Cours d'algorithmique et programmation 2

Anne Parrain

UFR des Sciences
Licence Sciences et Technologie
mentions Mathématiques et Informatique
Semestre 2

Section 1

Les fractions

Retour aux fractions. . .- À vous !

- Proposez les méthodes pour
  - retourner l'addition de deux fractions
  - retourner la soustraction de deux fractions
  - retourner l'inverse d'une fraction
  - retourner l'arrondi à un entier d'une fraction
  - retourner l'arrondi à un flottant d'une fraction

Correction - les fractions

```
1  # dans la classe Frac
2      def add(self,f) :
3          '''Frac, Frac -> Frac retourne le resultat de
4          l'addition de self et f.
5
6          >>> f1 = Frac(1,3)
7          >>> f2 = Frac(1,2)
8          >>> f3 = f1.add(f2)
9          >>> f3.affiche()
10         5/6
11         '''
12         numer = self.__num*f.__den+f.__num*self.__den
13         denom = self.__den*f.__den
14         return Frac(numerateur,denominateur)
```

Correction - les fractions (2)

```
1     def sub(self,f) :
2         '''Frac, Frac -> Frac
3         retourne le resultat de la soustraction
4         a self de f.
5
6         >>> f1 = Frac(1,3)
7         >>> f2 = Frac(1,2)
8         >>> f3 = f1.sub(f2)
9         >>> f3.affiche()
10        -1/6
11        >>> f3 = f2.sub(f1)
12        >>> f3.affiche()
13        1/6
14        '''
15        return self.add(f.mult_ent(-1))
```

Correction - les fractions (3)

```
1     def int(self) :
2         '''Frac -> int
3         retourne la partie entiere de la fraction.
4
5         >>> f = Frac(1,3)
6         >>> f.int()
7         0
8         >>> f = Frac(2,2)
9         >>> f.int()
10        1
11        >>> f = Frac(10,4)
12        >>> f.int()
13        2
14        '''
15        return self.__num//self.__den
```

Correction - les fractions (4)

```
1     def float(self) :
2         '''Frac -> float
3         retourne un arrondi de la fraction.
4
5         >>> f = Frac(1,3)
6         >>> f.float()
7         0.3333333333333333
8         >>> f = Frac(2,2)
9         >>> f.float()
10        1.0
11        >>> f = Frac(10,4)
12        >>> f.float()
13        2.5
14        '''
15        return self.__num/self.__den
```

Correction - les fractions (5)

```
1     def inverse(self) :
2         '''Frac -> Frac
3         retourne l'inverse de la fraction.
4
5         >>> f = Frac(1,3)
6         >>> f2 = f.inverse()
7         >>> f2.affiche()
8         3/1
9         '''
10        return Frac(self.__den,self.__num)
```

Utiliser des objets

- On peut maintenant manipuler des éléments du type `Frac` comme on le fait avec un type quelconque déjà connu
- excepté qu'on peut utiliser toutes les méthodes créées dans la classe `Frac` !

On veut écrire un programme pour un utilisateur qui doit faire des opérations sur des fractions. Il peut :

- saisir une ou plusieurs fractions
- afficher toutes les fractions en mémoire
- faire des opérations sur les fractions qu'il désigne. . .

Le début de l'application (1)

```
1  import fractions
2
3  ##################################
4  ## Fonctions
5  ##################################
6  def affiche_menu():
7      ''' -> None
8      affiche les actions possibles.
9      '''
10     print("Choisissez une action parmi :")
11     print("1 - creer une nouvelle fraction")
12     print("2 - creer plusieurs nouvelles fractions")
13     print("3 - afficher toutes les fractions")
14     print("4 - additionner deux fractions")
15     print("5 - soustraire deux fractions")
16     print("6 - inverser une fraction")
17     print("7 - multiplier deux fractions")
18     print("8 - multiplier une fraction par un entier")
19     print("9 - quitter l'application")
```

Le début de l'application (2)

```
1  def choisit_action() :
2      ''' -> int
3      retourne l'action choisie par l'utilisateur.
4      '''
5      affiche_menu()
6      rep = int(input("Quelle action choisissez-vous
7                       (entre 1 et 9) ? "))
8      while not (1 <= rep <= 9) :
9          affiche_menu()
10         rep = int(input("Quelle action choisissez-vous
11                      (entre 1 et 9) ? "))
12     return rep
```

Le début de l'application (3)

```
1  def main():
2      '''-> None
3      fct qui lance l'application manipulation
4      de fractions.
5      '''
6      lesFrac = []
7      action = choisit_action()
8      while action != 9 :
9          if action == 1 :
10             # nouvelle fraction
11             ...
12         ...
13         elif action == 8 :
14             ...
15         action = choisit_action()
16     ## fin du while - l'utilisateur quitte l'appli
17     print("Bye")
18
19 if __name__ == '__main__' :
20     main()
```

Les fonctions qui manquent - À vous !

