Le compte est bon !

Objectif du projet :

Le but du programme qui suit est de donner (entre autres) une réponse systématique au jeu « Le compte est bon ! » pour toute donnée de départ.

« Le compte est bon !» est une composante d'un jeu télévisé très populaire (mais qui date un peu maintenant).

Ses règles sont simples. Etant donnés :

- un nombre entier N entre 0 et 999,
- un ensemble de 6 plaques comportant chacune un des 14 entiers :
  1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100,

l'objectif est de déterminer, si elle existe, une formule mathématique permettant de retrouver le nombre N, en combinant les 6 valeurs de départ à l'aide des quatre opérateurs élémentaires, que sont : l'addition, la soustraction (lorsqu'elle donne un résultat positif), la multiplication, et la division entière (lorsqu'elle « tombe juste »). Les valeurs de départ ne peuvent être utilisées qu'une seule fois au plus chacune et on doit toujours utiliser au moins une plaque, y compris si la valeur à atteindre est 0.

Enfin, l'ordre de priorité des opérateurs doit être respecté mais l'usage de parenthèses est possible.

Ainsi, avec les 6 plaques 7, 7, 9, 10, 25, 2, on peut calculer N = 678 via la formule : (2*25 – 7)*(7 + 9) – 10

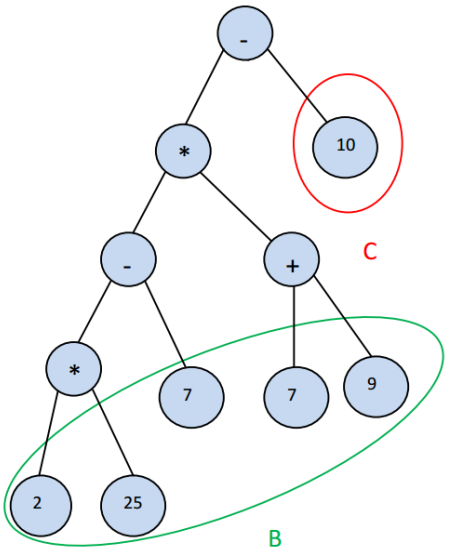
Dans cet exemple, toutes les plaques ont été utilisées mais ce n'est pas une obligation.

Dans tous les cas, et même si N ne peut être retrouvé à l'aide des 6 plaques, le programme affichera tous les nombres entiers entre 0 et 999 qui ne peuvent être calculés à partir des 6 plaques, selon les contraintes mentionnées ci-dessus.

Détails sur l'algorithme et l'implantation du projet :

Le principe essentiel de l'algorithme est de calculer, pour toute partie non vide A de l'ensemble des 6 plaques, la liste des nombres entre 0 et 999 qui peuvent être calculés avec ce sous-ensemble de plaques, en gardant "une trace" des opérations effectuées.

Cela peut se faire par induction en remarquant que si n est un nombre calculé à partir de A, il peut s'écrire sous la forme : n = n1 # n2 où # est un des 4 opérateurs + , *, -, /, n1 est un nombre entier calculé à partir d'un sous-ensemble B de A et n2 est calculé à partir de C où C est le complémentaire de B dans A : on met ainsi en évidence la structure d'arbre binaire arithmétique menant au calcul de n (cf figure ci-dessous).



Ici, $B = \{2, 25, 7, 7, 9\}$, $C = \{10\}$,
$A = B \cup C = \{2, 25, 7, 7, 9, 10\}$
et $n = n_1 - n_2$ où
$n_1 = 688 = (2 \times 25 - 7) \times (7 + 9)$ et $n_2 = 10$.

