Mathématiques pratiques
Propriétés des opérations

Dans ce travail pratique, on va parler de *lois de composition interne*. L'objectif est d'explorer (et comprendre) les propriétés des opérations dans le cadre de l'arithmétique modulaire et des applications entre ensembles finis.

Spoiler

Les définitions impliquées sont **importantes** et certainement pas triviales. Prenez le temps de réfléchir avec un papier et un crayon.

Exercice 1 : arithmétique modulaire

Soit $n \in \mathbb{N}^*$. On découpe $\mathbb{Z}$ selon les restes possibles dans la division euclidienne par n. On définit la *classe d'équivalence* de $r \in [\![0, n-1]\!]$ par :

$$\overline{r} = \{p \in \mathbb{Z} \text{ tel que } p \equiv r[n]\}$$

Cette partie de $\mathbb{Z}$ contient les entiers dont le reste dans la division euclidienne par n est r. Le représentant d'une classe n'est pas unique, avec par exemple :

$$\overline{1} = \overline{n+1} = \overline{2n+1} = \overline{1-n} = \dots$$

On définit deux opérations sur les classes :

$$\forall (p,q) \in \mathbb{Z}^2, \quad \begin{cases} \overline{p} + \overline{q} = \overline{p+q} \\ \overline{p} \times \overline{q} = \overline{p \times q} \end{cases}$$

Histoire d'horloge

L'ensemble $\mathbb{Z}/12\mathbb{Z}$ peut se comprendre en jouant avec la petite aiguille d'une horloge analogique.

→ Si l'aiguille des heures est sur 2 et que 13 heures passent, l'aiguille des heures est sur 3. Par le calcul :

$$\overline{2} + \overline{13} = \overline{15} = \overline{3}$$

→ Si l'aiguille des heures est sur midi et qu'elle se déplace de trois heures en trois heures, elle est sur 9 au bout de 7 étapes (ou 19, 31, 43 et ainsi de suite). Par le calcul :

$$\overline{3} \times \overline{7} = \overline{21} = \overline{9}$$

→ Si l'aiguille des heures est sur midi et qu'elle se déplace de cinq heures en cinq heures, il est possible d'atterrir sur 1 au bout de 5 étapes (ou 17, 29, 41 et ainsi de suite). Par le calcul :

$$\overline{5} \times \overline{5} = \overline{25} = 1$$

On dit que $\overline{5}$ est la classe **inverse** de $\overline{5}$ et on note $\overline{5}^{-1} = \overline{5}$.

Votre mission

Vous disposez d'une horloge un peu particulière : le *pas* de l'aiguille des minutes est réglable. Par exemple, avec un pas de 4, l'aiguille initialement sur 0 se retrouve en 4 après une minute, en 8 après deux minutes et ainsi de suite.

Déterminer les pas qui permettent d'avoir (au bout d'un nombre d'étapes à préciser) l'aiguille des minutes sur 1 sachant qu'elle était initialement sur 0. Par exemple :

- → 7 est un pas possible. On atterrit sur 1 au bout de 43 étapes.
- → 6 n'est pas un pas possible. On ne pourra jamais finir sur 1 en partant de 0 et en sautant de 6 minutes en 6 minutes.

Sauriez-vous dire quelle est la particularité des nombres trouvés ?

Exercice 2 : autour d'applications

On s'intéresse maintenant à des applications entre ensembles finis, que l'on peut implémenter à l'aide de tableaux associatifs appelés *dictionnaires* en python. Un exemple d'application de $\{a, b, c, d\}$ vers $[\![1, 7]\!]$ est :

```
f = {"a": 1, "b":2, "c":7, "d":3} # create a dictionnary
```

À chaque *champ* (élément de l'espace de départ, type `string`) est *associé* une *valeur* (élément de l'espace d'arrivée, pas de type imposé). Pour évaluer la fonction `f` sur un élément du domaine, on utilise l'opération *crochets*.

```
f["b"]  # should print the value associated to field "b"
```

Dans le fichier `README`, vous trouverez un utilitaire `two_line_repr(f)` pour afficher sur deux lignes la description en extension d'une application `f`. Dans cette représentation, `y` en dessous de `x` signifie que `f(x) = y`. Par exemple :

```
two_line_repr(f)
```

devrait afficher :

```
[ a b c d ]
[ 1 2 7 3 ]
```

Partie 1 : composition

Soient (A, B, C) trois ensembles finis et $f : A \to B$ et $g : B \to C$ deux applications. On peut définir l'application composée $g \circ f$ par :

$$\forall a \in A,\ (g \circ f)(a) = g\left(f(a)\right)$$

avec $f(a)$ un élément de B donc dans le domaine de définition de g (ouf). Définir une fonction `comp(f,g)` qui prend en argument deux fonctions `f` et `g` données sous forme de dictionnaire et renvoie l'application composée. Une erreur est affichée si les fonctions ne sont pas composables. Vérifier sur quelques exemples que vous avez le fonctionnement attendu.

Partie 2 : générer les fonctions

Définir une fonction (récursive si vous le souhaitez) `all_func(n)` prenant en argument un entier `n` et renvoyant une liste de toutes les fonctions de $[\![1, n]\!]$ dans lui-même. En vous rappelant votre cours sur le dénombrement, vérifier que vous obtenez le bon nombre d'applications.

Partie 3 : associativité

Soient quatre ensembles A, B, C, D et trois applications $f : A \to B$, $g : B \to C$ et $h : C \to D$. La composition d'applications est une opération **associative** c'est-à-dire qu'il n'y a pas d'ambiguïtés pour calculer la composée $h \circ g \circ f$. Faire $h \circ (g \circ f)$ ou $(h \circ g) \circ f$ (qui sont les seules possibilités de calcul de $h \circ g \circ f$) donne le même résultat.

