ee et/rl.sllib.sq\\:sqttd

octobre 2021



X STRUCT. DE DONNÉES

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

une ValueError si le dénominateur fourni n'est pas strictement 1. Écrire le constructeur de cette classe. Le constructeur doit lever

- de la forme "12 / 35", ou simplement de la forme "12" lorsque 2. Ajouter une méthode __atr__ qui renvoie une chaîne de caractères .fifizoq
- fraction représente respectivement un nombre égal ou un nombre deuxième fraction en argument et renvoient True si la première 3. Ajouter des méthodes __eq__ et __tt__ qui reçoivent une le dénominateur vaut 1.
- représentant respectivement la somme et le produit des deux fracdeuxième fraction en argument et renvoient une nouvelle fraction 4. Ajouter des méthodes __add__ et __mul__ qui reçoivent une strictement inférieur à la deuxième fraction.
- 5. Tester ces opérations.

tions.

.etiub 6. (bonus) S'assurer que les fractions sont toujours sous forme ré-

```
()bomtaet
                               return str(self.contenu)
                                   [10, 10, 10, 10, 10]
                                           (1) Ining <<<
                           (01'6'g) x \ni puInp \ni I q pI = 1 <<<
                                         def __str__(self):
                              v = [soibni]unstnoo.lls
                              indice = i - self.premier
                               raise IndexError (i)
:(llea)nel + reimerq.llea =< i ro reimerq.llea > i li
                                                      27
                                               [6-]7 <<<
                                              [01-]7 <<<
                                        21 = [01-17 <<<
                      (S$ '6 '01-)xspuInvsIdvI = $ <<<
                              def __setitem__(self, i, v):
    """Exemples:
                            [ebibni]unetnob.llea nuuter
                              indice = i + self.premier
```

```
[]: from doctest import testmod
     class Fraction:
         def __init__(self, num, denom):
             """Nombre rationnel définit par la fraction
             num/denom avec denom > 0
                num (int): numérateur
                 denom (int): dénominateur > 0
             >>> nb = Fraction(1,2)
             >>> print(nb.num)
             >>> print (nb.denom)
             if denom <= 0:
                raise ValueError ("le dénominateur doit être strictement positif")
             self.num = num
             self.denom = denom
         def __str__(self):
             """Affichage de la classe Fraction
             >>> print(Fraction(1,2))
             >>> print (Fraction (5,1))
             if self.denom == 1:
                return str(self.num)
             texte = str(self.num) + " / " + str(self.denom)
             return texte
         def __eq__ (self, frac):
             """Êst ce que le nombre rationnel est égal à frac?
             Aras:
                 frac (Fraction): nombre rationnel à comparer
             >>> Fraction(1,2) == Fraction(1,2)
             >>> Fraction(1.2) == Fraction (2.1)
             >>> Fraction(1.2) == Fraction(5.10)
             True
             if self.denom == frac.denom:
                return self.num == frac.num
```

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.

```
[]: from doctest import testmod
    class TableauIndex():
        def __init__(self, imin, imax, v):
            """Tableau d'indices quelconque.
               imin (int): indice de début
                imax (int): indice de fin de tableau
               v (object): valeur initiale de toutes les cellules du tableau
            Exemples:
            >>> t = TableauIndex(-10, 9, 42)
            self.premier = imin
            longueur = imax - imin + 1
            self.contenu = [v] * longueur
        def __len__(self):
            \Rightarrow t = TableauIndex(-10, 9, 42)
            >>> len(t)
            return len(self.contenu)
        {\tt def} \ \_\_{\tt getitem}\_\_({\tt self}, \ i):
             """Exemples:
            >>> t = TableauIndex(-10, 9, 42)
            >>> t[-10]
            if i < self.premier or i >= self.premier + len(self):
               raise IndexError (i)
```

méthode du produit en croix pour tester égalité
l'avantage est de ne comparer que des nombres entiers

