Exercices - Semaine 6

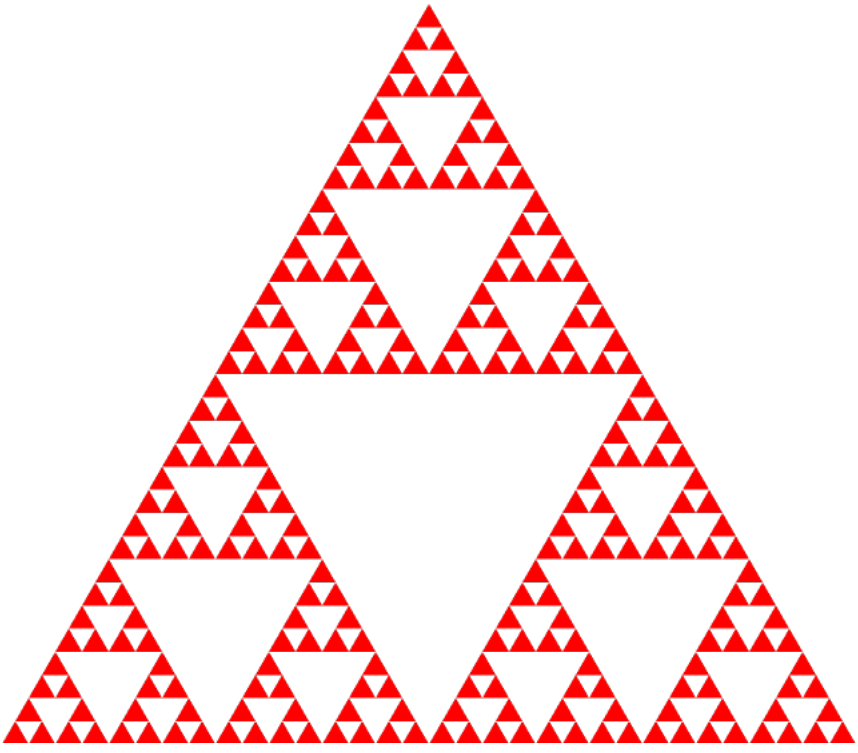
1. Somme de nombres

Réalisez une fonction récursive qui permet de calculer la somme des nombres de 0 à *n*, où *n* est le paramètre de la fonction.

Pour cela, commencez par identifier le cas de base et le cas récursif. Puis définissez une fonction qui commence par regarder s'il s'agit d'un cas de base ou d'un cas récursif, puis qui agit en conséquence.

2. Triangle de Sierpiński

À l'aide de PyTamaro, réalisez le dessin ci-dessous, appelé *triangle de Sierpiński*.
Donnez une implémentation récursive.

