

**DEVOIR DE CONTROLE DU 1<sup>er</sup> SEMESTRE**

**Matière : INFORMATIQUE**

**Classes : 2<sup>ème</sup> Année MP, PC, PT et BG      Durée : 1h**

**EXERCICE 1: Suite aliquote**

On s'intéresse aux diviseurs propres des entiers naturels, plus particulièrement à la somme de leurs diviseurs propres.

Un entier naturel non nul  $d$  est un **diviseur propre** d'un entier naturel non nul  $n$  quand  $d$  est un diviseur de  $n$  avec  $d \neq n$ . Par exemple, 1, 2, 3, 4 et 6 sont les diviseurs propres de 12.

1. Définir une fonction **est\_div\_propre** qui, à partir de deux entiers naturels non nuls  $d$  et  $n$ , retourne True si  $d$  est un diviseur propre de  $n$  et False sinon.

Par exemple :

```
>>> est_div_propre(6, 12)
True
>>> est_div_propre(7, 12)
False
```

2. Définir une fonction **liste\_div\_propres** qui, à partir d'un entier naturel non nul  $n$ , retourne la liste des diviseurs propres de  $n$ .

Par exemple :

```
>>> liste_div_propres(12)
[1, 2, 3, 4, 6]
```

3. Définir une fonction **s\_div\_propres** qui, à partir d'un entier naturel non nul  $n$ , retourne la somme de ses diviseurs propres.

Par exemple :

```
>>> s_div_propres(220)
284
```

**Dans la suite**, on note  $\sigma(n)$  la somme des diviseurs propres de  $n$ .

Un entier naturel non nul est **parfait** quand il est égal à la somme de ses diviseurs propres, c'est à dire quand  $n = \sigma(n)$ . Par exemple, 6 est parfait car les diviseurs propres de 6 sont 1, 2 et 3 et leur somme vaut 6. Par contre, 12 n'est pas parfait car  $\sigma(12)$  vaut 16.

4. Définir une fonction **parfaits1** qui, à partir d'un entier naturel non nul  $n$ , retourne la liste des nombres parfaits compris entre 1 et  $n$  (inclus), **sans utiliser les listes en compréhension**.

Par exemple :

```
>>> parfaits(1000)
```

```
[6, 28, 496]
```

5. Définir une fonction **parfaits2** qui, à partir d'un entier naturel non nul  $n$ , retourne la liste des nombres parfaits compris entre 1 et  $n$  (inclus), **en utilisant les listes en compréhension**.

6. Étant donné un entier naturel non nul  $n$ , on définit la **suite aliquote**  $(a_k)_{k \in \mathbb{N}}$  de  $n$  :

$$\begin{cases} a_0 = n \\ \forall k \in \mathbb{N}, a_{k+1} = \begin{cases} 1 & \text{si } a_k = 1 \\ \sigma(a_k) & \text{si non} \end{cases} \end{cases}$$

Par exemple, les 5 premiers termes de la suite aliquote de 9 sont 9, 4, 3, 1, 1 et les 5 premiers termes de la suite aliquote de 30 sont 30, 42, 54, 66, 78.

Définir une fonction **terme\_aliquote** qui, à partir d'un entier naturel non nul  $n$  et un entier naturel  $k$ , renvoie la valeur du terme d'indice  $k$  de la suite aliquote de  $n$ .

Par exemple :

```
>>> terme_aliquote(30, 3)
```

```
66
```

## EXERCICE 2: En famille

Dans cet exercice, une famille est représentée par un dictionnaire dont les éléments sont des associations entre une clé qui est le prénom d'une personne et une valeur qui est l'**ensemble** des prénoms des enfants de cette personne.

Si une personne n'a pas d'enfants alors elle n'apparaît pas comme clé du dictionnaire.

Dans tout l'exercice on fait l'hypothèse que les personnes ont toutes des prénoms différents.