return self.num * frac.denom == self.denom * frac.num

et pas des float

X STRUCT. DE DONNÉES

```
testmod()
                                return Fraction(num, denom)
       num = self.num * trac.denom + trac.num * self.denom
                            denom = self.denom * frac.denom
      (S,E) noitionT == (00,00) noitionT + (S,E)
      def __add__(self, frac):
                                return Fraction(num, denom)
                            moneb.srl * moneb.lles = moneb
                                  mum.self.num * frac.num
        (3,1) noitonal == (3,5) noitonal * (2,1) noitonal <<<
     Fraction: nb rationnel produit des deux fractions
frac (Fraction): nb rationnel avec lequel on multiplie
                           def __mul__(self, frac):
""nésultat du produit par frac
      return self.num * frac.denom > self.denom * frac.num
      (thoil sai sup dq ab sniom) sraitna'b noithluginum #
          methode de comparaison par le produit en croix
                             return self.num < frac.num
                          >>> Fraction (3,4) < Fraction (1,2)
                        \langle \uparrow 1,01 \rangle noitonT \langle \uparrow T \langle \uparrow T,01 \rangle \langle \uparrow T,01 \rangle
                          (7,4) noitonal > (7,2) noitonal <<<
                          (7,8) noitonal > (7,8) noitonal <<<
                        onnt à rusiritai is surT : lood
          Trac (Fraction): nombre rationnel à comparer
   ional à ausirèlai tes lennoiter exdmon el sup es tellum
                                        def __lt__(self, frac):
```

```
()bomfaet
                                        return False
                                     return True
                           : ruoj.elsb > ruoj.llea li
                                     return False
                           :siom.etsb < siom.iles li
                                     return True
                           :siom.etsb > siom.lles li
                         :eenns.alsb < eenns.lles li
                                     return True
                         :eenns.etsb > eenns.iles li
                                                ənaj
                 >>> Date(1,4,2003) < Date(1,5,2003)
                 >>> Date(1,1,2004) < Date(1,1,2006)
                 >>> Date(1,1,2006) < Date(1,1,2006)
                 (8761, 8, 7) sind > (8761, 8, 3) sind <<<
                 (8761,6,4) stad > (8761,8,8) stad <<<
bool: True ssi est strictement antérieure à date
```

ACTIVITÉ 5

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

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

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.

```
[]: class Intervalle:
    def __init__(self, debut, fin):
        self.a = debut
        self.b = fin

    def est_vide(self):
```

- 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.
- 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.

```
[]: # version avec des test unitaire
     from doctest import testmod
     class Date:
        mois_fr = [ None,
                'janvier', 'février', 'mars',
                 'avril', 'mai', 'juin',
                 'juillet', 'août', 'septembre',
                 'octobre', 'novembre', 'décembre']
        def __init__(self, j, m, a):
            self.jour = j
            self.mois = m
            self.annee = a
        def _str_(self):
             """Affiche une date en français.
             Par exemple '8 mai 1945'.
             Returns:
             >>> print(Date(8,5,1945))
             8 mai 1945
             >>> print(Date(5,6,1977))
             5 juin 1977
             date = str(self.jour) + " "
             date += Date.mois_fr[self.mois] + " "
             date += str(self.annee)
            return date
         def __lt__(self, date):
             """Est ce que la date est antérieure à date?
             Args:
                 date (Date)
             Returns:
```

```
()bomfaet
  ( (elgas.lles) asiber_.elgaA ) niz awter
                     nie troqmi dasm mort
         ( 3, () nis.(34) slgnA ) bnuor <<<
 signs'i sh saussm ol sh sunis :tooll
                                 :suanqəy
                      əlbur, ləpsnuis....
                               :(llea)mia leb
   ( (elgns.llea)asibsr_.slgaA )soo nurter
                     from math import cos
                                  IITOT.0
           (g, () sos. (d4) slgnA ) bnuor <<<
                                 : əįdwəxg
slyno'l sh saussm bl sh sunisos :thoil
                    ·əlbup, ləb sunisol"""
                               def cos(self):
                  return mesure * pi / 180
                       from math import pi
                                   8078.1
         (3, (00) nsibsr_.slenk ) bnuor <<<
                                 :əįdwəxg
              loat: valeur en radian
                                 :suintəy
        èrpsb ns rusibu :(tni) srussm
   nsibor ne èrgeb ne eruzem al titreunol
                         .nie to soo enab
 sésilitu sainilimus seenle ab shoatsh"""
                         def _radian(mesure):
                     return Angle (mesure)
          mesure = self.angle + obj.angle
```



