Récursivité

Informatique #02

Partie A – Pour commencer...

Exercice 1 — factorielle

1. Écrire une fonction récursive `fact(n)` qui renvoie la factorielle d'un entier `n` donné.
2. Écrire une fonction `fact2(n)` similaire mais implémentée à l'aide d'une boucle.
3. Construire un tableau aléatoire de 100 entiers compris entre 1 et 5000 puis calculer les factorielles correspondantes à l'aide des fonctions précédentes. Comparer les temps de calcul.

On peut utiliser la commande `setrecursionlimit` disponible dans la bibliothèque `sys` pour modifier la longueur de la pile d'appels récursifs.

Exercice 2 — puissance

1. Écrire une fonction récursive `puis(x, n)` qui retourne x^n pour x réel et n entier naturel.
2. Écrire une fonction analogue itérative.

Exercice 3 — somme des entiers

Écrire une fonction récursive `somme_ent(n)` qui calcule la somme des entiers compris entre 0 et n.

Exercice 4 — triangle de Pascal

1. Écrire une fonction `binom(n, k)` qui calcule $\binom{n}{k}$ à l'aide de la formule :

$$\forall (k,n) \in \mathbb{N}^* \times \mathbb{N}^* \quad \binom{n-1}{k-1} + \binom{n-1}{k} = \binom{n}{k}$$

2. En déduire une fonction `trianglepascal(n)` qui affiche les $n+1$ premières lignes du triangle de Pascal.
3. Essayer d'optimiser la fonction précédente.

Exercice 5 — somme des éléments d'une liste

Écrire une fonction `sommelist(L)` qui calcule récursivement la somme des éléments d'une liste `L`.

Partie B – Autour de la suite de Fibonacci

Exercice 6 — Fibonacci récursive v1

On rappelle que la suite de Fibonacci vérifie la relation de récurrence suivante :

$$F_0 = 0, \quad F_1 = 1, \quad \forall n \in \mathbb{N} \quad F_{n+2} = F_{n+1} + F_n$$

1. Écrire une fonction récursive `fibo_rec1(n)` qui retourne la valeur de F_n.
2. Quelle est sa complexité ?

Exercice 7 — Fibonacci itérative

1. Écrire une fonction itérative `fibo_it(n)` qui retourne la valeur de F_n.
2. Quelle est sa complexité ?

Exercice 8 — Fibonacci récursive v2 (récursivité terminale)

1. Écrire une fonction récursive `fibo_tempo(a, b, k)` qui prend en argument F_n et F_{n-1} puis qui retourne F_{n+k}.
2. En déduire une nouvelle fonction récursive `fibo_rec2(n)` qui renvoie F_n.
3. Quelle est sa complexité ?

Exercice 9 — Fibonnaci récursive v3 (mémoïsation)

Écrire une fonction récursive `fibo_rec(n)` qui renvoie F_n en sauvagardant dans une liste ou bien un dictionnaire les termes de la suite déjà calculés.

Exercice 10 — comparaison des différentes méthodes (☕)

1. Comparer les temps de calcul dans les 4 cas. On pourra prendre $n = 34$.
2. Comparer graphiquement les temps de calcul des 4 méthodes.

Partie C – Arithmétique

Exercice 11 — algorithme d'Euclide
Écrire une fonction `euclide(a, b)` permettant de calculer le pgcd de deux entiers de façon récursive.

Exercice 12 — conversion d'entiers (☕)

1. Écrire une fonction `listechiffres(n)` d'un entier `n` qui retourne la liste des chiffres de cet entier (en base 10) sous forme de liste.
2. Écrire deux fonctions `codbinaire_it(n)` et `codbinaire_rec(n)` qui retournent une chaîne de caractères représentant l'entier `n` converti en base 2 en procédant soit de façon itérative, soit de façon récursive.
3. Écrire enfin une fonction `codage(n, base)` permettant de convertir tout entier `n` dans une base donnée, en supposant que la base est inférieure ou égale à 16.

Exercice 13 — exponentiation rapide

1. Écrire une fonction permettant, à l'aide de l'algorithme d'exponentiation rapide vu en cours, de calculer x^n pour un réel x et un entier n donnés, en n'utilisant que des multiplications.
2. Comparer le temps d'exécution avec l'algorithme naïf.

Partie D – Suites récurrentes

Exercice 14 — conjecture de Syracuse
On considère la suite de Syracuse définie par $u_0 = c$ et :

$$\forall n \in \mathbb{N} \quad u_{n+1} = \begin{cases} \dfrac{u_n}{2} \text{ si } u_n \text{ est pair} \\ 3u_n + 1 \text{ sinon} \end{cases}$$

On conjecture, mais ce résultat n'a jamais été démontré, que quel que soit c, la suite $(u_n)_{n\in\mathbb{N}}$ atteint la valeur 1, c'est-à-dire qu'il existe $N \in \mathbb{N}$ tel que $u_N = 1$.

1. Écrire une fonction récursive `syracuse(c)` qui retourne les différentes valeurs prises par la suite jusqu'à qu'elle atteigne 1.
2. Écrire une fonction analagogue mais cette fois-ci de manière itérative.

Exercice 15 — approximation de $\sqrt{2}$

– *Méthode de Héron*
Trouver une valeur approchée de $\sqrt{2}$ en considérant la suite définie par $u_0 = 5$ et $u_{n+1} = \dfrac{u_n}{2} + \dfrac{1}{u_n}$ pour tout $n \geqslant 1$.

– *Méthode Théon de Smyrne*
Trouver de nouveau une valeur approchée de $\sqrt{2}$ à l'aide du rapport $\dfrac{p_n}{q_n}$ en considérant les suite définies par $p_0 = q_0 = 1$ et pour tout $n \geqslant 0$,

$$p_{n+1} = p_n + 2q_n \quad \text{et} \quad q_{n+1} = p_n + q_n$$

Exercice 16 — suites récurrentes et graphes en escalier (☕)
Écrire une fonction `escargot(f, n, u0, xmin, xmax)` permettant de visualiser le graphe en colimaçons engendré par une suite récurrente de la forme $u_{n+1} = f(u_n)$, les arguments `xmin` et `xmax` permettant de régler la fenêtre d'affichage.

Partie E – Miscellaneous

Exercice 17 — palindrome

1. Écrire une fonction récursive `inverse(mot)` qui renverse la chaîne de caractères `mot`.
2. En déduire une fonction `palindrome(mot)` qui détecte la présence d'un palindrome.

Exercice 18 — liste d'anagrammes (☕)

1. Écrire une fonction récursive `anagramme(mot)` qui retourne la liste des anagrammes d'un mot donné.
2. Améliorer la fonction précédente pour éliminer les éventuels doublons.