Nombres parfaits, nombres amicaux

Un nombre est considéré comme parfait lorsqu'il est égal à la somme de ses diviseurs stricts : par exemple, le nombre 6 est un nombre parfait car ses diviseurs stricts sont 1, 2 et 3.
On aimerait créer un programme qui détermine automatiquement si un nombre est parfait ou non. Mais pour commencer, nous allons réaliser une fonction `somme_diviseurs(n)` qui réalise automatiquement la somme des diviseurs stricts d'un entier n.

Calculer la somme des diviseurs d'un nombre

Niveau 1 : sans liste

Le programme est très semblable à celui de l'activité précédente. En effet, on propose de passer en revue les entiers naturels et de calculer progressivement la somme des diviseurs du nombre n en créant une variable `somme` : chaque fois qu'un nombre est reconnu comme un diviseur, on ajoute sa valeur à cette variable.

1. Au début de la fonction, quelle est la valeur de la variable `somme` ?

   A l'initialisation, `somme=0`.

2. Quelle instruction va nous permettre de passer en revue les entiers naturels afin de les tester comme diviseurs de n ? Quelle ligne de code doit-on utiliser ?

   On utilise une boucle pour tester tous les entiers naturels de 1 à n exclu car, attention, on cherche à lister les diviseurs stricts. On écrira donc `for i in range(1,n):`.

3. Quel test allons-nous réaliser afin de vérifier si un nombre est diviseur de n ? Quelle ligne de code doit-on utiliser ?

   C'est un diviseur si le reste est nul lorsque l'on réalise la division euclidienne de n par ce nombre. Pour cela, on utilisera le symbole % contenu dans le module `math` et le test sera : `if n%i==0:`

4. A l'aide des questions précédentes (et si nécessaire, de l'activité précédente), écrire la fonction `somme_diviseurs(n)`. On pourra utiliser la touche VAR pour retrouver les variables déjà définies dans le script et écrire plus rapidement le programme.

   La fonction reprend le même modèle que celle abordée dans l'activité "Lister les diviseurs d'un nombre".

```
from math import *
def somme_diviseurs(n):
  somme=0
```

```
    for i in range(1,n):
      if n%==0:
        somme=somme+i
    return somme
```

On pourra aussi utiliser `somme+=i` pour incrémenter la variable.

Niveau 2 : avec une liste

La fonction `sum(liste)` permet de calculer automatiquement la somme de tous les éléments d'une liste.
En reprenant l'algorithme de la fonction `diviseurs(n)` de l'activité précédente, c'est-à-dire en créant une liste vide dans laquelle on va progressivement stocker les diviseurs de n, écrire une fonction `sum_div(n)` qui utilise l'instruction `sum`.

```
def sum_div(n):
  div=[]
  for i in range(1,n):
    if n%i==0:
      div.append(i)
  return sum(div)
```

Contrairement à l'activité précédente, on souhaite ici tous les diviseurs stricts, c'est pourquoi la boucle s'arrête en n exclu.

Niveau expert : avec une liste en compréhension

On a vu qu'il était possible de générer automatiquement une liste en une seule ligne. Essayez de créer une fonction `sum_div_short(n)` qui ne doit pas dépasser deux lignes !

```
def sum_div_short(n):
  return sum([i for i in range(1,n) if n%i==0])
```

Nombre parfait

Ecrire une fonction `parfait(n)` qui renvoie `True` si l'entier n est parfait, `False` sinon. Pour cela, on pourra appeler l'une des fonctions que nous avons déjà définies dans le script pour calculer la somme des diviseurs stricts d'un nombre. On pourra utiliser la touche VAR pour retrouver les fonctions déjà définies dans le script.

```
def parfait(n):
  if somme_diviseurs(n)==n:
    return True
  else:
    return False
```

Nombres amicaux

Deux nombres sont amicaux si chacun est égal à la somme des diviseurs stricts de l'autre.
Ecrire une fonction `amicaux(a,b)` qui renvoie `True` si c'est le cas, `False` sinon.
On rappelle que l'instruction `and` permet d'ajouter plusieurs conditions au même test.

```
def amicaux(a,b):
  if somme_diviseurs(a)==b and somme_diviseurs(b)==a:
    return True
  else:
    return False
```