Attention
Respectez les noms et les signatures des fonctions indiqués dans les exercices.

Première partie : Flots

Exercice 1

La suite $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \cdots$ converge vers $\ln(2) \approx 0.693147180559945309$.

1. Définissez un flot formé des éléments de cette suite, en utilisant le même principe que celui vu au cours pour la suite convergeant vers π.
2. Appliquez la technique d'Euler pour accélérer la convergence.
3. Appliquez ensuite la technique des tableaux.

Mesurez combien d'itérations accélérées (par technique d'Euler et par tableaux) sont nécessaires pour que la valeur estimée soit égale à la valeur convergente jusqu'à la 10^{e} décimale. Mettez le résultat comme commentaire dans vôtre solution.

Ensuite, estimez et notez le nombre d'itérations nécessaires pour obtenir cette précision avec la série non-accélérée.

Exercice 2

1. Définissez `fromTo` qui retourne le flot de tous les entiers n tels que $l \leq n \leq u$.
   ```
   def fromTo(l: Int, u: Int): Stream[Int] = ...
   ```
2. Définissez `intPairs` qui retourne le flot de toutes les pairs d'entiers de la forme (i, j) tels que $i \geq 0 \wedge j \geq 0$ à l'aide de compréhensions for.
   ```
   def intPairs: Stream[(Int, Int)] = ...
   ```
3. Définissez une fonction équivalente à `intPairs` sans utiliser de compréhension for.

Exercice 3

Considérez le flot $(1), (1, 1), (2, 1), (1, 2, 1, 1), (1, 1, 1, 2, 2, 1), (3, 1, 2, 2, 1, 1), \ldots$ dont les éléments sont des flots d'entiers. Quel est le prochain élément de ce flot ? Remarquez que chaque nouvel élément contient certains entiers de l'élément précédent ainsi que des compteurs.

1. Ecrivez `countStream` qui prend un élément en argument et calcule le prochain élément du flot. Par exemple `countStream(cons(2, cons(1, empty)))` vaut :
   ```
   cons(1, cons(2, cons(1, cons(1, empty))))
   ```
   La fonction a la signature suivante.
   ```
   def countStream(s: Stream[Int]): Stream[Int] = ...
   ```
2. Ecrivez `allCountStreams` qui retourne le flot ci-dessus, en vous servant de `countStream`.
   ```
   def allCountStreams: Stream[Stream[Int]] = ...
   ```

Seconde partie : Résolution de Contraintes

Le système de résolution de contraintes du cours est composé de *quantités* et de *contraintes*.

- Une quantité contient une valeur, définie ou non, et relie un certain nombre de contraintes. Les quantités sont les arêtes d'un réseau de contraintes.
- Une contrainte lie plusieurs quantités par une équation simple. Les contraintes sont les nœuds d'un réseau de contraintes.

Une contrainte d'addition modélise une équation de la forme $a + b = c$ où a, b et c sont des quantités.

Exercice 1

Complétez l'implantation du système de contraintes.

- Ajoutez une contrainte de multiplication modélisant une équation de la forme $a \times b = c$; notez que $0 \times x = x \times 0 = 0$ quel que soit x (même si x est indéfini).
- Ajoutez une contrainte d'égalité modélisant une équation de la forme $a = b$.

Pour chaque nouvelles contraintes, ajoutez un opérateur à la classe `Quantity` qui permette l'écriture agréable de contraintes, en vous inspirant de l'opérateur `+` déjà défini. Nommez l'opérateur pour les contraintes de multiplication $*$, et celui pour les contraintes d'égalité `===`.

Exercice 2

Est-il possible d'implanter une équation d'élévation au carré de la forme $a^2 = b$ au moyen de votre contrainte de multiplication ? Si oui, implantez-la. Si non, expliquez pourquoi et implantez une solution qui n'utilise pas d'autres contraintes.

A l'aide de la contrainte d'élévation au carré, ajoutez les deux opérateurs `square` et `sqrt` à la classe `Quantity`.