Le compte est bon

Qui n'a jamais rêvé d'épater sa grand-mère en gagnant à tous les coups au jeu « Des chiffres et des lettres » ? Une partie du jeu est « Le compte est bon » dans lequel il faut atteindre un total à partir de chiffres donnés et des quatre opérations élémentaires. L'autre partie du jeu est « Le mot le plus long », cette fois il faut trouver le plus long mot français à partir d'un tirage de lettres. Pour ces deux jeux les ordinateurs sont plus rapides que les humains, il ne te reste plus qu'à écrire les programmes !

La résolution complète du « Compte est bon » utilise un algorithme récursif.
Quatre activités sont proposées :

1. on commence par générer un tirage de nombres ;
2. on continue par un algorithme très simple qui teste beaucoup de solutions mais ne fonctionne pas pour certains cas ;
3. on continue avec un problème plus simple : atteindre une somme fixée avec des nombres donnés ;
4. enfin on mixe nos deux activités pour résoudre complètement notre jeu.

Cours 1 (Le compte est bon)**.**
On se donne une liste d'entiers et un total à atteindre. Il faut réaliser ce total à l'aide des opérations « + », « − », « × » et « / ». Exemple :

$$[1, 3, 6, 8, 75, 100] \qquad T = 524$$

Une solution : $6 - 1 = 5$, puis $5 \times 100 = 500$, puis $3 \times 8 = 24$, $500 + 24 = 524$!
Voici les règles du jeu :

- le total à atteindre est un entier tiré au hasard entre 100 et 999,
- on dispose de 6 plaques tirées au hasard sur lesquelles sont inscrits des entiers,
- les plaques sont tirées parmi 28 plaques (elles sont en double) :

$$1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9, 10, 10, 25, 25, 50, 50, 75, 75, 100, 100$$

- chacune des 6 plaques ne peut être utilisée qu'une seule fois (mais on n'est pas obligé d'utiliser toutes les plaques),
- les seules opérations autorisées sont « + », « − », « × » et « / »,
- les calculs ne se font qu'avec des entiers positifs (les soustractions doivent donner un résultat positif, les divisions doivent avoir un reste nul).

Il n'est pas toujours possible de trouver le bon résultat. Dans ce cas, on cherche à s'approcher le plus possible du total.

Activité 1 (Tirage)**.**

Objectifs : programmer le tirage du total et des plaques.

1. Programme une fonction `tirage_total()` qui renvoie un entier au hasard compris entre 100 et 999.

   Indication. Utilise la fonction `randint(a,b)` du module `random`.

2. Programme une fonction `tirage_chiffres()` qui renvoie une liste de 6 entiers tirés au hasard parmi la liste des plaques possibles.

   Exemple de tirage : $[3,5,7,7,25,100]$.

   Indication. Il s'agit d'un tirage sans remise : une plaque tirée est ensuite écartée pour le reste du tirage. Comme l'ordre n'a pas d'importance pour le jeu, on peut renvoyer une liste ordonnée.

3. Programme une fonction `operation(a,b,op)` où a et b sont deux nombres et `op` est un caractère parmi `'+'`, `'-'`, `'*'`, `'/'` et qui renvoie le calcul demandé parmi $a+b$, $a-b$, $a\times b$, a/b.

   Exemple. `operation(7,5,'-')` renvoie $7-5=2$.

Activité 2 (Recherche basique)**.**

Objectifs : programmer simplement une recherche brutale mais pas complète.

Solutions séquentielles.
Dans un premier temps on va chercher seulement les solutions séquentielles, dans lesquelles les calculs s'enchaînent de gauche à droite. Par exemple avec : $[2,3,4,5]$

- on peut obtenir séquentiellement 29 par le calcul
$$\big((2+3)\times 5\big)+4=29$$
c'est-à-dire que l'on fait les calculs un par un en repartant à chaque fois du résultat précédent : $2+3=5$, puis $5\times 5=25$, puis $25+4=29$;
- par contre on ne peut obtenir séquentiellement le total de 49, même si on pourrait l'obtenir par les règles normales : d'une part $2+5=7$, d'autre part $3+4=7$ et on mixe ces deux sous-résultats $7\times 7=49$.

