L3 - Méthodes numériques
TP 1 - Représentation numérique et rappels de Python

Support de TP

Les TPs visent à appliquer de manière concrète les notions vues en cours. Il est donc conseillé de venir avec vos supports de cours ou de les télécharger sur le web.

Les TPs notés doivent être soumis sur la page Moodle du cours dans les deux semaines suivant le TP. Vous devez soumettre un fichier zip contenant le code **sans bug** avec un scripte python **TP1_1.py,TP1_2.py,..** par partie du TP ainsi qu'un court rapport de 2 pages maximum en **PDF** (sans page de titre). Le rapport ne doit contenir aucune sortie terminal ou ligne de code. Vous devez par contre pour chaque section du TP rédiger une réflexion sur les notions utilisées et ce que vous avez appris.

1 Taille mémoire et limites des différents types

cette section vise à étudier la taille mémoire et les limites des types numériques suivants : `int, float, np.int8, np.int16, np.int32, np.int64, np.float32, np.float64` . Pour utiliser ces différents types au maximum de leur potentiel, il est nécessaire de connaître leur taille mémoire ainsi que leurs limites numériques. On rappelle qu'en Python, on définit le type d'une variable lors de l'initialisation, il est possible que ce type évolue lors des calculs.

1. Initialiser une variable de type `int`. Quelle est sa taille mémoire en bit retournée par la méthode `bit_length()` ?
2. Coder un programme qui imprime à l'écran la taille mémoire en bits de chacun des types numpy suivants : `np.int8, np.int16, np.int32, np.int64, np.float32, p.float64` . Vous pouvez pour celà faire appel à la propriété `nbytes`.
3. Pour chaque type numpy ci dessus imprimer les valeurs limites et informations. il faudra pour cela utiliser la fonction `np.iinfo` for les entiers et `np.finfo` pour les flottants.

2 Fonction factorielle

1. Coder une fonction `factorielle` qui calcule la factorielle d'un nombre de manière récursive. Le type de la valeur retournée devra être celui de la variable donnée en entrée (en utilisant le type retourné par la fonction `type` ex : `type(n)(1)` retourne un 1 du même type que n). Faire attention à bien retourner la valeur 1 pour une entrée 0 de la fonction (condition d'arrêt).
2. Coder dans le programme une boucle permettant d'afficher à l'écran la valeur de la factorielle pour une entrée allant de 0 à 22.
3. Faire varier le type de l'entier donné à la fonction `factorielle` (`int`,`np.int\{8,16,32,64\}`) et déterminer jusqu'à quelle valeur d'entrée le résultat est exact pour chaque type (valeurs exactes sur wikipedia). Commenter.
4. Quelle est la complexité de la fonction `factorielle` ? Quelle est la complexité totale du programme ? Serait-il possible de diminuer la complexité asymptotique (du programme entier avec la boucle) ? Si oui, comment ?

3 Série entière

Soit la série entière suivante :

$$\sum_{i=1}^{n} \frac{1}{i}$$

Cette série peut être implémentée à l'aide d'une boucle `for` qui commence soit à `i=1`, soit à `i=n`.

1. Coder une première fonction `somme1` qui utilise en interne le type `np.float32` et qui commence à `i=1`. Imprimer son résultat pour `n=100000`.
2. Coder une deuxième fonction `sommen` qui retourne un `np.float32` et qui commence à `i=n` Imprimer son résultat pour `n=100000`.
3. Calculer la différence entre les deux résultats et l'imprimer à l'écran.
4. Coder une fonction `somme` plus précise qui utilise le type `float` pour déterminer une valeur plus exacte (64bits).
5. D'où vient la différence entre `somme1` et `sommen` ? Quelle implémentation est la plus précise ?

4 Plus grand commun diviseur (PGCD)

Dans cette section, on vous demande de coder une fonction retournant le PGCD de deux nombres entier.

1. Chercher sur le web des informations concernant l'algorithme d'Euclide.
2. Implémenter cet algorithme dans une fonction `pgcd`.
3. Afficher à l'écran des résultats pour différents entiers `a` et `b` et vérifier le fonctionnement correct de votre fonction.