L'objectif de ce TD est de continuer à s'entraîner à écrire des fonctions récursives. Cette fois-ci, nous allons utiliser notre ordinateur pour explorer de façon systématique toutes les possibilités offertes par des lancers de dés à six faces.

Dans ce TD, toutes les fonctions demandées **doivent être récursives**.

Exercice 1 : énumération des lancers possibles

Question 1

!!! question " " En guise d'échauffement, implémenter une fonction `lance(nb_des)` renvoyant la somme des valeurs du nombre de dés demandé lancés aléatoirement (une seule fois).

Correction question 1

Cliquez ici pour révéler la correction.

Cette première fonction devrait normalement être est assez simple à réaliser :

```
def lance(nb_des):
    """Renvoie la somme des valeurs du nombre de dés lancés aléatoirement."""
    if nb_des == 0:
        return 0
    return random.randint(1, 6) + lance(nb_des - 1)
```

Question 2

!!! question " " Sans lien avec la question précédente, on souhaite maintenant énumérer tous les lancers possibles pour un nombre de dés donné. Il n'est donc plus question d'aléatoire ici. Par énumération nous entendons ici afficher sur la sortie standard tous les lancés de dés possibles.

```
Par exemple pour 2 dés on cherche à afficher les lancers suivants.
La première colonne représente le premier dé, et la seconde le deuxième.

```
1 1
1 2
1 3
...
2 1
2 2
...
6 5
6 6
```

Pour ce faire, on utilise un tableau dynamique (donc une `list` Python) `des` dont chaque ca
```

`Initialement, le tableau dynamique est rempli de `0` pour indiquer qu'aucun dé n'a encore ét`
`Il sera ensuite rempli avec des valeurs entre 1 et 6 au cours de l'énumération.`

`On effectue une récurrence sur le nombre de dés non encore lancés.`

`Implémenter une fonction récursive `enumere_lancers_rec(des, des_restants)`, où `des` contie`

`Implémenter une fonction `enumere_lancers(nb_des)` qui appelle `enumere_lancers_rec(des, des`

Correction question 2

Cliquez ici pour révéler la correction.

Le paramètre optionnel `somme` dans la fonction suivante sera ajouté pour répondre à la question 3, on l'ignore pour le moment.

La complexité de `enumere_lancers(nb_des)` est `6^nb_des`.

```python
def enumere_lancers_rec(des, des_restants, somme=None):
    """Enumere toutes les lancers possibles de des_restants dés.

    Nombre d'appels récursifs = 6^des_restants.
    """

    # Cas de base
    if des_restants == 0:
        if sum(des) == somme:
            print("***", des, "***")
        else:
            print(des)
        return

    # Sinon on tire le dés "courant" 6 fois
    # et on fait les appels récursifs
    for de_courant in range(1, 7):
        des[-des_restants] = de_courant
        enumere_lancers_rec(des, des_restants - 1, somme)


def enumere_lancers(nb_des):
    """Enumere toutes les lancers possibles de nb_des dés: complexité = 6^nb_des"""
    enumere_lancers_rec([0] * nb_des, nb_des)
```

Correction vidéo proposant une autre façon de faire :

Question 3

!!! question " " Implémenter une fonction `enumere_lancers_somme(nb_des, somme)` affichant tous les lancers comme `enumere_lancers(nb_des)` mais qui "encadre" l'affichage des lancers dont la somme vaut `somme` par "***".

`Afin de factoriser notre code, on utilisera la fonction `enumere_lancers_rec` de la questio`
`Nous ajouterons le nécessaire à cette fonction pour pouvoir répondre à cette question tout e`

Correction question 3

Cliquez ici pour révéler la correction.

Pour ne pas dupliquer de code, on ajoute donc un paramètre optionnel `somme` à la fonction `enumere_lancers_rec(des, des_restants)` comme c'est le cas dans le code ci-dessus. Et ensuite on appelle la fonction avec une valeur différente de `None` pour ce paramètre `somme` comme ci-dessous.

```
def enumere_lancers_somme(nb_des, somme):
    """Enumere toutes les lancers possibles de nb_des dés.

    Les lancés dont la somme vaut somme sont affichés entre "***".
    """
    enumere_lancers_rec([0] * nb_des, nb_des, somme=somme)
```

Exercice 2 : compter le nombre de lancers

Question 1

!!! question " " Implémenter une fonction `compte_occurence_somme(nb_des, somme)` renvoyant le nombre de lancers du nombre de dés donné atteignant la somme demandée. On essaiera d'éviter d'énumérer toutes les combinaisons possibles.

Correction question 1

Cliquez ici pour révéler la correction.

