I11 - Programmation I : Python
TP 5

Semestre 1 2018

Fonctions

Les exercices 1 à 3 sont à faire dans le même fichier.

Exercice 1. **Les nombres parfaits**

Les diviseurs propres d'un nombre sont tous les diviseurs du nombre excepté lui-même. Par exemple les diviseurs propres de 8 sont $1, 2, 4$.

Un nombre est parfait s'il est égal à la somme de ses diviseurs propres, par exemple $6 = 1 + 2 + 3$.

1. Écrire une fonction `som_div_propres(n)` qui retourne la somme des diviseurs propres de l'entier n.
2. Écrire une fonction `est_parfait(n)` qui retourne le booléen `True` si le nombre n est parfait.
3. Écrire une procédure `affiche_parfait(k)` qui affiche tous les nombres parfaits plus petits que 2^k.

Exercice 2. **Les nombres presque-parfaits**

Un nombre n est presque-parfait si la somme de ses diviseurs propres est égale à $n - 1$.

1. Écrire une fonction `est_presque_parfait(n)` qui retourne le booléen `True` si le nombre n est presque-parfait.
2. Écrire une procédure `affiche_presque_parfait(k)` qui affiche tous les nombres presque-parfaits plus petits que 2^k.

Exercice 3. ($\star$)**Les nombres amicaux**

Deux nombres n et m sont dits amicaux s'ils sont égaux à la somme des diviseurs propres de l'autre. Par exemple, les nombres 220 et 284 sont amicaux, la somme des diviseurs propres de 220 est $1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284$ et la somme des diviseurs propres de 284 est $1 + 2 + 4 + 71 + 142 = 220$.

1. Écrire une fonction `amicaux(n,m)` qui retourne le booléen `True` si les nombres n et m sont amicaux.
2. Écrire une procédure `affiche_amicaux(k)` qui affiche tous les nombres amicaux plus petits que 2^k. Notons que pour un n donné, son ami potentiel est `som_div_propres(n)`. Pour lister les nombres amicaux, il faut vérifier que `n` et `som_div_propres(n)` sont amicaux pour chaque n compris entre 1 et 2^k.

Exercice 4. **Vecteur et produit scalaire**

Reprenez les définitions vues aux TP 4. Tous les vecteurs sont des vecteurs réels.

1. Définir une fonction `somme(u,v)` qui calcule et retourne la somme de deux vecteurs donnés en paramètre.
2. Définir une fonction `mult(a,u)` qui calcule et retourne le produit du vecteurs `u` par le réel `a`.
3. Écrire le script qui affiche le résultat de $a.\vec{u} + b.\vec{v}$ où a, b sont des réels donnés par l'utilisateur et $\vec{u} = (1, 2, 1, 3)$ et $\vec{v} = (4, 1, 0, 2)$.
4. Définir une fonction `produit_scalaire(u,v)` qui calcule et retourne le produit scalaire de deux vecteurs donnés en paramètres. Notons que pour faire le produit scalaire les vecteurs doivent avoir le même nombre de composante.
5. Définir la fonction `norme(u)` qui calcule et retourne la norme du vecteur u donné en paramètre.
6. Définir la fonction `cosinus(u,v)` qui calcule et retourne le cosinus de l'angle $(\vec{u}, \vec{v})$.
7. Définir une fonction `saisit_vecteur(n)` qui retourne le vecteur de n coordonnées données par l'utilisateur dans le corps de la fonction.
8. Écrire le script qui saisit deux vecteurs `x, y` de dimension 3, qui affiche les vecteurs, leur norme, leur produit scalaire et le cosinus de l'angle formé par ces deux vecteurs.

Exercice 5. **Calcul de l'exponentielle avec une série entière**

Cet exercice a été en partie vue en TD. On pourra utliser la fonction `exp(x)` du module `math` qui retourne e^x.

1. Définir la fonction `factorielle(n)` qui retourne $n! = n \times (n-1) \times (n-2) \times \cdots \times 1$.
2. Définir la fonction `puissance(x,n)` qui retourne x^n où x est un réel et n un entier.
3. Définir une fonction `serie_exp(x,n)` qui retourne le flottant

$$\sum_{i=0}^{n} \frac{x^i}{i!} = \frac{x^0}{0!} + \frac{x^1}{1!} + \cdots + \frac{x^n}{n!}$$

4. Écrire un script qui demande un flottant `x` à l'utilisateur puis une précision entière `p` et affiche la plus petite valeur `n` telle que

$$\left| \sum_{i=0}^{n} \frac{x^i}{i!} - e^x \right| < 10^{-p}.$$