Avec deux chiffres.
Voici comment calculer toutes les opérations c_1+c_2 et $c_1\times c_2$ à partir d'une liste de chiffres et afficher si le total souhaité (ici 18) est atteint.

```
chiffres = [2,3,5,6,8,10]           # Plaques disponibles
total = 18                          # Objectif

chiffres1 = list(chiffres)          # Copie
for c1 in chiffres1:                # Premier chiffre
    chiffres2 = list(chiffres1)     # Copie
    chiffres2.remove(c1)            # Retirer la plaque déjà choisie

    for op in ['+','*']:           # Opérations

        for c2 in chiffres2:        # Second chiffre

            calcul = operation(c1,c2,op)   # Résultat du calcul

            if calcul == total:            # Total atteint ?
                print("Trouvé !",c1,op,c2,'=',calcul)
```

1. Analyse ces ligne de code et transforme-le en une fonction `deux_plaques(total,chiffres)` qui gère cette fois les quatre types d'opérations. Est-ce que le total de 18 peut être atteint avec les chiffres disponibles $[2,3,5,6,8,10]$? Si oui, de combien de façons différentes ?
2. Modifie ton programme pour rechercher tous les calculs possibles avec trois plaques, c'est-à-dire $(c_1 \square c_2) \circ c_3$, où $\square$ et $\circ$ sont des opérations parmi les quatre opérations autorisées.
3. Persévère avec une fonction `recherche_basique(total,chiffres)` qui teste si on peut obtenir le total demandé à l'aide des 6 plaques données par la liste `chiffres`.

   Indications. Il faut imbriquer beaucoup de boucles ! On ne s'occupe pas encore des règles strictes du « Compte est bon », on s'autorise des soustractions négatives et des divisions non entières.

   Exemples :
   - Laisse la machine obtenir 809 à partir de $[2,3,6,8,75,100]$.
   - Atteins 779 avec $[5,6,7,8,25,50]$.
   - Peux-tu trouver 773 avec $[2,4,6,8,10,50]$? Et 769 avec ces mêmes chiffres ?
4. On se donne un tirage de 6 plaques. Combien l'algorithme teste-t-il de combinaisons ? Quel ordre de grandeur de durée met `Python` pour tester toutes ces possibilités ? (Donne la réponse sous la forme 0.1 seconde, 1 seconde, 10 secondes, 100 secondes, 1000 secondes, 10 000 secondes…)

   Pour le nombre de combinaisons, aide-toi des calculs plus simples suivants :
   - Combien y a-t-il de combinaisons avec une seule opération $c_1 \square c_2$? Réponse : il y a 6 choix pour c_1 (une plaque parmi les 6), il y a 4 opérations possibles, et enfin il y a 5 choix pour c_2 (une plaque parmi les 5 plaques restantes). Bilan :
     $$6 \times 4 \times 5 = 120 \text{ combinaisons.}$$
   - Avec deux opérations $(c_1 \square c_2) \circ c_3$, il y a
     $$6 \times 4 \times 5 \times 4 \times 4 = 1920 \text{ combinaisons.}$$

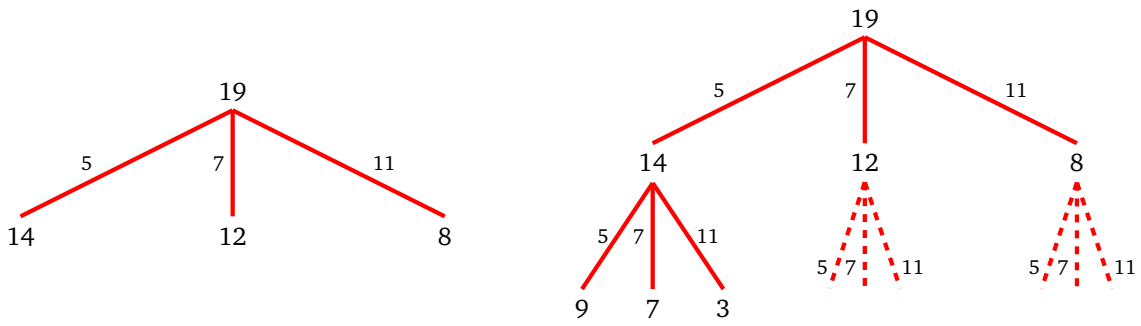
Activité 3 (Parcours d'arbre)**.**

Objectifs : résoudre des problèmes en parcourant des arbres. Cette activité utilise la récursivité.