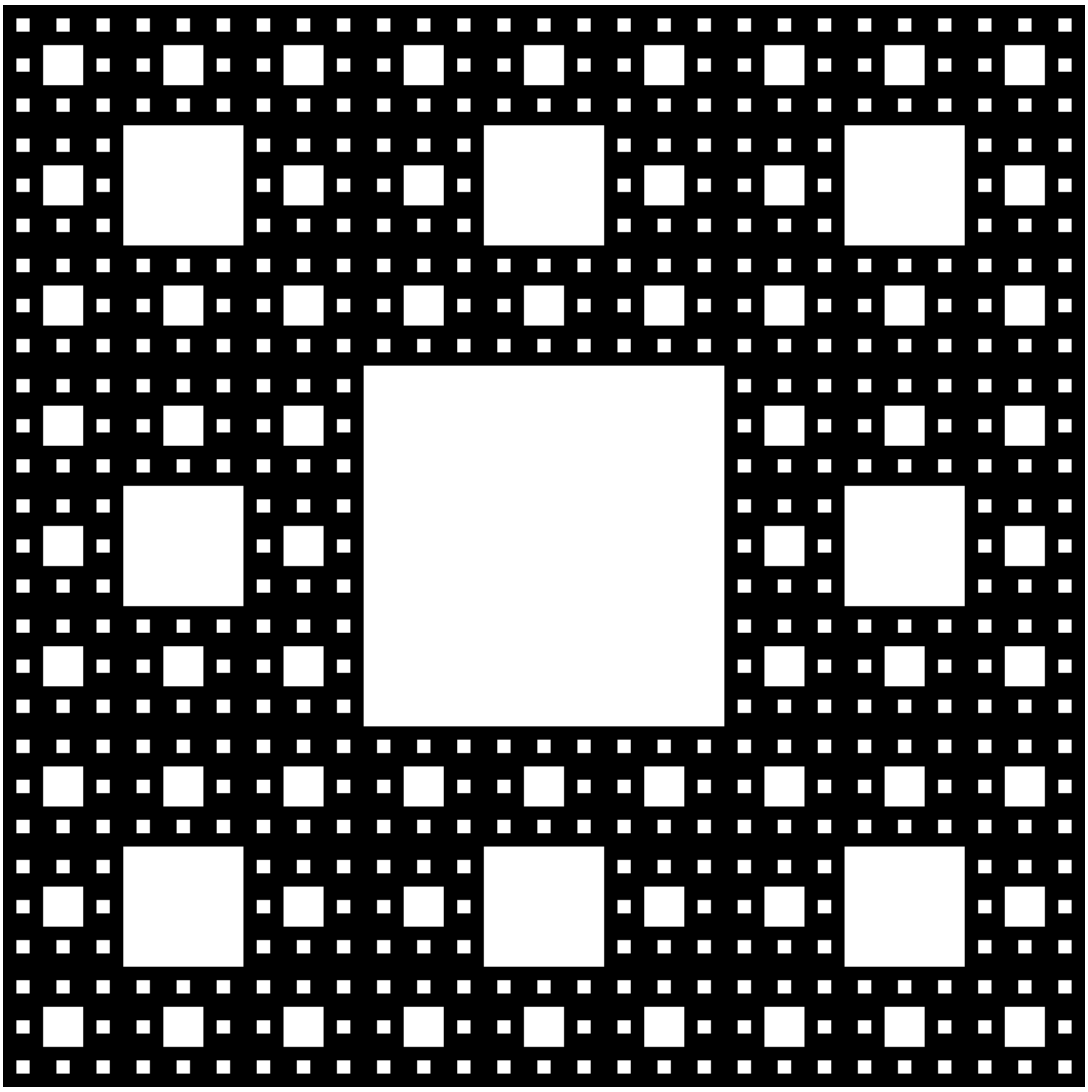


Notez qu'un triangle de Sierpiński est composé de 3 triangles de Sierpiński plus petits, ou, dans le cas de base, simplement de triangles normaux.

Indice : Un unique appel récursif est nécessaire ; son résultat peut être utilisé plusieurs fois.

3. Tapis de Sierpiński

De manière similaire à l'exercice précédent, concevez un programme qui produit la forme ci-dessous, appelée *tapis de Sierpiński*.



Un tapis de Sierpinski est soit un carré plein, soit composé de 8 tapis de Sierpiński plus petits avec un carré vide au centre.

4. Jeu de Nim

Dans cet exercice, nous allons nous intéresser à la résolution d'un jeu de stratégie de la famille des *jeux de Nim*.

Dans le jeu que nous allons considérer, deux joueurs s'affrontent autour de *n* allumettes. Chacun leur tour, chaque joueur doit retirer 1, 2 ou 3 allumettes. Le joueur qui arrive à retirer la dernière allumette est déclaré gagnant du jeu.

Pour cet exercice, nous allons essayer de déterminer la façon optimale de jouer à ce jeu à l'aide d'un programme Python.

Pour cela, nous allons concevoir une fonction appelée `winning_move`. La fonction a un unique paramètre `n` qui représente le nombre d'allumettes encore en jeu. La fonction devra retourner le nombre d'allumettes à retirer (entre 1 et 3) afin d'être assuré de gagner, et `None` s'il n'y a aucun coup qui ne garantit une victoire.