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

```
return Intervalle(tmp_a, tmp_b)
         def __str__(self):
              return "[" + str(self.a) + "; " + str(self.b) + "]"
[]: # Tests des méthodes
     mon_inter = Intervalle(5,12)
     print("mon_inter.a:", mon_inter.a)
     print("mon_inter.b:", mon_inter.b)
     mon_inter2 = Intervalle(5, 3)
     print("mon_inter2.a:", mon_inter2.a)
     print("mon_inter2.b:", mon_inter2.b)
     print("mon_inter.est_vide():" , mon_inter.est_vide() )
     print("mon_inter2.est_vide():", mon_inter2.est_vide() )
     print("len(mon_inter):", len(mon_inter) )
print("len(mon_inter2):", len(mon_inter2) )
     print("mon_inter.__contains__(5):", mon_inter.__contains__(5))
     print("5 in mon_inter:", 5 in mon_inter)
     print( "3 in mon_inter:", 3 in mon_inter )
     print( "12.001 in mon_inter:", 12.001 in mon_inter )
     print ("mon_inter == mon_inter2:", mon_inter == mon_inter2)
     print("Intervalle(4,10) <= Intervalle(5,12):", Intervalle(4,10) <= Intervalle(5,12))
     print("[4,6] inters. [5,10] == [5,6]:", Intervalle(4,6).intersection(Intervalle(5,10)) == <math>[5,6]:"
      \hookrightarrowIntervalle(5,6))
     print("[4,6] reunion [9,10] == [9,10]:", Intervalle(4,6).reunion(Intervalle(9,10)) == _u
```

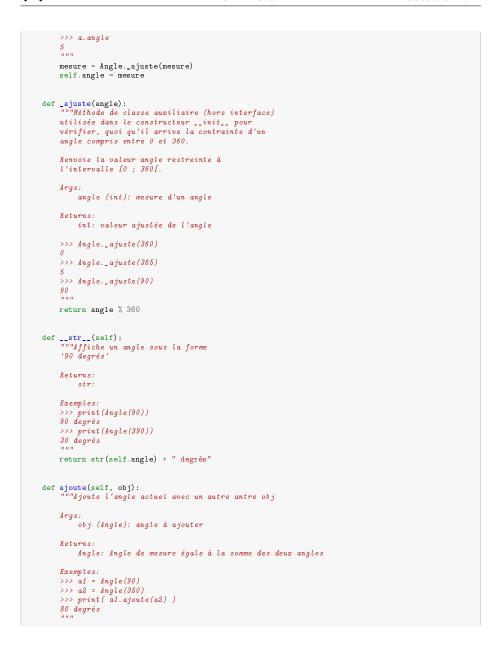
Voici ci-dessous une version du code précédent :

avec une documentation correcte

⇔8))

- avec des tests unitaires utilisant la bibliothèque doctest.

print("[4,6] reunion [5, 8] == [4, 8]:", Intervalle(4,6).reunion(Intervalle(5, 8)) == Intervalle(4, 0)



```
:()əbiv_taə.lləz li
                                               (SI, 3) sllauretal ni 9.4 <<<
                                                        (Si, 2) slinureini ni 3 <<<
                                                       (SI, 3) slintervalle (5, 12)
                                        [d, n] ni x iss surT : lood
                                                        endmon : (ini/inoll) x
                                                                      i[d,n] sllnurstni'l b
           def __contains__(self, x):
    """Est ce que la unleur x appartient
                                                                    return self.b - self.a
                                                                                                          return O
                                                                            : ()abiv_tas.llas li
                                                           ((E, 3) slinuratni) nsi <<<
                                                        ((S1, d) slinursini) nsi <<<
[d,b] sliburstni'i sh rusugnoi :ini
                                              def __len__(lles):
.def __len__(lles):
.def __len_len_len
.def __len_len
.def __len
                                                                   return self.b < self.a
                                                                       () sbiv_tes. Graini <<<
                                        () sbiv_tesi. astai <<<
                                        (S1,2) slintervall = Irstni <<<
                        obiv oliburotni is our :lood
                                                                                     .n => d tnob [d, n]
                         Sallaurodni nu obiu orobiznos ted
           """Est ce que l'intervalle est vide?
                                                                                              def est_vide(self):
                                                                                                                                                 13
                                                                                                               d.rotai <<<
```

