I) Introduction

Dans un algorithme par dichotomie, du grec « couper en deux », on coupe à chaque étape l'espace en deux parties. C'est un cas simple des techniques appelées « diviser pour régner ».
Les fonctions programmées dans les parties II) et III) sont à connaître et à savoir refaire.

II) Fonctions

Implémenter l'algorithme vu en cours, à l'aide d'une fonction `Dichotomie`.
Nous allons la tester sur des fonctions (du type $x \mapsto x^2 - 2$). La syntaxe à base de `def` est un peu trop lourde pour d'aussi petites fonctions. On utilise le mot clé `lambda`. La fonction $x \mapsto x^2 - 2$ s'écrirait

```
lambda x : x**2-2
```

Si votre fonction `Dichotomie` prend pour arguments une fonction f et trois réels a, b et ε, on écrira `Dichotomie(lambda x : x**2-2,a,b,epsilon)`.

1) $f(x) = x^2 - 2$. Résoudre $f(x) = 0$ entre 0 et 2 à 10^{-10} près.

2) Déterminer une valeur approchée de $\sqrt[3]{5}$ à 10^{-5} près.

3) Pour $n \in [\![0, N]\!]$, résoudre sur $[0, 1]$ l'équation $f_n(x) = 0$, où $f_n(x) = x^5 + nx - 1$. Le nombre N sera rentré par l'utilisateur. Conjecturer la limite de (u_n) et de (nu_n), où u_n est la solution de $f_n(x) = 0$.

III) Listes triées

Cette partie ressemble beaucoup au devoir à la maison.

1) Créer une fonction qui cherche un élément dans une liste triée (de nombres). La fonction retourne l'indice de l'élément dans la liste, ou -1, selon que l'élément est présent ou non dans la liste.

2) Tester votre fonction sur les listes suivantes :

- **a)** `L1=list(range(10**8))`
- **b)** `L2=list(range(42,10**8,58))`
- **c)** `L3=[i**2 for i in range(10**5)]`

3) Déterminer la complexité de cet algorithme.
Bonus : La liste `L3` permet de tester si un nombre est un carré. En supposant cette liste déjà construite jusqu'à N assez grand, déterminer la complexité de l'algorithme qui permet de déterminer si n est un carré. Le comparer à un autre algorithme de votre choix.

Pour estimer finement l'efficacité d'un programme, si vous avez le temps, vous pouvez regarder la page `http://docs.python.org/2/library/profile.html`.

IV) Récursivité (bonus : 2e année)

On dit qu'une fonction est récursive si elle s'appelle elle-même. Implémenter l'algorithme précédent à l'aide d'une fonction récursive.

V) Tri rapide : *quicksort* (idem)

Implémenter le tri rapide (1961) : le but est de trier une liste de nombres.
Esquisse de l'algorithme :

- On choisit un élément `p` (par exemple le premier de la liste) appelé pivot.
- On parcourt la liste restante en rangeant dans une liste `Lmin` tous les éléments plus petits que `p`, et dans une autre liste `Lmax` tous les éléments plus grands que `p`.
- On applique la fonction aux listes `Lmin` et `Lmax`

Implémenter proprement cet algorithme.
Pour une jolie animation, voir `http://en.wikipedia.org/wiki/Quicksort`