On aura donc besoin (entre autres) de modéliser chaque partie non vide de l'ensemble des 6 plaques.
Si $\{a, b, c, d, e, f\}$ est l'ensemble des 6 plaques, on modélisera un sous-ensemble de cet ensemble à l'aide d'une liste de 6 chiffres égaux soit à 1 soit à 0, de sorte que les emplacements des chiffres 1 déterminent les plaques appartenant à ce sous-ensemble.
Ainsi par exemple, la liste $[0, 1, 0, 0, 0, 1]$ représentera le sous-ensemble $\{b, f\}$ et la liste $[1, 0, 1, 1, 0, 1]$ représentera $\{a, c, d, f\}$.
Nous verrons que tout nombre $n \in [\![1, 2^6 - 1]\!]$ correspond à une et une seule liste (non identiquement nulle) de 6 chiffres égaux à 0 ou 1 (donné par son écriture en base 2). Ainsi, chaque sous-ensemble non vide d'un ensemble de 6 plaques pourra être représenté par un entier appartenant à $[\![1, 2^6 - 1]\!]$.
Pour chacun de ces sous-ensembles, l'idée sera de répertorier l'ensemble des opérations et résultats permis avec toutes les plaques de ce sous-ensemble.
Par exemple, si l'ensemble des 6 plaques est $\{5, 7, 8, 10, 75, 100\}$, on associera au sous-ensemble $\{7, 10, 75, 100\}$ la liste (incomplète) :

"$(7 + 10) \times 75 - 100$"	"$((100 \times 7)/10) \times 75$"	"$(10 - 7) \times (75 + 100)$"	...

Pour construire l'expression $(10 - 7) \times (75 + 100)$, nous voyons que nous avons eu besoin d'accéder à l'opération $10 - 7$ possible avec le sous-ensemble de deux plaques $\{7, 10\}$ et à l'opération $75 + 100$ possible avec le sous-ensemble $\{75, 100\}$. On répertoriera les résultats possibles (par opérations) avec chaque sous-ensemble de plaque (en commençant par les sous-ensembles d'une plaque, puis de deux plaques, puis de trois, etc.).

Note : Il faudra traiter les exercices ci-dessous dans l'ordre où ils apparaissent (sauf l'exercice 4 qui pourra être traité indépendamment).

Exercice 1 : Écriture d'un nombre en base 2

Préambule :

Soit $n \in \mathbb{N}^*$ (dans notre projet, on aura $n = 6$, le nombre de plaques disponibles).

On admet que tout nombre entier $m \in [\![1, 2^n - 1]\!]$ s'écrit de manière unique sous la forme

$$m = a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \cdots + a_2 \times 2^2 + a_1 \times 2^1 + a_0 \times 2^0$$

où $a_k \in \{0, 1\}$ pour tout $k \in [\![0, n-1]\!]$ (écriture de m en base 2). On peut donc associer à m la liste de n zéros ou uns $[a_{n-1}, a_{n-2}, \ldots, a_1, a_0]$. Réciproquement, toute liste non identiquement nulle L de n zéros ou uns correspond à un unique entier appartenant à $[\![1, 2^n - 1]\!]$ dont L est l'écriture en base 2.

On peut obtenir l'écriture L en base 2 d'un entier $m \geqslant 1$ comme suit :

$(*)$

```
L = []
tant que m est non nul,
    on considère le reste r de la division euclidienne de m par 2
    (r = 0 si m est pair et r = 1 sinon)
    on place r en début de liste L
    on retranche r à m et on divise m par 2
```

Par exemple, si $m = 670$, l'exécution de cet algorithme donnerait

$$\begin{aligned}
L &= [],\ m = 670\\
L &= [0],\ m = 670/2 = 335\\
L &= [1, 0],\ m = (335 - 1)/2 = 167\\
L &= [1, 1, 0],\ m = (167 - 1)/2 = 83\\
L &= [1, 1, 1, 0],\ m = (83 - 1)/2 = 41\\
L &= [1, 1, 1, 1, 0],\ m = (41 - 1)/2 = 20\\
L &= [0, 1, 1, 1, 1, 0],\ m = 20/2 = 10\\
L &= [0, 0, 1, 1, 1, 1, 0],\ m = 10/2 = 5\\
L &= [1, 0, 0, 1, 1, 1, 1, 0],\ m = (5 - 1)/2 = 2\\
L &= [0, 1, 0, 0, 1, 1, 1, 1, 0],\ m = 2/2 = 1\\
L &= [1, 0, 1, 0, 0, 1, 1, 1, 1, 0],\ m = (1 - 1)/2 = 0
\end{aligned}$$

et on a bien $670 = 2^9 + 2^7 + 2^4 + 2^3 + 2^2 + 2^1$.