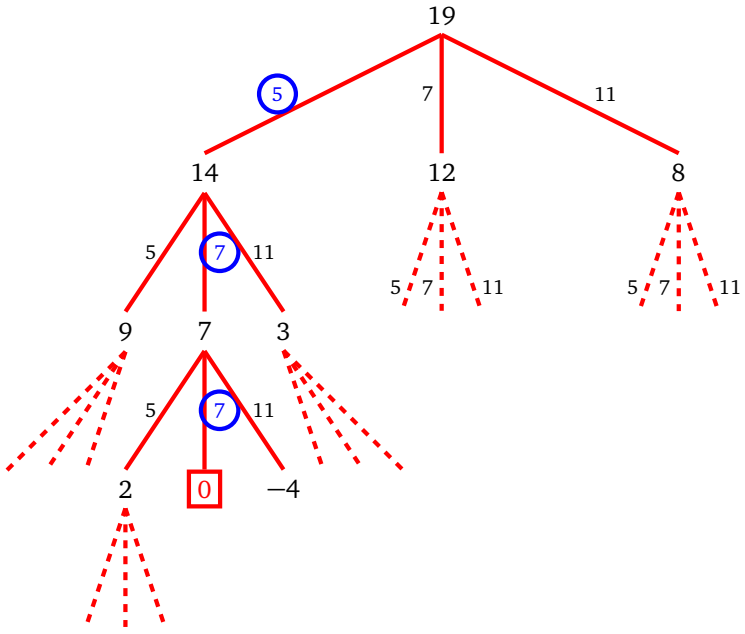
Atteindre une somme.
On se donne une liste de nombres, par exemple $[5,7,11,13]$. À partir de ces nombres on doit obtenir une somme S fixée, par exemple $S = 28$. Ici c'est possible par exemple :

$$5+5+7+11 = 28$$

Remarques. Ici on peut utiliser plusieurs fois le même nombre ; on n'est pas obligé d'utiliser tous les nombres ; il n'y a pas toujours de solution ; il peut aussi y avoir plusieurs solutions.
Pour résoudre ce problème, on teste chacun des entiers possibles, ici 5, puis 7, puis 11, puis 13. Commençons par tester 5. Si on choisit 5 alors la somme qu'il reste à atteindre est $S' = S - 5 = 23$. On a donc simplifié le problème. Si on avait choisit 7 alors la somme qu'il reste à atteindre serait $S' = S - 7 = 21$, etc.
Cela s'écrit sous la forme d'un arbre : les arêtes correspondent à un entier, les sommets à la somme qu'il reste à atteindre. L'arbre se termine quand la somme à atteindre est 0 (on a gagné) ou est négative (cette branche ne fonctionne pas).
Voyons l'exemple de la somme $S = 19$ à atteindre avec les entiers $[5,7,11]$. Sur l'arbre de gauche la somme S avec pour enfants les valeurs $S-5$, $S-7$ et $S-11$. Sur l'arbre de droite on poursuit les calculs.



Si un sommet a une somme à atteindre négative c'est terminé, le choix testé ne convient pas. Si par contre la somme vaut 0, c'est que la somme est réalisée. Les entiers à choisir pour réaliser la somme initiale sont ceux lus sur les arêtes. Sur la figure ci-dessous, on arrive à obtenir un sommet dont la somme est 0 (dans le carré), les arêtes qui permettent d'atteindre ce sommet portent les poids 5, puis 7, puis 7 (dans les cercles). Et effectivement $5+7+7=19$.



Programme une fonction `atteindre_somme(S,chiffres)` qui renvoie une solution possible. Par exemple `atteindre_somme(53,[6,7,15])` renvoie `[6,6,6,6,7,7,15]` et effectivement $6+6+6+6+7+7+15=53$.
Voilà l'algorithme proposé qui correspond au parcours d'un arbre jusqu'à atteindre la somme S.

Algorithme.

- — Entête : `atteindre_somme(S,chiffres)`
  — Entrée : une somme S et une liste d'entiers strictement positifs.
  — Sortie : une combinaison, sous la forme d'une liste, permettant de réaliser la somme ou bien `None` en cas d'échec.
  — Action : fonction récursive.
- *Cas terminaux.* Si $S = 0$ alors renvoyer la liste vide `[]` qui va plus tard contenir le parcours gagnant. Si $S < 0$, renvoyer `None`.
- *Cas général.* Pour chaque élément x de la liste des chiffres :
  — par un appel récursif, on définit `parcours` comme le résultat de `atteindre_somme(S-x,chiffres)`,
  — si `parcours` ne vaut pas `None` alors on lui ajoute x en tête de liste, puis on renvoie `parcours`.
- Renvoyer `None` (c'est le cas uniquement si rien n'a été renvoyé au cours des instructions précédentes).

Activité 4 (Le compte est bon)**.**

Objectifs : programmer la solution du « Compte est bon » en s'inspirant des deux activités précédentes.

