TP 10 : récursivité

PTSI 2025/2026

Script 1 : *calcul des termes d'une suite récurrente*

Niveau : 1

On considère une suite récurrente définie par son premier terme u_0 puis pour tout entier naturel n par :

$$u_{n+1} = f(u_n)$$

où f est une fonction définie sur $\mathbb{R}$ (pour simplifier).

1. Écrire une fonction python `suite_iter` itérative qui prend en argument une fonction `f`, un flottant `u0` et un entier `n` et renvoie une valeur approchée du terme d'indice `n` de la suite u définie comme ci-dessus.
2. Écrire une fonction `suite_recur(f,u0,n)` effectuant la même tâche.
3. Effectuer à la main la suite d'appels de votre fonction dans le cas $f : x \mapsto 2x + 1$, $u_0 = 3$ et $n = 2$ en la représentant sous forme arborescente puis faire la remontée de cette pile d'appel pour calculer le résultat.
4. Faire un comparatif de temps d'exécution `n` valant 100.

Script 2 : *fonction mystère*

Niveau : 2

SANS LA FAIRE exécuter par Python, dire ce que fait la fonction suivante.

```
def dk(L1,L2=[]):
    if L1 == []:
        return L2
    else:
        s = L1.pop(0)
        if s not in L2:
            L2.append(s)
        return dk(L1,L2)
```

Script 3 : *dichotomie*

Niveau : 2

On rappelle l'algorithme de dichotomie pour la résolution approchée d'une équation $f(x) = 0$ où :

- f est continue sur l'intervalle $[a, b]$
- $f(a)$ et $f(b)$ sont de signe différents.

Algorithme de dichotomie :

On construit par récurrence deux suites de valeurs $(a_n)_n$, $(b_n)_n$ qui encadre une solution α de l'équation $f(x) = 0$

- $a_0 = a, b_0 = b$.
- Pour $n \in \mathbb{N}$:
  - On considère $c_n = \dfrac{a_n + b_n}{2}$
  - si $f(c_n)$ et $f(a_n)$ sont de signes différents :

$$a_{n+1} = a_n \quad , \quad b_{n+1} = c_n$$

  - sinon

$$a_{n+1} = c_n \quad , \quad b_{n+1} = b_n$$

1. Quel théorème mathématique permet d'affirmer qu'il y a toujours une solution à l'équation $f(x) = 0$ entre a_n et b_n ?

2. `f` une fonction Python prenant en entrée un flottant x et renvoyant un flottant qu'on suppose définie préalablement. (voir squelette).

   Programmer une fonction itérative, `dicho_iter` prenant en argument une fonction python `f`, deux flottants `a,b`, et une précision `p` et qui renvoie une valeurs approchée d'une solution de `f(x)=0` en utilisant l'algorithme de dichotomie.

3. Programmer une version récursive de cette fonction : `dicho_recur`.
4. Effectuer un test de temps pour comparer les deux fonctions.

Script 4 : *tri rapide* — Niveau : 3

L'algorithme de tri rapide pour une liste `L` de nombre est un algorithme dans lequel on procède comme ci-dessous pour créer une version triée par ordre croissant de la liste `L` :

- on choisit au hasard un indice : `ind` (dans les indices possibles pour la liste `L`)
- On créé deux listes `L1` et `L2` de la manière suivante :
  - `L1` contient toutes les valeurs de `L` qui sont inférieures ou égales à `L[ind]`
  - `L2` contient toutes les valeurs de `L` qui sont strictement supérieures à `L[ind]`
- On relance l'algorithme sur les deux listes et on concatène le résultat.

1. Écrire une fonction `partition` qui prend en entrée une liste `L` et un indice `ind` et renvoie les deux listes `L1` et `L2` décrites ci-dessus.
2. En déduire une fonction `tri_rapide` d'argument une liste de nombres `L` et renvoyant une version triée de `L` par l'algorithme précédent.
3. Faire des comparaison de temps d'exécution avec :
   a. avec une fonction de tri utilisant l'algorithme de tri par insertion ou par sélection.
   b. avec la fonction de tri `sort` native en Python.

Script 5 : *exponentiation rapide* — Niveau : 2

Pour calculer a^n lorsque a est un nombre réel et n un entier naturel non nul, on peut adopter la méthode suivante :

- Si n est pair on calcul $(a^{n/2})^2$
- sinon on calcule $a \times a^{n-1}$

1. Écrire une fonction python récursive qui utilise l'idée précédente pour effectuer le calcul de a^n (sans utiliser bien sûr la l'opération $\star\star$ de Python).
2. Introduire dans cette fonction un compteur permettant de déterminer le nombre d'appels fait par la fonction à elle même.