Devoir Surveillé 1 : Nombres Binaires et Bases de la programmation

L'évaluation porte sur 3 exercices indépendants.
Les exercices sont notés sur 18 et la rigueur, rédaction et justifications sont notés sur 2 points.
La présence des en-têtes de fonctions ou documentations compte dans les deux points de rigueur.

Exercice 1 : Conversions binaire décimal (3 points)

1. Donner la représentation binaire des nombres en base 10 suivants:

- 74_{10}
- 132_{10}
- 534_{10}

2. Donner la représentation décimale des nombres en base 2 suivants:

- 11010_2
- 1001101_2
- 101111010_2

Exercice 2 : Complétion de code (6 points)

Un nombre est un nombre d'Armstrong si la somme de chacun de ses chiffres, élevés à la puissance de son nombre de chiffre, est égale au nombre lui-même.
Exemple : le nombre 153 comporte 3 chiffres, alors on réalisera : $153 = 1^3 + 5^3 + 3^3$

Compléter la fonction `nombre_armstrong` qui prend en paramètre un nombre entier et renvoie `True` s'il est un nombre d'Armstrong, `False` sinon.
Rappel : la fonction `len(sequence)` permet de donner le nombre d'éléments dans la séquence :
Exemple $len(”Bonjour”) = 7$.
$8208 = 8^4 + 2^4 + 0^4 + 8^4$
Exemple : nombre_armstrong(8208) doit renvoyer True.

```
def nombre_armstrong(nombre : int)-> bool:
    str_nombre = str(...)
    somme = 0
    taille = len(...)
    for chiffre in ...:
        int_chiffre = int(chiffre)
        puissance = ... **taille
        somme = somme + ...
    if ... == ...:
        return True
    else:
        return False
```

Exercice 3 : Écriture de code (9 points)

Le but de cet exercice est de trouver ce que l'on appelle des nombres `parfaits`.

Un nombre est un nombre parfait s'il est égal à la somme de ses diviseurs (hormis lui-même).
Par exemple 6 est un nombre parfait car : 6 = 1 + 2 + 3 car 1,2 et 3 divisent 6.

Les diviseurs

Écrire une fonction `est_divisible` qui prend en paramètres deux entiers a et b et renvoie `True` si b est divisible par a, `False` sinon.
Exemple:

```
>>> est_divisible(10, 2)
True
>>> est_divisible(10, 3)
False
```

Compléter la fonction `tous_les_diviseurs` qui prend en paramètre un entier et renvoie une liste contenant tous ses diviseurs (lui-même non compris).
Vous utiliserez la fonction précédente.

```
def tous_les_diviseurs(n : int)-> list:
l = []
    for ... in range(...): # Compléter le for
        if ... : # Compléter le if
            l = l + [...]
return l
```

Nombres parfaits

On dispose d'une fonction `somme_elements_listes` qui donne la somme des éléments d'une liste.
Elle n'a pas a être réalisée !
Elle fonctionne ainsi :

```
    >>> l = [1,2,3,4]
    >>> somme_liste = somme_liste(l)
    >>> print(somme_liste)
    10
```

Écrire une fonction `est_parfait` qui prend en paramètre un entier n et renvoie `True` s'il est un nombre parfait, `False` sinon. Elle utilisera les fonctions précédentes.
Pour 12, ses diviseurs sont 1,2,3,4 et 6. Comme 1+2+3+4+6 = 16, il n'est pas parfait.
Pour 28, ses diviseurs sont 1,2,4,7,14. Comme 1+2+4+7+14 = 28, il est parfait
Exemple :

```
>>> est_parfait(28)
True
>>> est_parfait(19)
False
```

Écrire une fonction `nombres_parfaits_jusqu_a` qui prend en paramètre un entier n et renvoie une chaîne de caractères contenant tous les nombres parfaits de 1 à n (compris) séparés par un espace.

```
>>> nombres_parfaits_jusqu_a(3000)
'6 28 496'
```

Rappel : une liste est une séquence qui peut être parcourue comme une chaîne de caractères.

```
l = [1,3,5,2] # une liste d'entiers
for elt in l:
    print(l)
# Va afficher les entiers
1
3
5
2
```

Bonus

Écrire la fonction `somme_elements_listes` qui prend en paramètre une liste d'entiers et renvoie la somme de ses éléments.