TP 3

Fonctions récursives / fonctions itératives

3 octobre 2018

Créez un fichier `VotreNom-TP3.py` dans votre dossier `infoPSI`.
Pour chacun des exercices, faites deux fonctions : une fonction récursive et une fonction itérative.
Testez vos fonctions au fur et à mesure pour les vérifier. Lorsque vous testez une fonction, mettre les appels à la fonction en commentaire avant de passer à l'exercice suivant (ne pas les effacer).

Exercice 1 : Somme

Écrivez une fonction `somme` qui prend en argument un entier positif n et qui renvoie la somme des n premiers entiers. Testez vos fonctions (par exemple $1 + 2 + 3 + 4 + 5 + 6 + 7 = 28$).

Exercice 2 : Factorielle

Écrivez une fonction `factorielle` qui prend en argument un entier positif n et qui renvoie le produit des n premiers entiers. Testez vos fonctions (par exemple $7!$ vaut 5040).

Exercice 3 : Carrés

Écrivez une fonction `sommeCarres` qui prend en argument un entier positif n et qui renvoie $1^2 + 2^2 + \cdots + n^2$. Testez vos fonctions (par exemple $1^2 + 2^2 + \cdots + 4^2 = 30$).

Exercice 4 : Compter

Écrivez une fonction `compter` qui prend en argument un entier n et affiche les entiers compris entre 1 et n. Par exemple, l'exécution de compter(4) doit donner :

```
1
2
3
4
```

Exercice 5 : Compte à rebours

Écrivez une fonction `rebours` qui prend en argument un entier n et affiche en ordre décroissant les entiers compris entre n et 0. Par exemple, l'exécution de rebours(4) doit donner :

```
4
3
2
1
0
```

Exercice 6 : Puissance

Écrivez une fonction `puissance` qui prend en argument un réel x et entier positif n et qui renvoie x^n. Ne pas utiliser **. Testez vos fonctions (par exemple $(-2.5)^3 = -15.625$).
Adaptez vos fonctions (version récursive et itérative de puissance) pour prendre en compte le cas où n est négatif (par exemple $(2.5)^{-3} = 0.064$).

Exercice 7 : Suite

Écrivez une fonction `terme` qui prend en argument un entier n et qui renvoie le n-ième terme de la suite définie par $u_n = \frac{1}{2}(u_{n-1} + \frac{3}{u_{n-1}})$ et $u_0 = 2$ (par exemple $u_5 = 1.7320508075688772$).

Exercice 8 : Fibonacci

Écrivez une fonction `fibonacci` qui prend en argument un entier n et qui renvoie le n-ième terme de la suite définie par $F_n = F_{n-1} + F_{n-2}$, $F_0 = 0$ et $F_1 = 1$ (par exemple $F_7 = 13$).

Exercice 9 : Double factorielle

Écrivez une fonction `doubleFactorielle` qui prend en argument un entier n et qui renvoie $n!!$, sachant que $n!! = n \times (n-2) \times (n-4) \cdots \times 1$ si n est impair, et $n!! = n \times (n-2) \times (n-4) \cdots \times 2$ si n est pair (par exemple $5!! = 15$ et $6!! = 48$).

Exercice 10 : Multifactorielle

Écrivez une fonction `multiFactorielle` qui prend en argument deux entiers n et k et qui renvoie $n!_k$ la factorielle de n d'ordre k, sachant que $n!_k = n \times (n-k) \times (n-2k) \cdots \times$ (par exemple $6!_3 = 18$ et $8!_3 = 80$).

Exercice 11 : Partie entière

Écrivez une fonction `partieEntiere` qui prend en argument un réel x et qui renvoie la partie entière de x (par exemple la partie entière de -6.4 est -7). Ne pas utiliser la fonction `int()`, faire comme si vous êtes le programmeur qui code cette fonction (même si en réalité elle est codée de façon bien plus efficace).

Exercice 0 : Complexité

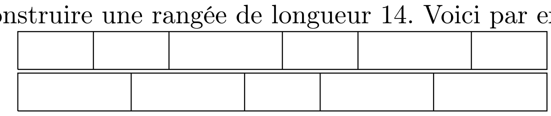
Estimez la complexité de chacune de vos fonctions jusqu'à l'exercice 11 inclus.

Exercice 12 : PGCD

Écrivez une fonction `pgcd` qui prend en argument deux entiers a et b et qui renvoie le pgcd de a et de b en utilisant l'algorithme d'Euclide (par exemple le pgcd de 24 et de 42 est 6).

Exercice 13* : Briques

Écrivez une fonction `briques` qui prend en argument un entier n et qui renvoie le nombre de façons de construire une rangée de longueur n avec des briques de longueur 2 et 3. Par exemple il y a 21 façons de construire une rangée de longueur 14. Voici par exemple deux rangées de longueur 14 :



Exercice 14* : Ackermann

La fonction d'Ackermann est définie par :
$Ack(0, n) := n + 1$
$Ack(m, 0) := Ack(m-1, 1) \quad \forall m > 0$
$Ack(m, n) := Ack(m-1, Ack(m, n-1)) \quad \forall m, n > 0$
Écrivez une fonction ack, sauvegardez votre travail puis testez la fonction ack sur de PETITS entiers.
Bonus :

- prouver par récurrence que $ack(1, n) = 2 + n$
- prouver par récurrence que $ack(2, n) = 2n + 3$
- prouver par récurrence que $ack(3, n) = 2^{n+3} - 3$
- donner un argument montrant que le calcul de ack(m,n) ne provoque pas de boucle infinie.