1) Écrire une fonction Python prenant un entier $m \geqslant 1$ en entrée et donnant en sortie son écriture en base 2 sous forme de liste (traduire le pseudo-code $(*)$ en Python).

2) Effectuer la procédure inverse : écrire une fonction Python prenant en entrée une liste L de zéros ou de uns et donnant en sortie le nombre entier dont L est l'écriture en base 2.

Exercice 2 : Écriture en base 2 et sous-ensembles d'une liste à 6 éléments

Préambule :

On rappelle que toute partie non vide d'un ensemble $\{a, b, c, d, e, f\}$ à 6 élément peut être modélisé par un entier appartenant à $[\![1, 2^6 - 1]\!]$.

On se propose ici de classer chaque partie d'un tel ensemble suivant le nombre de ses éléments.
1) a) Écrire une fonction Python qui prend une liste L de 0 ou de 1 en entrée, et qui donne en sortie le nombre de chiffre 1 de cette liste (peut importe la taille de la liste L).
b) En déduire une fonction Python qui, étant donné un nombre $m \in [\![1, 2^6 - 1]\!]$, déterminer le nombre de 1 dans son écriture en base 2.
2) Utiliser la fonction écrite dernièrement pour construire la liste de 7 listes T définie ainsi :

- L'élément d'indice 0 de T est la liste vide.
- Pour $k \in [\![1, 6]\!]$, l'élément d'indice k de T est la liste formée des entiers $m \in [\![1, 2^6 - 1]\!]$ dont l'écriture en base 2 possède exactement k chiffre 1.

Interprétation : L'élément d'indice k de T (pour $k \in [\![1, 6]\!]$) répertorie les parties à k éléments d'un ensemble à 6 éléments.

Exercice 3 : Parties incluses dans un sous-ensemble

Préambule :
Supposons que nous voulons déterminer les opérations et résultats possibles avec un ensemble A de 5 plaques : $A = \{2, 25, 7, 7, 9\}$.
$(2 \times 25 - 7) \times (7 + 9)$ est une telle opération construite à partir des plaques $\{2, 25, 7\}$ et $\{7, 9\}$. Il faut donc avoir accès aux opérations possibles avec les sous-ensembles de plaques $B = \{2, 25, 7\}$ et $C = \{7, 9\}$. Il en va de même pour toute autre partie de A.
Plus généralement, étant donné un ensemble A de k plaques ($k \in [\![1, 6]\!]$), il est donc important de savoir répertorier tous les ensembles $B \subset A$.
Si A correspond à la liste $[1, 0, 1, 1, 0, 1]$, c'est un ensemble de quatre plaques et ses sous-ensembles B correspondent aux listes B où ne figurent pas de 1 aux endroits où 0 figure dans la liste représentant A.
Par exemple, $B = [1, 0, 1, 0, 0, 1]$ et $B' = [1, 0, 0, 1, 0, 0]$ correspondent à des sous-ensembles de A mais pas $B'' = [1, 1, 0, 0, 0, 0]$.
Important à noter :
Si A, B sont deux parties non vides d'un ensemble de 6 plaques telles que $B \subset A$ et si b (respectivement a) est l'entier de $[\![1, 2^6 - 1]\!]$ représentant B (respectivement : représentant A), alors $b \leqslant a$ et $a - b$ est l'entier représentant $A \backslash B$.
Illustrons cela sur un exemple : Si A correspond à la liste $[1, 0, 1, 1, 0, 1]$ et B à $[1, 0, 0, 1, 0, 0]$, alors on a $B \subset A$, $a = 2^5 + 2^3 + 2^2 + 2^0 = 45$ et $b = 2^5 + 2^2 = 36$. Le nombre $a - b = 45 - 36 = 9$ admet $[0, 0, 1, 0, 0, 1]$ pour écriture binaire (obtenu par "$[1, 0, 1, 1, 0, 1] - [1, 0, 0, 1, 0, 0] = [0, 0, 1, 0, 0, 1]$") et $a - b$ représente $A \backslash B$, c'est-à-dire l'ensemble C disjoint de B tel que $A = B \cup C$ (complémentaire de B dans A). Si A possède k éléments, on peut donc se contenter de rechercher les parties B de A possédant au plus $\left\lfloor \frac{k}{2} \right\rfloor$ éléments (où $\left\lfloor \frac{k}{2} \right\rfloor$ est la partie entière de $\frac{k}{2}$). En effet, quand B décrit les parties de A avec au plus $\left\lfloor \frac{k}{2} \right\rfloor$ éléments, $A \backslash B$ décrit les parties de A avec au moins $\left\lfloor \frac{k}{2} \right\rfloor$ éléments.

