

TD 2 - Fonctions, modules, exceptions

Ex1- Suite de Fibonacci

Fonctions simples

1. Ecrire une fonction qui affiche la suite de Fibonacci jusqu'à une limite passée en paramètre.
2. Ecrire une fonction qui retourne la suite de Fibonacci jusqu'à une limite passée en paramètre.
3. Ecrire le programme principal qui appelle ces 2 fonctions apres avoir demandé à l'utilisateur la valeur limite
4. Ajouter une exception qui permet de traiter le cas où l'utilisateur saisirait une valeur non entière ou un entier négatif.

Ex2- Couicable

Fonctions simples, retour d'une ou plusieurs variables

1. Ecrire la fonction `sommechiffre()` qui retourne la somme des chiffres composant le nombre passé en paramètre.
2. Ecrire la fonction `nombreChiffres()` qui retourne le nombre des chiffres composant le nombre passé en paramètre.
3. Ecrire la fonction `separeNombre()` separe le nombre passé en paramètre en 2 : partie droite et gauche si il est composé d'un nombre de chiffres pair. Sinon il retourne `0,0`
4. Ecrire le programme principal qui demande un nombre à l'utilisateur et affiche si celui si est couicalbe ou non. Un nombre est dit couicable s' 'il est composé d'un nombre de chiffres pair et si la sommes des chiffres de sa partie droite est égale à la somme des chiffres de sa partie gauche.
5. Re-écrire la fonction `sommeChiffre()` mais cette fois-ci de facon récursive.

Exo 3- Distances 2D

Fonctions simples, utilisation du module math, arguments par défaut

1. Ecrire une fonction `calc_distance_2D()` qui calcule la distance euclidienne entre deux points 2D définis par leurs coordonnées x,y. Vous pouvez utiliser la fonction `sqrt()` définie dans le module `math`. Au cas où la fonction est appelée avec un seul point ce sera la distance du point à l'origine.

Testez votre fonction sur les 2 points A(2,2) et B(1,1). Trouvez-vous bien racine de 2 ?

2. Ecrire une autre fonction `calc_dist2ori()` à laquelle vous passez en argument deux listes de floats `list_x` et `list_y` représentant les coordonnées d'une fonction mathématique (par exemple `x` et `sin(x)`). Cette fonction renverra une liste de floats représentant la distance entre chaque point de la fonction et l'origine (de coordonnées (0,0)).
3. Ecrire une fonction `genere_list_sin (inf, sup, pas)` qui retourne deux listes correspondant aux coordonnées x et y de la fonction sinus entre `inf` et `sup` avec un `pas` donné.
4. Ecrire le programme principal qui appelle la fonction `genere_list_sin()` et affiche la distance à l'origine des points calculés.

Exo4 - Carré magique

Fonctions simples, définition et utilisation d'un module, listes de listes

Nous voudrions définir un module `carre_magique` qui contient les fonctions relatives aux carrés magiques. Les fonctions demandées dans les questions 1 à 6 seront définies dans ce module. La question 7 sera définie dans un fichier contenant aussi la fonction `main()` qui utilisera ce module.

Défintion d'un carré magique

On considère un entier `n` strictement positif. Un carré magique d'ordre `n` est une matrice carrée d'ordre `n` (`n` lignes et `n` colonnes), qui contient des nombres entiers strictement positifs. Ces nombres sont disposés de sorte que les sommes sur chaque ligne, les sommes sur chaque colonne et les sommes sur chaque diagonale principale soient égales. La valeur de ces sommes est appelée : constante magique.

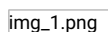
Exemple : Carré magique d'ordre 3, sa constante magique vaut 45



1. Écrire la fonction `somme_ligne(M,i)` , qui reçoit en paramètres une matrice carrée `M` contenant des nombres, et un entier `i` qui représente l'indice d'une ligne dans `M`. La fonction retourne la somme des nombres de la ligne d'indice `i` dans `M`.
2. Écrire la fonction `somme_colonne(M,j)` , qui reçoit en paramètres une matrice carrée `M` contenant des nombres, et un entier `j` qui représente l'indice d'une colonne dans `M`. La fonction retourne la somme des nombres de la colonne `j` dans `M`.
3. Écrire la fonction `somme_diag1(M)` , qui reçoit en paramètre une matrice carrée `M` contenant des nombres, et qui retourne la somme des éléments de la première diagonale principale dans `M`.
4. Écrire la fonction `somme_diag2(M)` , qui reçoit en paramètre une matrice carrée `M` contenant des nombres, et qui retourne la somme des éléments de la deuxième diagonale principale dans `M`.
5. Écrire la fonction `magique(C)` , qui reçoit en paramètre une matrice carrée `C` contenant des entiers strictement positifs, et qui retourne `True`, si la matrice `C` est un carré magique et `False`, sinon.

Un carré magique normal d'ordre `n` est un carré magique d'ordre `n`, constitué de tous les nombres entiers positifs compris entre 1 et `n^2` .

Exemple : Carré magique normal d'ordre 4, composé des nombres entiers : 1, 2, 3, ..., 15, 16.



6. Écrire la fonction ``carre_magique_normal(C)`, qui reçoit en paramètre une matrice carrée `C` qui représente un carré magique. La fonction retourne `True` si le carré magique `C` est normal, sinon, elle retourne `False`.

Exemples

La fonction `carre_magique_normal` ([[8, 1, 6] , [3, 5, 7] , [4, 9, 2]]) retourne `True`. La fonction `carre_magique_normal`([[21, 7, 17] , [11, 15, 19] , [13, 23, 9]]) retourne `False`

7. Dans une autre fichier, écrire un programme principal qui teste des carrés et affiche si ils sont magiques et le cas échéant si ils sont "normaux". vous pouvez tester les carrés suivants

```
C1= [[8, 1, 6], [3, 5, 7], [4, 9, 2]]
C2 = [[21,7,17],[11,15,19],[13,23,9]]
C3= [[30,39,48,1,10,19,28], [38,47,7,9,18,27,29],
      [46,6,8,17,26,35,37], [5,14,16,25,34,36,45],
      [13,15,24,33,42,44,4], [21,23,32,41,43,3,12],
      [22,31,40,49,2 ,11,20]]
```

Exo5 - Décorateur

Reprendre l'exercice précédent et ajouter un décorateur à la fonction `affiche_test_cm`. Le décorateur a pour role de vérifier que le module `carre_magique` a bien été importé. Dans ce cas , il exécute la fonction, sinon, il affiche un message d'erreur.