EXERCICE 1 (4 points)

On rappelle que :

- le nombre a^n est le nombre $a \times a \times \ldots \times a$, où le facteur a apparaît n fois,
- en langage Python, l'instruction `t[-1]` permet d'accéder au dernier élément du tableau `t`.

Dans cet exercice, l'opérateur `**` et la fonction `pow` ne sont pas autorisés.

Programmer en langage Python une fonction `liste_puissances` qui prend en argument un nombre entier `a`, un entier strictement positif `n` et qui renvoie la liste de ses puissances `[a¹, a², ... , aⁿ ]`.

Programmer également une fonction `liste_puisssances_borne` qui prend en argument un nombre entier `a` supérieur ou égal à 2 et un entier `borne`, et qui renvoie la liste de ses puissances, à l'exclusion de a^0, strictement inférieures à `borne`.

Exemples :

```
>>> liste_puissances(3, 5)
[3, 9, 27, 81, 243]
>>> liste_puissances(-2, 4)
[-2, 4, -8, 16]
>>> liste_puissances_borne(2, 16)
[2, 4, 8]
>>> liste_puissances_borne(2, 17)
[2, 4, 8, 16]
>>> liste_puissances_borne(5, 5)
[]
```

EXERCICE 2 (4 points)

On affecte à chaque lettre de l'alphabet un code selon les tableaux ci-dessous :

A	B	C	D	E	F	G	H	I	J	K	L	M
1	2	3	4	5	6	7	8	9	10	11	12	13

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
14	15	16	17	18	19	20	21	22	23	24	25	26

Pour un mot donné, on détermine d'une part son *code alphabétique concaténé*, obtenu par la juxtaposition des codes de chacun de ses caractères, et d'autre part, son *code additionné*, qui est la somme des codes de chacun de ses caractères.
Par ailleurs, on dit que ce mot est « *parfait* » si le code additionné divise le code concaténé.

Exemples :

- Pour le mot `"PAUL"`, le code concaténé est la chaîne 1612112, soit l'entier 1612112.
  Son code additionné est l'entier 50 car $16 + 1 + 21 + 12 = 50$.
  50 ne divise pas l'entier 1612112 ; par conséquent, le mot `"PAUL"` n'est pas parfait.
-
- Pour le mot `"ALAIN"`, le code concaténé est la chaîne 1121914, soit l'entier 1121914.
  Le code additionné est l'entier 37 car $1 + 12 + 1 + 9 + 14 = 37$.
  37 divise l'entier 1121914 ; par conséquent, le mot `"ALAIN"` est parfait.

Compléter la fonction `est_parfait` fournie à la page suivante qui prend comme argument une chaîne de caractères `mot` (en lettres majuscules) et qui renvoie le code alphabétique concaténé, le code additionné de `mot`, ainsi qu'un booléen qui indique si `mot` est parfait ou pas.

```
dico = {"A": 1, "B": 2, "C": 3, "D": 4, "E": 5, "F": 6,
        "G": 7, "H": 8, "I": 9, "J": 10, "K": 11, "L": 12,
        "M": 13, "N": 14, "O": 15, "P": 16, "Q": 17,
        "R": 18, "S": 19, "T": 20, "U": 21, "V": 22,
        "W": 23, "X": 24, "Y": 25, "Z": 26}

def est_parfait(mot) :
    # mot est une chaîne de caractères (en lettres majuscules)
    code_concatene = ""
    code_additionne = ...
    for c in mot:
        code_concatene = code_concatene + ...
        code_additionne = ...
    code_concatene = int(code_concatene)
    if ... :
        mot_est_parfait = True
    else :
        mot_est_parfait = False
    return code_additionne, code_concatene, mot_est_parfait
```

Exemples :

```
>>> est_parfait("PAUL")
(50, 1612112, False)
>>> est_parfait("ALAIN")
(37, 1121914, True)
```