1) a) Écrire une fonction Python qui, étant données deux listes L et L' formées de 6 chiffres parmi 0 et 1, donne en sortie `True` si les 1 figurant dans L figurent aussi dans L' (aux mêmes places) et renvoie `False` sinon.
Interprétation : Si L représente un ensemble B et L' un ensemble A, alors la fonction renvoie `True` si et seulement si $B \subset A$.

b) En déduire une fonction Python qui, étant donnés deux entiers b et a appartenant à $[\![1, 2^6 - 1]\!]$ (codant des ensembles de plaques B et A), renvoie `True` si $B \subset A$ et `False` sinon. (On complètera au besoin les écritures en base 2 de a et b par des zéros en début de liste pour que chacune de ces écritures possède 6 chiffres).

2) **Par souci de concision, on confondra désormais un nombre $m \in [\![1, 2^6 - 1]\!]$ avec la partie d'un ensemble de 6 plaques qu'il représente.**

On veut construire ici une liste de listes de listes ! (qu'on appellera M). Pour chaque $m \in [\![1, 2^6 - 1]\!]$, on calculera le nombre k de chiffres 1 dans l'écriture en base 2 de m (k est aussi le nombre de plaques de l'ensemble que représente m) et $M[m]$ sera une liste de $\left\lfloor \frac{k}{2} \right\rfloor$ listes définie comme suit : Pour $i \in [\![0, \left\lfloor \frac{k}{2} \right\rfloor - 1]\!]$, $M[m][i]$ sera la liste des parties de m à $i + 1$ éléments.

En utilisant la liste T construite dans l'exercice 2 et la fonction de la question 1) b), écrire une fonction Python (sans argument, ou prenant T comme argument) permettant de construire la liste M (la liste $M[0]$ pourra être vide).

L'exercice qui suit est indépendant des précédents.

Exercice 4 : Génération des plaques

1) Écrire une fonction Python sans argument et donnant en sortie une liste de 6 nombres choisis au hasard parmi 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 25, 50, 75, 100 (avec répétitions éventuelles).

2) Écrire une fonction Python comme à la question 1) mais sans choisir deux fois le même nombre.

Indications : on procèdera à des "tirages sans remise" de l'ensemble des 14 nombres. Pour ne pas tirer une même plaque, on échangera celle-ci avec la dernière plaque non tirée. Illustration d'une itération (avec en gris les plaques déjà choisies) :

1	2	3	4	6	8	9	25	50	5	7	10	75

Choix aléatoire d'une plaque (blanc sur fond noir)

1	2	3	4	6	8	**9**	25	50	5	7	10	75

échange avec la dernière plaque non tirée

1	2	3	4	6	8	50	25	9	5	7	10	75

(la dernière plaque est alors à choisir parmi 1, 2, 3, 4, 6, 8, 50 et 25)

3) Écrire une fonction Python sans argument proposant à l'utilisateur de saisir ses 6 plaques suivant trois proposition :

1 - Plaques choisies par l'utilisateur (sous forme de liste)

2 - Plaques choisies aléatoirement (avec éventuellement répétition)

3 - Plaques choisies aléatoirement sans répétition.