La fonction peut s'implémenter de façon récursive. Réfléchissez aux cas de base et aux cas récursifs. Utilisez les observations suivantes :

1. S'il n'y a pas de coup gagnant après avoir retiré un nombre k d'allumettes (avec k entre 1 et 3), alors k est le nombre d'allumettes qu'il faut retirer. En effet, en retirant k allumettes l'adversaire se trouvera dans une situation où il n'a pas de coup gagnant.
2. Au contraire, si peu importe le nombre d'allumettes retirées il existe toujours un coup gagnant (pour l'adversaire), alors il n'y a pas de coup gagnant pour le joueur.

Une fois votre fonction implémentée, examinez les résultats qu'elle donne pour un nombre d'allumettes entre 1 et 20. Pouvez-vous identifier un motif ? Est-il possible de donner une implémentation plus simple de cette fonction ?

5. Système de fichiers

Pour cet exercice, vous allez concevoir un programme pour analyser le contenu de dossier de l'ordinateur. Un exemple d'interaction avec le programme est donné ci-dessous.

```
Entrez le chemin du répertoire à analyser: /Users/romainedelmann/Desktop
Taille totale:  258113004
Nombre total de fichiers:  1769
Poids moyen des fichiers:  145908.99039005087
```

Un fichier de base vous est fourni pour cet exercice. Vous y trouverez une fonction déjà définie qui permet de prendre un chemin de fichier et d'en obtenir le contenu sous la forme d'un arbre. L'arbre est encodé de la façon suivante :

1. Les fichiers sont représentés par des dictionnaires de la forme suivante :

   ```
   {'type': 'file', 'name': 'woaw.png', 'size': 3078}
   ```

   Chaque fichier contient comme information un nom et une taille (en octets).

2. Les dossiers sont représentés eux aussi par des dictionnaires, mais avec une forme légèrement différente :

   ```
   {'type': 'directory', 'name': 'out', 'contents': […]}
   ```

   Le champ `contents` contient une liste de fichiers ou de dossiers. Il s'agit donc d'une structure récursive.

Votre but dans cet exercice est d'implémenter les fonctions `get_file_count` et `get_size`.

- La fonction `get_file_count` prend un arbre en argument et doit retourner le nombre total de fichiers contenus dans l'arbre.
- La fonction `get_size` doit retourner le poids total (en octets) des fichiers contenus dans l'arbre.

Ces fonctions sont à implémenter de manière récursive.

6. Calculatrice en notation polonaise

Dans cet exercice, vous allez implémenter en Python une petite calculatrice capable de faire des opérations arithmétiques simples. La calculatrice prendra en entrée des calculs comme 3 + 4 et donnera le résultat du calcul, ici par exemple 7.

Bien qu'elle puisse paraître simple de premier abord, la tâche est relativement compliquée. En effet, l'expression à calculer est donnée **sous forme de texte**. Il s'agit donc de prendre des chaînes de caractères qui représente un calcul et d'en donner le résultat.

En plus de cela, la calculatrice ne devra cependant pas se limiter à des calculs avec une unique opération. Elle devra accepter des expressions à la structure compliquée, comme 3 × 2 + 5 × (2 + 1).

Gérer la priorité des opérateurs et les parenthèses deviendrait vite un casse-tête… Pour cette raison, notre calculatrice va utiliser une notation différente de la notation usuelle des expressions : la **notation polonaise**. Ainsi, dans cette notation, l'expression 3 + 4 se note :

+ 3 4

Alors que l'expression (1 + 2) × (3 + 4) se note :

× + 1 2 + 3 4

À la différence de la notation usuelle, qui place les opérateurs (comme + ou -) entre les opérandes, cette notation place les opérateurs directement avant les opérandes. Cette notation a l'avantage de ne pas nécessiter de parenthèses, ni de niveau de priorité pour les opérateurs.

Vous trouverez sur Moodle un fichier à compléter afin de réaliser votre calculatrice.

Pour aller plus loin : Comme idée d'amélioration, vous pourriez incorporer dans votre calculatrice d'autres opérations : la division, l'exponentiation ou encore la factorielle. Vous pourriez aussi y incorporer des constantes comme π ou e.