On utilise comme exemple :

```
F_ex = {'eve': {'tom', 'lea', 'luc', 'kim'},
        'tom': {'isa', 'bob'},
        'lea': {'ali'},
        'bob': {'bea', 'tim'},
        'luc': {'sam', 'jon', 'lou'},
        'sam': {'ana', 'guy'},
        'guy': {'ben'}}
```

Dans cet exemple :

- tom, lea, luc et kim sont les enfants d'eve
- isa et bob sont les enfants de tom (et donc les petits-enfants d'eve)
- lou n'est pas une clé du dictionnaire : lou n'a pas d'enfants
- etc.



Dans la suite le mot famille désigne un dictionnaire représentant une famille.

1. Définir une fonction **enfants** qui, à partir d'une famille  $F$  et d'une chaîne de caractères  $x$ , retourne l'ensemble des prénoms des enfants de  $x$  dans  $F$ . Cette fonction retourne l'ensemble vide si  $x$  n'est pas une clé de  $F$ .

Par exemple :

```
>>> enfants(F_ex, 'eve')
{'kim', 'lea', 'tom', 'luc'}
```

```
>>> enfants(F_ex, 'germaine')
set()
```

2. Définir une fonction **enfants\_ens** qui, à partir d'une famille  $F$  et d'un ensemble  $X$  de chaînes représentant des noms de personnes, retourne l'ensemble des prénoms des enfants des personnes de l'ensemble  $X$  dans  $F$ .

Par exemple :

```
>>> enfants_ens(F_ex, {'tom', 'lea'})
{'isa', 'bob', 'ali'}
```

```
>>> enfants_ens(F_ex, set())
set()
```

3. Définir une fonction **petits\_enfants** qui, à partir d'une famille  $F$  et d'une chaîne de caractères  $x$ , retourne l'ensemble des prénoms des petits-enfants de  $x$  dans  $F$ .

**Remarque :** les petits-enfants de  $x$  sont les enfants des enfants de  $x$ .

Par exemple :

```
>>> petits_enfants(F_ex, 'eve')
{'lou', 'isa', 'bob', 'ali', 'jon', 'sam'}
```

```
>>> petits_enfants(F_ex, 'lou')
set()
```

4. Définir une fonction **parents**, qui, à partir d'une famille  $F$  et d'une chaîne de caractères  $x$ , retourne l'ensemble des prénoms des parents de  $x$  dans  $F$  ( $y$  est un parent de  $x$  si  $x$  est un enfant de  $y$ ). Cette fonction retourne l'ensemble vide si  $x$  n'a pas de parent dans  $F$ .

Par exemple :

```
>>> parents(F_ex, 'tom')
{'eve'}
```

```
>>> parents(F_ex, 'germaine')
set()
```

5. Définir une fonction **freres\_soeurs**, qui, à partir d'une famille  $F$  et d'une chaîne de caractères  $x$ , retourne l'ensemble des prénoms des frères et sœurs de  $x$  dans  $F$ .

**Remarque :** les frères et sœurs de  $x$  sont les enfants des parents de  $x$ , mais sont différents de  $x$ .

**Indication :** on pourra utiliser des fonctions définies dans les questions précédentes.

Par exemple :

```
>>> freres_soeurs(F_ex, 'lea')  
{'tom', 'kim', 'luc'}
```

```
>>> freres_soeurs(F_ex, 'ben')  
set()
```

6. Définir une fonction **neveux\_nieces**, qui, à partir d'une famille  $F$  et d'une chaîne de caractères  $x$ , retourne l'ensemble des prénoms des neveux et nièces de  $x$  dans  $F$ .

**Remarque :** les neveux et nièces de  $x$  sont les enfants des frères et sœurs de  $x$ .

Par exemple :

```
>>> neveux_nieces(F_ex, 'lea')  
{'bob', 'lou', 'sam', 'jon', 'isa'}
```

```
>>> neveux_nieces(F_ex, 'germaine')  
set()
```