La fonction devra donner en sortie la liste des 6 plaques générées respectant l'option choisie par l'utilisateur.

Exercice 5 : Génération des opérations possibles pour chaque sous-ensemble de plaques

Préambule :

On suppose donné un ensemble E de 6 plaques (sous forme d'une liste de 6 éléments telle que générée dans l'exercice précédent).

Le but de l'exercice est d'écrire une liste N de $2^6 = 64$ listes qui, pour chaque $m \in [\![1, 2^6 - 1]\!]$, aura pour m-ième élément une liste d'expressions arithmétiques utilisant les plaques du sous-ensemble déterminé par m. Par exemple, si $E = [2, 8, 10, 25, 50, 75]$ et si m correspond au sous-ensemble $[2, 8, 50]$, la liste $N[m]$ contiendra "$50 - (2 \times 8)$" ainsi que "$(8/2) + 50$".

1) On évalue numériquement une expression arithmétique (sous forme de chaîne de caractère) avec la fonction `eval`. Par exemple, `eval("(10+5)/3")` donnera 5.

Écrire une fonction Python prenant en entrée deux expressions arithmétiques $E1$ et $E2$ (sous forme de chaînes de caractères), un des quatre opérateurs `op` (sous forme de chaîne : "$+$", "$-$", "$\times$" ou "$/$"), et donnant en sortie le couple formé de l'expression arithmétique :

($E1$)`op`($E2$) et du résultat de cette expression. (Voir tous les cas particuliers ci-dessous avant de vous lancer)

Exemple 1 :

En entrée : "$(50/2) - 10$", "25×3", "$+$" en sortie le couple ("$((50/2) - 10) + (25 \times 3)$", 90)

Si le résultat est négatif, on renverra ($E2$)`op`($E1$) (au lieu de ($E1$)`op`($E2$)) et le résultat de cette opération.

Exemple 2 :

En entrée : "$25 - 10$", "75", "$-$" en sortie le couple ("$(75) - (25 - 10)$", 60)

Si l'opération `op` est une division qui ne tombe pas juste, la fonction devra considérer ($E2$)`op`($E1$)

- Si cette dernière division tombe juste, on renvoie en sortie ($E2$)`op`($E1$) et le résultat de cette opération.

  Exemple 3 :

  En entrée : "2", "4×8", "/". En sortie : ("$(4 \times 8)/(2)$", 16).

- Si les deux divisions ($E1$)`op`($E2$) et ($E2$)`op`($E1$) ne tombent pas juste, la fonction devra renvoyer `False`.

Exemple 4 :

En entrée : "3×9", "5", "/" en sortie : `False`.

Attention : les divisions par zéro sont interdites ! Ainsi, ne tester que les divisions possibles ! Si $E1$ et $E2$ valent 0, la fonction devra renvoyer `False`.

2) Écrire une fonction Python initialisant N comme une liste de 64 listes vides. Pour chaque $m \in [\![1, 2^6 - 1]\!]$ correspondant à un sous-ensemble d'une seule plaque (m est une puissance de 2), placer le chiffre de cette plaque dans la liste $N[m]$ (avec un seul chiffre, on ne peut pas faire d'opération !). La fonction doit donner N ainsi initialisée en sortie.

3) À l'aide des tableaux T et M des exercices 2 et 3 et en utilisant la fonction de la question 1), écrire une fonction qui continue la construction de N.

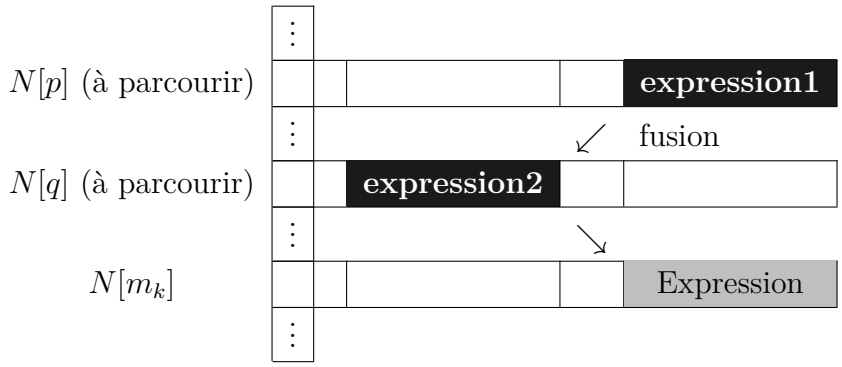
Aide :