On note $\mathcal{F}_n$ l'ensemble des applications de $[\![1, n]\!]$ dans $[\![1, n]\!]$.

1. Vérifier de manière brute l'associativité de la composition sur l'intégralité de $\mathcal{F}_3$. Même question pour $\mathcal{F}_4$.
2. En vous basant sur le temps d'exécution pour tester l'associativité de $\circ$ dans $\mathcal{F}_4$, estimer le temps que prendrait la vérification brute de l'associativité de $\circ$ dans $\mathcal{F}_5$. (Pour chronométrer l'éxécution d'une cellule, il suffit d'ajouter `%%time` sur la première ligne)

Partie 4 : applications symétrisables

Une application *symétrisable* est une application **bijective** c'est-à-dire qui peut se lire sans ambiguïtés dans les deux sens. Une application finie bijective est **nécessairement** (mais pas suffisamment) une application entre deux ensembles de même cardinal. Déterminer par force brute le nombre d'éléments de $\mathcal{F}_n$ qui sont symétrisables pour $n \geq 5$. Le résultat n'était-il pas prédictible ?

Partie 5 : applications qui commutent

Soit $(f, g) \in \mathcal{F}_n^2$. On dit que f et g commutent si et seulement si $f \circ g = g \circ f$. Déterminer le nombre de paires $(f, g) \in \mathcal{F}_n^2$ qui commutent pour $n \geq 5$. En utilisant `https://oeis.org/`, donner le nombres de paires dans le cas $n = 6$.

Partie 6 : orbites

On définit *l'orbite* de l'élément $x \in [\![1, n]\!]$ **par** l'application $f \in \mathcal{F}_n$ l'ensemble des itérés :

$$\boxed{\text{orbite}_f(x) = \{x,\ f(x),\ (f \circ f)(x), (f \circ f \circ f)(x), \ldots\}}$$

Important : l'ensemble d'arrivée de f étant fini, la suite des valeurs $(f^n(x))_{n \geq 1}$ finit nécessairement par boucler. On a du coup :

$$\boxed{\forall (x, f) \in [\![1, n]\!] \times \mathcal{F}_n : \quad |\text{orbite}_f(x)| \in [\![1, n+1]\!]}$$

1. Définir une fonction `orb(f,x)` renvoyant l'orbite de `x` par `f` sous forme de liste.
2. Évaluer la taille moyenne de l'orbite d'une fonction $f \in \mathcal{F}_7$ grâce à la formule :

$$\frac{1}{|\mathcal{F}_7|} \sum_{f \in \mathcal{F}_7} \left[\sum_{x=1}^{7} \frac{|\text{orbite}_f(x)|}{7} \right]$$

Exercice bonus : retour à l'arithmétique modulaire

Une manière de générer des nombres pseudo-aléatoires est d'utiliser un générateur congruentiel linéaire.

1. on se fixe un entier naturel non nul n appelé le **module**
2. on se donne un entier naturel x_0 appelé **graine**
3. on se donne deux entiers naturels (a, b) appelés **paramètres**
4. on construit la suite (x_k) par récurrence selon :

$$\boxed{x_{k+1} = ax_k + b \bmod n}$$

qui se lit :

$$\boxed{x_{k+1} \text{ est le reste dans la division euclidienne par } n \text{ de } ax_k + b}$$

C'est une manière de dire que l'on travaille avec les représentants principaux (ceux dans $[\![0, n-1]\!]$) dans la structure algébrique $(\mathbb{Z}/n\mathbb{Z}, +, \times)$. En python, l'opération mod est `%`.

Exponentation modulaire

La commande `pow` de sage permet de faire de l'exponentiation modulaire. Ci-dessous, on vérifie que $3 = 3^3 \bmod 12$ et $5 = 5^{-1} \bmod 12$ que l'on traduirait dans $\mathbb{Z}/12\mathbb{Z}$ par $\overline{3}^3 = \overline{27} = \overline{3}$ et $\overline{5}^{-1} = \overline{5}$.

```
pow(3,3, mod=12) # print 3
pow(5,-1, mod=12) # print 5
```

Suite arithmético-géométrique

On cherche une définition **explicite** de la suite (x_k). Pour cela :

— on détermine le point fixe de la fonction $f : x \mapsto ax + b \bmod n$. Il s'agit de $x_p \in [\![0, n-1]\!]$ tel que $x_p = f(x_p) \bmod n$. On le calcule selon :

$$\boxed{x_p = -b(a-1)^{-1} \bmod n}$$

Ici, $(a-1)^{-1}$ est l'inverse modulo n de $(a-1)$ c'est-à-dire que $(a-1)(a-1)^{-1} = 1 \bmod n$.

— On pose (v_k) la suite de terme général $v_k = v_k - x_p$. La relation $x_{k+1} = ax_k + b \bmod n$ donne :

$$\boxed{\forall k \in \mathbb{N},\ v_{k+1} = av_k \bmod n}$$

— La suite (v_k) est géométrique de raison a donc :

$$\forall k \in \mathbb{N},\ v_k = a^k v_0$$

soit :

$$\boxed{\forall k \in \mathbb{N},\ x_k = a^k(x_0 - x_p) + x_p \bmod n}$$

Votre mission

Soient trois nombres générés consécutivement selon le générateur de module $n = 21247$:

$$x_1 = 1936 \qquad x_2 = 12586 \qquad x_3 = 8850$$

Retrouver la graine x_0 et les paramètres (a, b) du générateur.