3. Ajouter une méthode `__lt__` qui permet de déterminer si une date `d1` est antérieure à une date `d2` en écrivant `d1 < d2`. La tester.

ACTIVITÉ 5

Dans certains langages de programmation les tableaux ne sont pas nécessairement indexés à partir de `0`. Par exemple, on peut déclarer un tableau dont les indices vont de `-10` à `9` si on le souhaite. Dans cet exercice, on se propose de construire une classe `Tableau` pour réaliser de tels tableaux. Un objet de cette classe aura deux attributs, un attribut `premier` qui est la valeur de premier indice et un attribut `contenu` qui est un tableau Python contenant les éléments. Ce dernier est un vrai tableau Python, indexé à partir de 0.

Écrire un constructeur `__init__(self, imin, imax, v)` où `imin` est le premier indice, `imax` le dernier indice et `v` la valeur utilisée pour initialiser toutes les cases du tableau. Ainsi, on peut écrire `t = Tableau(-10, 9, 42)` pour construire un tableau de vingt cases, indexées de `-10` à `9` et toutes initialisées avec la valeur `42`.

Écrire une méthode `__len__(self)` qui renvoie la taille du tableau.

Écrire une méthode `__getitem__(self, i)` qui renvoie l'élément du tableau self d'indice `i`. De même, écrire une méthode `__setitem__(self, i, v)` qui modifie l'élément du tableau `self` d'indice `i` pour lui donner la valeur `v`. Ces deux méthodes doivent vérifier que l'indice `i` est bien valide et, dans le cas contraire, lever l'exception `IndexError` avec la valeur de `i` en argument (c'est-à-dire `raise IndexError (i)`).

Enfin, écrire une méthode `__str__(self)` qui renvoie une chaîne de caractères décrivant le contenu du tableau.

https://pa.dilla.fr/19

ACTIVITÉ 1

Définir une classe `Fraction` pour représenter un nombre rationnel. Cette classe possède deux attributs, `num` et `denom`, qui sont des entiers et désignent respectivement le numérateur et le dénominateur. On demande que le dénominateur soit plus particulièrement un entier strictement positif.

1. Écrire le constructeur de cette classe. Le constructeur doit lever une `ValueError` si le dénominateur fourni n'est pas strictement positif.
2. Ajouter une méthode `__str__` qui renvoie une chaîne de caractères de la forme `"12 / 35"`, ou simplement de la forme `"12"` lorsque le dénominateur vaut 1.
3. Ajouter des méthodes `__eq__` et `__lt__` qui reçoivent une deuxième fraction en argument et renvoient `True` si la première fraction représente respectivement un nombre égal ou un nombre strictement inférieur à la deuxième fraction.
4. Ajouter des méthodes `__add__` et `__mul__` qui reçoivent une deuxième fraction en argument et renvoient une nouvelle fraction représentant respectivement la somme et le produit des deux fractions.
5. Tester ces opérations.
6. (bonus) S'assurer que les fractions sont toujours sous forme réduite.

ACTIVITÉ 2

Définir une classe `Intervalle` représentant des intervalles de nombres. Cette classe possède deux attributs `a` et `b` représentant respectivement l'extrémité inférieure et l'extrémité supérieure de l'intervalle. Les deux extrémités sont considérées comme incluses dans l'intervalle. Tout intervalle avec `b < a` représente l'intervalle vide.

1. Écrire le constructeur de la classe `Intervalle` et une méthode `est_ vide` renvoyant `True` si l'objet représente l'intervalle vide et `False` sinon.
2. Ajouter des méthodes `__len__` renvoyant la longueur de l'intervalle (l'intervalle vide a une longueur `0`) et `__contains__` testant l'appartenance d'un élément `x` à l'intervalle.
3. Ajouter une méthode `__eq__` permettant de tester l'égalité de deux intervalles avec `==` et une méthode `__le__` permettant de tester l'inclusion d'un intervalle dans un autre avec `<=`. Attention : toutes les représentations de l'intervalle vide doivent être considérées égales, et incluses dans tout intervalle.
4. Ajouter des méthodes intersection et union calculant respectivement l'intersection de deux intervalles et le plus petit intervalle contenant l'union de deux intervalles (l'intersection est bien toujours un intervalle, alors que l'union ne l'est pas forcément). Ces deux fonctions doivent renvoyer un nouvel intervalle sans modifier leurs paramètres.
5. Tester ces méthodes.

ACTIVITÉ 3

Définir une classe `Angle` pour représenter un angle en degrés. Cette classe contient un unique attribut, `angle`, qui est un entier. On demande que, quoiqu'il arrive, l'égalité $0 \leq$ `angle` < 360 reste vérifiée.

1. Écrire le constructeur de cette classe.
2. Ajouter une méthode `__str __` qui renvoie une chaîne de caractères de la forme `"60 degrés"`. Observer son effet en construisant un objet de la classe `Angle` puis en l'affichant avec `print`.
3. Ajouter une méthode `ajoute` qui reçoit un autre angle en argument (un objet de la classe `Angle`) et l'ajoute au champ `angle` de l'objet. Attention à ce que la valeur d'`angle` reste bien dans le bon intervalle.
4. Ajouter deux méthodes `cos` et `sin` pour calculer respectivement le cosinus et le sinus de l'angle. On utilisera pour cela les fonctions `cos` et `sin` de la bibliothèque `math`. Attention : il faut convertir l'angle en radians (en le multipliant par $\pi/180$) avant d'appeler les fonctions `cos` et `sin`.
5. Tester les méthodes `ajoute`, `cos` et `sin`.

ACTIVITÉ 4

Définir une classe `Date` pour représenter une date, avec trois attributs `jour`, `mois` et `annee`.

1. Écrire son constructeur.
2. Ajouter une méthode `__str __` qui renvoie une chaîne de caractères de la forme `"8 mai 1945"`. On pourra se servir d'un attribut de classe qui est un tableau donnant les noms des douze mois de l'année. Tester en construisant des objets de la classe `Date` puis en les affichant avec `print`.