3. **Sauvegarder ce travail dans un fichier que vous appellerez `gamme.py`**

## . UN PREMIER ALGORITHME DE TRI

Les fonctions de tri permettent de classer les donnée d'un tableau dans un ordre défini (par exemple ordre croissant, décroissant). Certains langages ont de fonctions prédéfinies qui effectuent ces opérations de tri mais il est nécessaire de connaître les principes des algorithmes de tris pour être paré en cas de besoin de tri inconnu de notre langage.

**L'exercice 2 est à faire dans un fichier que vous appellerez `tri_insertion.py` et qui nous servira dans d'autres TP.**

### Tri par insertion

C'est le tri utilisé par les joueurs de cartes pour ranger leur jeu.
Regarder la vidéo suivante pour une première approche du tri par insertion.

On peut écrire l'algorithme de la manière suivante, on remarque qu'il y a 2 boucles (bleue et rouge) imbriquées.

```
1  On dispose d'un tableau de nombres de longueur L
2  Pour i allant de 1 à (L)
3          stocker tab[i]
4          j=i
5          tant que (j>0) et tab[j-1]>valeur stockée
6                  tab[j]=tab[j-1]
7                  décrémenter j
8          remplacer tab[j] par la valeur stockée
```

Si vous souhaitez voir l'évolution des indices des boucles de l'algorithme ci-dessus, ce lien vous amène sur une page où vous pouvez faire avancer l'algorithme pas à pas. (Attendez peut-être quelques secondes pour pouvoir observer).

- Dans la partie gauche, utilisez le bouton *next* pour faire avancer votre algorithme pas à pas.
- Dans la partie droite, suivez le résultat de l'algorithme.

Soyez particulièrement attentifs à l'évolution du tableau à trier (en jaune) et à l'évolution des valeurs des itérateurs (i et j) et de la valeur stockée (élément) dans la partie bleue.

Encore une autre façon de voir ce tri avec des danses roumaines

### . EXERCICE 2

1. Appliquer manuellement le tri par insertion au tableau [8,6,1,9,7,2] en notant sur votre cahier l'état du tableau à la fin de chaque passage par la boucle bleue.
2. Écrire la fonction de `tri_insertion` dans un fichier `tris.py`
3. Compléter le code suivant pour l'appliquer au tableau précédent.

```
# Question 3 : Ne s'executera que si ce module est le programme principal
# (donc pas s'il est importé dans un autre)
if __name__ == '__main__':
    # A COMPLETER
```

### . EXERCICE 3

Nous allons nous pencher sur l'aspect théorique de cet algorithme de tri

1. Quel est la complexité de l'algorithme de tri par insertion

**Lire la partie du cours sur la terminaison.**

2. Quel est le variant de la boucle rouge de cet algorithme ? Comment varie-t-il ?
3. Quelle est la condition d'arrêt de la boucle rouge ?
4. Expliquer pourquoi on est sûr de la terminaison de cette boucle.
5. Reprendre les questions 2,3 et 4 pour la boucle bleue
6. Conclure sur la terminaison générale de l'algorithme de ce tri.

# . RETOUR SUR NOS GAMMES

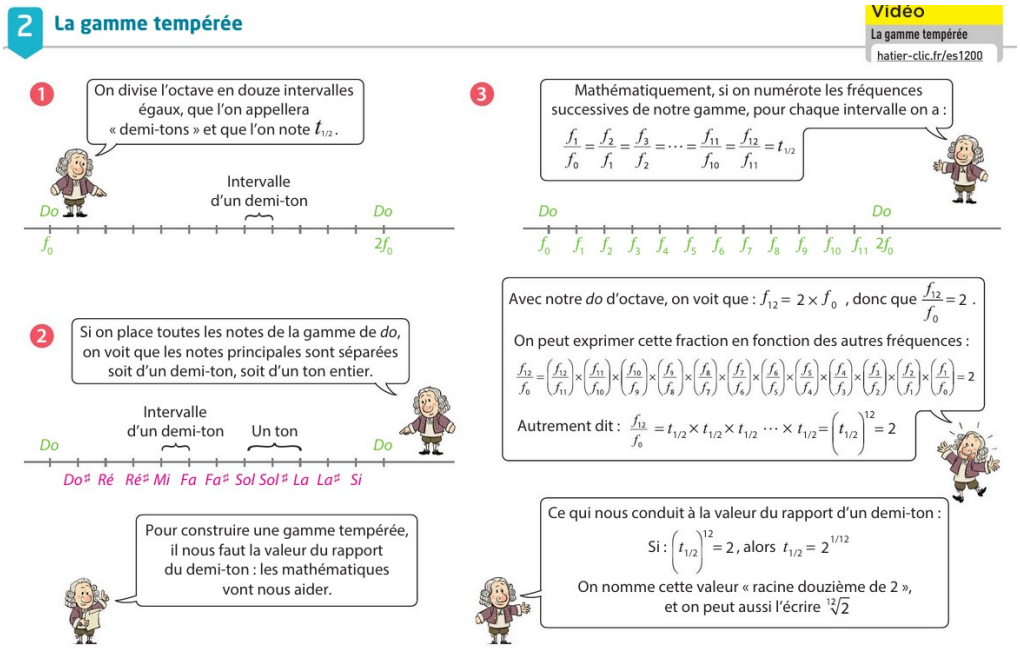
## . EXERCICE 4

Nous allons maintenant pouvoir trier notre gamme.

1. Dans le fichier `gamme.py`, importer le module `tri_insertion.py` (qui doit contenir les exercices 2 et 3)
2. Appliquer une fonction de tri de votre choix au résultat votre fonction `gamme_Pythagore` pour obtenir les fréquences des notes dans l'ordre croissant.

## . EXERCICE 5 (POUR LES PLUS RAPIDES)

**Nous allons compléter le fichier `gamme.py` avec la gamme tempérée de J.S. Bach.**




*Extrait de la page 200 du manuel "Enseignement scientifique 1er", édition 2019, Hatier*

1. Écrire une fonction `gamme_Bach` qui reçoit en entrée une fréquence `f` et renvoie la liste des fréquences des 12 notes (fréquence de départ comprise) de la gamme tempérée.
2. Afficher la liste renvoyée par la fonction `gamme_Bach` pour la gamme de Do (f=261,6 Hz) à 12 notes. Quel est l'ordre des fréquences dans cette liste ?

   **Lorsque l'on écoute les notes des deux gammes de Do (f=261,6 Hz) créées, on entend des différences entre les notes de même rang. On peut considérer 2 notes identiques si la différence entre leurs fréquence est inférieure à 0,5 Hz.**

3. Écrire une fonction `comparaison_gammes` qui reçoit une fréquence `f` de départ de gamme et renvoie les noms des notes communes entre les gammes de Pythagore et de Bach.
   Vous aurez besoin de créer la liste [Do,Do#,Ré,Ré#,Mi,Fa,Fa#,Sol,Sol#,La,la#,Si] pour avoir les noms
4. Quelles sont les notes communes des gammes de Do (261,6 Hz) de Pythagore et Bach ?