Informatique Pour Tous

Sujet d'Approfondissement 1 – Test de multiplicité

A / Fonction de découpe

On donne ci-dessous une fonction qui permette de découper un nombre en « son dernier chiffre », d'une part, et « le nombre constitué de tous ses autres chiffres (autres que le dernier) » d'autre part.

```
def decoupe(N) :  # N est un entier positif
    N_str = str(N)    # on convertit le nombre en chaine de caractères.
                      # Exemples : 132 --> "132"
    dernier_carac = N_str[-1]     # ceci est le dernier caractère de la chaine.
                                  # Exemple : "132" --> "2"
    dernier_chiffre = int(dernier_carac)    # on interprète ce caractère comme un chiffre.
                                             # Exemple "2" --> 2
    reste_des_carac = N_str[:-1]     # on récupère, par ailleurs, tous les caractères
                                     # ... sauf le dernier. Ex : "132" --> "13"
    reste_des_chiffres = int(reste_des_carac)    # et on les interprète comme un nombre.
                                                 # Ex : "13" --> 13
    return reste_des_chiffres, dernier_chiffre  # on renvoie les deux
```

Q1 – Copier-coller cette fonction dans un script *Python*, puis la tester, à l'aide d'un `print`, avec des cas simples : 17, 1379. Pourquoi y a-t-il une erreur lorsqu'on appelle `decoupe(8)` ?

On souhaite modifier la fonction afin qu'elle puisse prendre en argument des nombres (entiers naturels) constitués d'un seul chiffre ($N \leq 9$). On pourrait décider de renvoyer simplement le chiffre… mais dans ce cas on aurait un formatage de la sortie qui dépendrait de l'entrée. Avec cette solution simple, on aurait :

- `decoupe(8)   # renverrait 8          --> 1 donnée`
- `decoupe(147) # renverrait (14, 7)    --> 2 données`

Ceci est déconseillé, on préfèrera renvoyer toujours la même structure de données, quelle que soit l'entrée. On décide donc, dans le cas d'une entrée à un seul chiffre, de prendre pour convention que le « reste des chiffres » (c'est-à-dire le nombre constitué de tous les autres chiffres que le dernier) vaut 0. Ainsi :

- `decoupe(8)   # renverrait (0, 8)     --> 2 données`
- `decoupe(147) # renverrait (14, 7)    --> 2 données : même format`

Q2 – Modifier le corps de la fonction `decoupe(N)`, en utilisant une structure conditionnelle (**`if`**, **`else`**…) pour qu'elle remplisse le cahier des charges décrit ci-dessus.

Tests : tester, à l'aide de la commande `print`(…) insérée dans votre script, votre fonction sur les cas $N = 8$ et $N = 147$.

B / Divisibilité par 5

Un nombre (entier) est divisible par 5 ssi son chiffre des unités est 0 ou 5.

Q3 – Écrire une fonction `mult_5(N)` qui prenne en argument un entier N et renvoie un *booléen* : *`True`* si N est multiple de 5, *`False`* sinon. Cette fonction utilisera forcément la fonction auxiliaire `decoupe(N)`.

Aide : pour savoir si une variable `x` appartient à une liste de valeurs, les deux tests suivants sont équivalents :

```
x in [1,3,5,7]       # équivalent à #  x == 0 or x == 3 or x == 5 or x == 7
```

Variante : Modifier votre fonction en s'interdisant d'utiliser une structure conditionnelle (pas de **`if`**, **`elif`** ou **`else`**). (La fonction doit alors tenir en 2 ou 3 lignes, en-tête et return compris !)

Tests : tester, à l'aide de la commande `print`(…) insérée dans votre script, votre fonction sur les cas $N = 40$, $N = 144$ et $N = 1765$ (les réponses affichées devant être : *`True`*, *`False`*, *`True`*)

C / Divisibilité par 3

Un entier *N* est divisible par 3 si la somme de ses chiffres est un multiple de 3. On va donc appliquer l'algorithme qui consiste à faire la somme de tous les chiffres de *N*, autant de fois que nécessaire jusqu'à n'avoir plus qu'un seul chiffre. *N* sera un multiple de 3 ssi ce chiffre est un multiple de 3 (donc 0, 3, 6 ou 9).

Exemples : *N* = 189 → 1+8+9 = 18 → 1+8 = 9 ∈ $\{0,3,6,9\}$ donc 189 est divisible par 3
N = 8895 → 8+8+9+5 = 30 → 3+0 = 3 ∈ $\{0,3,6,9\}$ donc 8895 est divisible par 3
N = 83895 → 8+3+8+9+5 = 33 → 3+3 = 6 ∈ $\{0,3,6,9\}$ donc 83895 est divisible par 3
N = 83894 → 8+3+8+9+4 = 32 → 3+2 = 5 ∉ $\{0,3,6,9\}$ donc 83894 n'est pas divisible par 3

On va créer une fonction auxiliaire `somme_chiffres(N)` qui prend en argument un entier *N* et réalise la somme de ses chiffres (c'est-à-dire une étape intermédiaire, symbolisée ci-dessus par la flèche). Cette fonction s'appuiera elle-même obligatoirement sur la fonction `decoupe(N)`. L'algorithme de la fonction `somme_chiffres(N)` est imposé ci-dessous :

- Je vous laisse réfléchir à l'initialisation ! Nous allons initialiser, au minimum, une variable `somme`
- Puis nous allons utiliser `decoupe(N)` pour découper le dernier chiffre (`last`) du `reste` (2 autres variables).
- On va alors ajouter ce dernier chiffre `last` à la `somme`.
- Et nous allons reprendre ces deux dernières étapes en boucle (`while`) jusqu'à ce que le `reste` soit nul (c'est-à-dire jusqu'à avoir atteint le chiffre le plus à gauche de *N*).

Q4 – Écrire la fonction `somme_chiffres(N)` telle que décrite ci-dessus.

Test :
```
print( somme_chiffres(838940) )    # doit par exemple afficher 32
```

On peut donc maintenant écrire la fonction `mult_3(N)` qui prenne en argument un entier *N* et renvoie un *booléen* : `True` si *N* est multiple de 3, `False` sinon. Cette fonction utilisera forcément la fonction auxiliaire `somme_chiffres(N)`. L'algorithme imposé est de remplacer *N* par la somme de ces chiffres, et ce en boucle (`while`) jusqu'à ce que *N* ne compte plus qu'un seul chiffre.

Rmq : un entier naturel *N* est composé d'un seul chiffre ssi $N < 10$.

Q5 – Écrire la fonction `mult_3(N)` telle que décrite ci-dessus.

Tests : à l'aide de la commande `print( ... )` faire le test de votre fonction sur les 4 cas donnés ci-dessus (189, 8895, 83895, 83894).

D / Divisibilité par 15

Un entier *N* est divisible par 15 ssi il est divisible par 3 ET divisible par 5.

Q6 – Écrire la fonction `mult_15(N)` qui prenne en argument un entier *N* et renvoie un *booléen* : `True` si *N* est multiple de 15, `False` sinon. Cette fonction utilisera forcément les deux fonctions `mult_3(N)` et `mult_5(N)`.

Tests :
```
print(mult_15(45))   # doit afficher True
print(mult_15(33))   # doit afficher False
print(mult_15(35))   # doit afficher False
```

Variante : Modifier votre fonction en s'interdisant d'utiliser une structure conditionnelle (pas de **`if`**, **`elif`** ou **`else`**). (La fonction doit alors tenir en 2 : une ligne d'en-tête et un return, c'est tout !)