Programme une fonction `compte_est_bon(total,chiffres)` qui renvoie une solution possible au problème d'atteindre le total donné avec une liste de nombres donnés. Par exemple avec les nombres `chiffres = [2, 3, 7, 9, 9, 25]` on peut atteindre `total = 457`. Avec ces données la fonction `compte_est_bon(total,chiffres)` renvoie :

```
['2+3=5', '5+9=14', '7+25=32', '14*32=448', '9+448=457', '457']
```

Ce qui donne le détail des opérations à effectuer :

$$\big(2+3+9\big)\times(7+25)+9=457$$

Il fallait y penser !
Voici l'algorithme qui permet de résoudre le problème. Il est préférable d'avoir bien compris les deux activités précédentes.

Algorithme.

- — Entête : `compte_est_bon(total,chiffres)`
  — Entrée : un total à atteindre et une liste d'entiers strictement positifs.
  — Sortie : une liste d'instructions permettant d'atteindre le total ou bien `None` en cas d'échec.
  — Action : fonction récursive.
- *Cas terminaux.* Si le total à atteindre est un élément de la liste `chiffres` alors renvoyer la liste `[total]` qui contient ce total comme seul élément. Si la liste `chiffres` ne contient aucun élément ou bien un seul élément (qui n'est pas le total) alors renvoyer `None`.
- *Cas général.* Ordonner la liste des chiffres *du plus grand au plus petit*.
  Pour chaque chiffre c_1 de la liste, pour chaque chiffre c_2 plus loin dans la liste, pour chaque opération parmi « + », « − », « × » et « / » :
  — former une nouvelle liste `nouv_chiffres` à partir de `chiffres` en retirant c_1 et c_2,
  — noter `calcul` le résultat de l'opération $c_1 + c_2$ ou $c_1 \times c_2$,... selon l'opération,
  — on ne continue que si `calcul` est strictement positif et si c'est bien un entier,
  — on ajoute `calcul` à la liste `nouv_chiffres`,
  — par un appel récursif, on définit `parcours` comme le résultat de `compte_est_bon(total,nouv_chiffres)`,
  — si `parcours` ne vaut pas `None` alors c'est une liste d'instructions et on lui ajoute en tête de liste une instruction sous la forme d'une chaîne de caractères du type « $c_1 + c_2 =$ `resultat` » ou « $c_1 \times c_2 =$ `resultat` »,..., puis on renvoie `parcours`.
- Renvoyer `None` (c'est le cas uniquement si rien n'a été renvoyé au cours des instructions précédentes).

Exemple. Il faut bien comprendre qu'à chaque appel de la fonction, le total à atteindre reste le même, mais que la liste des chiffres change et diminue. Voyons un exemple dans lequel il faut atteindre 27 avec les chiffres $[4,5,6,7]$. L'appel de la fonction est `compte_est_bon(27,[4,5,6,7])`. Dans le corps de cette fonction, à un moment, on va tester l'opération 4×5, on retire alors 4 et 5 de la liste des chiffres (car on n'a plus le droit de les utiliser), mais en échange on rajoute le résultat 20 ($= 4 \times 5$) que l'on peut utiliser dans la suite. Il s'agit donc maintenant d'atteindre 27 mais avec les chiffres $[20,6,7]$. L'appel récursif est donc `compte_est_bon(27,[20,6,7])`. Lors de ce nouvel appel, on va faire l'opération $20 + 7$ (nous savons déjà que cela va être bon), on remplace les chiffres 20 et 7 par le résultat 27, et on effectue l'appel `compte_est_bon(27,[27,6])`. Comme l'un des chiffres est le total à atteindre, c'est un cas terminal gagnant ! La fonction renvoie l'historique des calculs « $4 \times 5 = 20$ », « $20 + 7 = 27$ ».

Indications.

- Ordonner la liste des chiffres du plus grand au plus petit est important. Cela permet de faire les opérations dans le bon sens : $7 - 5$ et pas $5 - 7$, $6/3$ et pas $3/6$.
- Voici comment tester si un nombre est un entier : `isinstance(x,int)`.

Voici des exemples à calculer :

- avec les chiffres $[2,4,5,6,8,10]$ il est possible d'atteindre 850 et 852 mais pas 851,
- dresse la liste des totaux compris entre 100 et 999 impossibles à atteindre avec ces chiffres,
- pour les chiffres $[1,2,5,7,75,100]$ vérifie que l'on peut atteindre tous les totaux possibles entre 100 et 999 (attention les calculs deviennent longs !),
- l'un des pire tirages est $[10,10,25,50,75,100]$ pour lequel il y a plus de totaux irréalisables que de totaux possibles.