3 - E0.0

Activité 3

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

- 1. Ecrire 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 inter-
- 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 \max th. 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.

```
# werston avec tests unitaires
# de toutes les méthodes
from dochest import bestmod

class Angle:

"""Angles compris entre 0 et 360.

def __init__(self, mesure):

"""Angle de mesure entière comprise entre 0 et 360.

krys:

mesure (ini): mesure de l'angle

scarpele:

comprise entre 0 et 360.

brangle:

comprise entre 0 et 360.

scarpele:

comprise entre (ini): mesure de l'angle

scarpele:

comprise entre (ini): mesure (ini): mesure de l'angle

scarpele:

comprise entre (ini): mesure (ini): mesure de l'angle

scarpele:

comprise entre (ini): mesure (ini): mesure de l'angle

scarpele:

comprise entre (ini): mesure (ini): mesure de l'angle

scarpele:

comprise entre (ini): mesure (ini): mesure entre (ini): mesure entre (ini): mesure (ini): mesure entre (ini): mesure (
```

```
return False
    if x > self.b or x < self.a:
       return False
    return True
def __eq__(self, inter):
    """Est ce que l'intervalle est
    égale à l'intervalle inter
    Args:
        inter2 (Intervalle): Intervalle à comparer
       bool: True ssi les deux intervalles ont les
        mêmes bornes
    >>> Intervalle(5,12) == Intervalle(5,7)
    >>> Intervalle(5,12) == Intervalle(5,12)
    >>> Intervalle(5,3) == Intervalle(7,2)
    True
    if self.est_vide() and inter.est_vide():
       return True
    if self.a == inter.a and self.b == inter.b:
       return True
    return False
def __le__(self, inter):
    """Est ce que l'intervalle est inclu
    dans l'intervalle inter?
    Args:
        inter (Intervalle): Intervalle [a,b]
       bool: True ssi l'intervalle est inclus dans inter
    Exemples:
    >>> Intervalle(4,10) <= Intervalle(5,12)
    >>> Intervalle(5,12) <= Intervalle(4,10)
    >>> Intervalle(6,10) <= Intervalle(5,12)
    True
    if self.est_vide():
       return True
    return self.a >= inter.a and self.b <= inter.b
def intersection(self, inter):
    """Intersection avec l'intervalle inter.
```

```
Returns:
            Intervalle: égal à l'intersection de self
           avec inter.
        >>> Intervalle(4,6).intersection(Intervalle(5,10)) == Intervalle(5,6)
       if self.est_vide():
           return self
       if inter.est_vide():
           return inter
        tmp_a = max(self.a, inter.a)
       tmp_b = min(self.b, inter.b)
       return Intervalle( tmp_a, tmp_b )
    def reunion(self, inter):
        """Réunion avec l'intervalle inter. Si la réunion
        n'est pas un intervalle, renvoie le plus petit
        des deux intervalles.
           inter (Intervalle)
        Returns:
            Intervalle: réunion avec inter
       >>> Intervalle(4,6).reunion(Intervalle(9,10)) == Intervalle(9,10)
        >>> Intervalle(4,6).reunion(Intervalle(5,8)) == Intervalle(4,8)
       True
       if self.est_vide():
           return inter
       if inter.est_vide():
           return self
        intersec = self.intersection(inter)
       if intersec.est_vide():
           if len(self) <= len(inter):
               return self
            else:
               return inter
        tmp_a = min(self.a, inter.a)
       tmp_b = max(self.b, inter.b)
       return Intervalle(tmp_a, tmp_b)
testmod()
```

inter (Intervalle)