Dans la fonction `main()`, on fait le choix de mémoriser les fractions manipulées dans la variable `lesFrac`.

Fonctions à spécifier et écrire :

- prévoir la saisie d'une fraction par l'utilisateur,
- prévoir la saisie de plusieurs fractions par l'utilisateur,
- afficher les fractions mémorisées dans une liste

Code à compléter dans la fonction `main` :

- cas `action == 1`
- cas `action == 4`

Section 2

Les classes mutables et non mutables

Classes non-mutables

La classe `Fraction` que nous venons d'écrire est non mutable :

- on ne peut pas accéder de l'extérieur aux attributs de la classe `Frac`
- aucune des méthodes de la classe `Frac` ne permet de modifier la valeur des attributs

Cela donne des résultats parfois étonnants :

```
>>> f = Frac(5,8)
>>> f.inverse()
>>> f.str()
5/8
```

Classes mutables

Une classe est mutable si elle permet, via ses méthodes, de modifier la valeur de ses attributs.

Par exemple, on pourrait imaginer une classe `FracMut` qui permettrait :

- d'inverser la fraction elle-même
- d'ajouter à la fraction elle-même une autre fraction
- idem pour les opérations de multiplication ou de soustraction

Une version mutable des fractions

```
1  class FracMut:
2      '''Classe mutable qui permet de modeliser
3      une fraction. On peut modifier la fraction
4      par inversion, addition, multiplication...
5      '''
6      # idem que Frac pour le constructeur,
7      # idem que Frac pour les methodes num, den, str,...
8
9      def inverse(self):
10         '''FracMut (modif) -> None
11         inverse la fraction.
12         '''
13         if self.__num != 0 :
14             self.__num, self.__den =
15                 self.__den, self.__num
```

Une version mutable des fractions

On aurait pu aussi faire ce choix :

```
1       def inverse(self):
2           '''FracMut (modif) -> FracMut
3           inverse la fraction.
4           '''
5           if self.__num != 0 :
6               self.__num, self.__den =
7                       self.__den, self.__num
8           return self
```

Une version mutable des fractions

ce qui permet de rester compatible dans l'utilisation.

```
>>> f = FracMut(5,2)
>>> f.inverse() # il n'est pas obligatoire de
                # recuperer le resultat
>>> f.str()
'2/5'
>>> f1 = f.inverse() # autre appel possible
       # attention on a fait une nouvelle inversion
>>> f1.str()
'5/2'
>>> f.str()
'5/2'
>>> id(f1)
140695997471600
>>> id(f)
140695997471600
```

Une version mutable des fractions

Ou bien, pour terminer sur les différentes possibilités :

```
1     def inverse(self):
2         '''FracMut (modif) -> boolean
3         inverse la fraction.
4         retourne True si l'inversion s'est bien passee
5         '''
6         if self.__num != 0 :
7             self.__num, self.__den =
8                   self.__den, self.__num
9             return True
10        return False
```

ce qui donne :

```
>>> f = FracMut(5,2)
>>> f.inverse()
True
>>> f.str()
'2/5'
```

```
>>> f = FracMut(0,2)
>>> f.inverse()
False
>>> f.str()
'0/2'
```

Une version mutable des fractions (3) - À vous !

- spécifier et écrire les méthodes `add`, `sub` et `mult` pour la classe `FracMut`

Classes mutables versus classes non-mutables - À vous !

À votre avis :

- Peut-on faire n'importe quoi avec la classe `FracMut` ? donnez un exemple qui montre le contrôle qu'elle effectue.
- Est-ce un problème de proposer les méthodes `num` et `den` ?

Classes mutables versus classes non-mutables

On considère une nouvelle proposition pour représenter une fraction :

```
1  class NewFrac:
2
3      def __init__(self,num,den):
4          # reduction de num et den par leur pgcd
5          self.__nd = [num,den]
6
7      def __inverse__(self):
8          '''NewFrac (modif) -> None'''
9          self.__nd[0], self.__nd[1] =
10             self.__nd[1], self.__nd[0]
11
12     # et toutes les methodes add, sub mult, str
13     # adaptees :
14     # - self.__nd[0] contient le numerateur
15     # - self.__nd[1] contient le denominateur
```

Classes mutables versus classes non-mutables - À vous !

```
1  # dans NewFrac
2
3      def num(self):
4          '''NewFrac -> int'''
5          return self.__nd[0]
6
7      def den(self):
8          '''NewFrac -> int'''
9          return self.__nd[1]
10
11     def num_den(self):
12         '''NewFrac -> list
13         retourne une liste de deux entiers
14         '''
15         return self.__nd
```

- Indiquez un gros problème de cette classe
- Proposez une ou plusieurs solutions pour y remédier