On va élaguer le plus possible de branches dans l'arbre d'appels. Néanmoins, cette solution reste exponentielle en le nombre de dés dans le pire cas. Pour se sortir de ça, il faudrait faire de la programmation dynamique (abordée plus tard dans les cursus Ensimag).

```
def compte_occurence_somme(nb_des, somme):
    """Renvoie le nombre de lancers de la taille donnée atteignant la somme demandée."""

    # Cas de base
    if nb_des == 0:
        if somme == 0:
            return 1
```

```
        return 0

    # Si en ne faisant que des 1 ensuite on dépasse
    # alors on peut s'arréter
    if nb_des * 1 > somme:
        return 0

    # Si en ne faisant que des 6 ensuite on y arrive pas
    # alors on s'arrête aussi
    if nb_des * 6 < somme:
        return 0

    # Sinon on tire le dés "courant" 6 fois
    # et on fait les appels récursifs
    compte = 0
    for de_courant in range(1, 7):

        # Sinon on fait l'appel récursif
        compte += compte_occurence_somme(nb_des - 1, somme - de_courant)

    return compte
```

Correction vidéo :

Question 2

!!! question " " On suppose disposer d'une fonction `fonction_validation` prenant un tableau dynamique de dés en argument. Cette fonction classe les lancers de dés en valides ou invalides selon un certain critère arbitraire : elle renvoie `True` lorsqu'un lancer doit être considéré comme valide et `False` dans le cas contraire.

Implémenter une fonction `compte_lancers_valides(nb_des, fonction_validation)` renvoyant pou

Correction question 2

Cliquez ici pour révéler la correction.

Ici on ne peut plus élaguer, il faut donc explorer l'espace des possibles **complètement**.

```
def compte_lancers_valides_rec(fonction_validation, des, des_restants):
    """Renvoie le nombre de lancers valides de nb_des dés."""

    # Cas de base
    if des_restants == 0:
        return int(fonction_validation(des))
```

```
    # Sinon on tire le dés "courant" 6 fois
    # et on fait les appels récursifs si besoin
    compte = 0
    for de_courant in range(1, 7):
        des[des_restants - 1] = de_courant
        compte += compte_lancers_valides_rec(fonction_validation, des, des_restants - 1)

    return compte


def compte_lancers_valides(nb_des, fonction_validation):
    """Renvoie le nombre de lancers valides de nb_des dés."""
    return compte_lancers_valides_rec(fonction_validation, [0] * nb_des, nb_des)
```

Exercice 3 : est-ce que `1, 2, 4` est différent de `4, 1, 2` ? (pour aller plus loin)

Quand on joue aux dés, et qu'on les lance tous d'un coup, "l'ordre" n'a aucune importance. Autrement dit, avec 3 dés par exemple, le lancer `1, 2, 4` est équivalent au lancer `4, 1, 2`.

Question 1

!!! question " " Reprendre la question 2 de l'exercice 1 pour énumérer seulement les lancers distincts (relativement à la permutation des dés).

Correction question 1

Cliquez ici pour révéler la correction.

L'idée est de ne considérer que les lancers où les valeurs des dés sont triées par ordre croissant par exemple. Pour ce faire, une solution consiste à ajouter un paramètre `vmin` à notre fonction récursive. Ce paramètre indique la valeur minimum des prochains dés à tirer.

```
def enumere_lancers_distincts_rec(des, des_restants, vmin):
    """Enumere toutes les lancers distincts possibles de des_restants dés."""

    # Cas de base
    if des_restants == 0:
        print(des)
        return

    # Sinon on tire le dés "courant" 6 fois
    # et on fait les appels récursifs
    for de_courant in range(vmin, 7):
        des[-des_restants] = de_courant
```

```
        enumere_lancers_distincts_rec(des, des_restants - 1, de_courant)


def enumere_lancers_distincts(nb_des):
    """Enumere toutes les lancers distincts possibles de nb_des dés.

    Complexité = (nb_des + 5, 5)
               = (nb_des + 5)! / (5! * (nb_des + 5 - 5)!)
               = (nb_des + 5)! / (5! * nb_des!)

    Pour 3 dés ça donne : (8*7*6) / 5! = 56.

    Sinon pour info, itertools.combinations_with_replacement(range(1, 7), 3)
    fait exactement ce qu'on veut ici :)
    """
    enumere_lancers_distincts_rec([0] * nb_des, nb_des, 1)
```

Question 2

!!! question " " Combien de lancers distincts existent avec `nbdes` dés ?

Correction question 2

Cliquez ici pour révéler la correction.

Le nombre de lancers distincts est le nombre de combinaisons de taille `nbdes` avec répétition parmi un ensemble de taille `6` qui vaut :

$$\binom{nbdes + 5}{5}$$

Sinon pour information, `itertools.combinations_with_replacement(range(1, 7), 3)` fait exactement ce qu'on veut :)