Pour construire les listes d'expressions arithmétiques $N[m_k]$ où $m_k \in [\![1, 2^6 - 1]\!]$ correspond à un sous-ensemble A_k de k plaques (d'abord $k = 2$, puis $k = 3$,etc. jusqu'à 6), on écrira une boucle `for k in range(2,7):`. On pourra aussi utiliser la liste $T[k]$ des nombres correspondant à ces sous-ensembles.

L'idée pour construire $N[m_k]$ est la suivante : pour tout entier $p \subset m_k$ correspondant à une partie (non vide) B d'au plus $\left\lfloor \frac{k}{2} \right\rfloor$ plaques de A_k (utiliser le tableau M de l'exercice 3 pour accéder à ces entiers), on considère q tel que $p + q = m_k$ (et on note C le sous-ensemble de plaques associé à q. On a donc $C = A_k \backslash B$).

Toute expression arithmétique construite avec les plaques de B combinée à l'aide de l'un des quatre opérateurs $+, -, \times, /$ avec une expression arithmétique de C donne une expression arithmétique utilisant

toutes les plaques de A_k. Cette opération de "fusion" des expressions arithmétiques peut se faire à l'aide de la fonction de la question 1 suivant le schéma ci-dessous :



4) Dans cette question, on complète un peu ce qui a été réalisé aux deux questions précédentes.
Utiliser un tableau pour faire en sorte que lors des remplissage des listes $N[m]$ de la question 3), il n'apparaisse pas deux fois une expression menant à un même résultat avec les plaques définies par m. Concrètement, on pourra ne considérer que les résultats inférieurs ou égaux à 10^6 et construire un tableau Tab à 10^6+1 éléments initialisé avec 10^6+1 entrées `False` et tel que, lors de la construction de $N[m]$, $Tab[k] = (m, h)$ si k a pu être calculé à partir des plaques définies par m (pour tout $k \in [\![0, 10^6]\!]$) par une expression arithmétique située dans $N[m][h]$.
Pour tout nombre k entre 0 et 999, ce tableau pourra aussi permettre de détecter si le nombre k a pu être calculé à partir d'un sous-ensemble de l'ensemble des 6 plaques.
Pour cela, il faut savoir que $Tab[k]$, s'il est constitué d'un couple (m, h), est considéré comme `True` vu comme un booléen.

Exercice 6 : Synthèse

On suppose donnée la liste E des 6 plaques
1) Écrire une fonction Python sans argument proposant à l'utilisateur de saisir un nombre $a \in [\![0, 999]\!]$ de deux manière :
1 - saisie automatique (aléatoire)
2 - saisie manuelle par l'utilisateur.
La fonction doit afficher en sortie le nombre a en respectant l'option choisie par l'utilisateur.
2) À l'aide des résultats et des structures de données de l'exercice 5, indiquer si le nombre a peut être calculé à l'aide des plaques. Si tel est le cas, afficher : "Le compte est bon !" puis afficher une expression arithmétique permettant de calculer a avec les plaques.
Si le nombre a ne peut être calculé, donner une expression arithmétique approchant au mieux a avec les plaques de E.

Extensions possibles :

- Interface graphique avec Tkinter.
- Cacher trois éléments (nombres ou opérateurs) dans l'expression arithmétique fournie en réponse à la question 2) de l'exercice 6, et évaluer les réponses proposées par l'utilisateur (avec un nombre limité d'essai).
- Déterminer statistiquement les fréquences des nombres de l'intervalle $[\![0, 999]\!]$ les plus souvent atteints pour plusieurs données aléatoires de 6 plaques.