NAME

OperationsOnFractions : **algomaths** Perl 6 module in /maths/modules/pm6/**operations-on-fractions.pm6**

AUTHOR

https://github.com/Chrissealed/algomaths.git

VERSION

2019.01.04

Ce module est destiné à faire des opérations sur des fractions.

Il utilise les modules **ppcm.pm6**, **irreducible-fraction.pm6** et **prime-factors.pm6** et hérite du rôle **PrimeFactors**. La méthode principale de la classe **OperationsOnFractions** est **calculate-fractions(Str** *operation*) ∗ ∗*ouleparamtre* ∗ ∗**operation** peut être l'un des suivants :

- 'add-up' ou '+' ou 'a',
- 'subtract' ou '-' ou '−' ou 's',
- 'multiply' ou '*'ou'×'ou 'm',
- 'divide' ou ':' ou '÷' ou 'd',

ceci dans le cas ou il on opère que sur que deux fractions.

Elle renvoie une **paire** (Pair) comme valeur de retour. Les champs obligatoires à renseigner sont **nudepair1**, et **nudepair2**, correspondant aux numérateur et dénominateur de chaque fraction sur laquelle pratiquer l'opération choisie.

On peut justement se demander pourquoi l'implémentation des fractions se fait par l'emploi du type **Pair** plutôt que du type **Rat** (nombre rationnel) et que l'objet renvoyé par **calculate-fractions(Str $operation)** est une paire et non une fraction. Cette implémentation est en fait sujette à la multiplication des options pour le calcul des fractions; en effet les fractions littérales sont automatiquement réduites à leur plus simple expression ce qui oblitère la possibilité de choisir expressément la façon dont on veut effectuer les calculs. En fournissant les données sous cette forme, on a plus de contrôle sur le déroulement des calculs,

par exemple on a le choix de réduire ou pas les fractions en entrée en fournissant une valeur True aux booléens **reduce-fractionN** (Voir ci-dessous). De même on pourra choisir de livrer la dernière fraction résultante sans la réduire systématiquement. (Voir plus bas).

Le champ suivant est facultatif :

- **nudepair3**, pour lequel d'autres opérateurs sont disponibles pour l'**addition** et la **soustraction** uniquement, c'est-à-dire :
- 'add-upx2' ou '++' ou 'aa',
- 'subtractx2' ou '–' ou '−−' ou 'ss',
- 'add-up-subtract' ou '+-' ou '+−' ou 'as', et enfin
- 'subtract-add-up' ou '-+' ou '−+' ou 'sa'.

Ensuite, trois champs booléens facultatifs associés aux champs **nudepair1**, **nudepair2** et **nudepair3** servent à préciser si l'on effectue la réduction de la fraction en question avant le traitement des données,

- ce sont **reduce-fraction1**, **reduce-fraction2** et **reduce-fraction3**;

ces champs par défaut sont à **False**, autrement dit on ne tente pas d'opérer une réduction de chaque fraction avant d'effectuer l'opération. Noter que la dernière fraction résultante de l'opération est toujours donnée irréductible, mais qu'on peut jouer sur deux autres attributs Booléens :

- **reduce-last-once** (**False** par défaut), et **reduce-last-one** (**True** par défaut);

le premier visant dans le cadre de l'***addition*** ou de la ***soustraction*** à réduire une seule fois la première fraction au terme du calcul dans une liste chaînée d'opérations (par exemple : +−) lors du calcul de trois fractions, et le deuxième à réduire systématiquement la dernière fraction obtenue à tous les niveaux de cette liste ou bien lors d'une opération simple (+ ou −). Ces attributs visent à modifier les opérations de calcul de manière à produire plusieurs options pour générer un même résultat.

Six autres attributs, peuvent être utilisés lors d'une opération de ***multiplication*** ou de ***division***, pour une granularité maximale, ce sont :

- **breakdown-numerator1**, réduire en facteurs premiers le numérateur de la première fraction,
- **breakdown-numerator2**, réduire en facteurs premiers le numérateur de la deuxième fraction,
- **breakdown-numerator3**, réduire en facteurs premiers le numérateur de la troisième fraction facultative.
- **breakdown-denominator1**, réduire en facteurs premiers le dénominateur de la première fraction,

- **breakdown-denominator2**, réduire en facteurs premiers le dénominateur de la deuxième fraction,
- **breakdown-denominator3**, réduire en facteurs premiers le dénominateur de la troisième fraction facultative.

Ces six attributs ont par défaut la valeur **False**.

Viennent ensuite deux autres champs facultatifs utilisables pour la ***multiplication*** ou la ***division*** de fractions uniquement et qui jouent avec les six attributs décrits précédemment :

- **breakdown-numerators**
- **breakdown-denominators**

qui pour le premier réduit en facteurs premiers les numérateurs de chaque fraction et le deuxième réduit en facteurs premiers les dénominateurs de chaque fraction; ces attributs sont tous les deux à **True** par défaut.

Vous disposez ainsi de deux niveaux de contrôle, l'un global pour tous les numérateurs et/ou dénominateurs et l'autre pour un contrôle dont la granularité vous permet d'agir exactement sur les numérateurs et/ou dénominateurs de chaque fraction séparément.

Note: chaque fois que vous mettez à **True** l'un des six attributs granulaires, l'attribut global correspondant (c'est-à-dire **breakdown-numerators** et/ou **breakdown-denominators**) est automatiquement désactivé.

- L'attribut **compute-prime-factors**

enfin permet de supprimer les facteurs en double dans le numérateur et le dénominateur de deux fractions pour opérer leur réduction. Ce champ est à **True** par défaut lui aussi.

Les autres champs, facultatifs, sont destinés à choisir parmi les diverses méthodes utilisées pour le calcul des méthodes de classes appelées en interne :

- **which-ppcm-algorithm**, peut prendre l'une des valeurs suivantes :
- 'by-larger-number-multiples' ou 'b.l.n.m.' ou 'by-m' ou 'bm';
- 'by-prime-factors' ou 'b.p.f.' ou 'by-f' ou 'bf';
- 'by-use-of-pgcd' ou 'b.u.o.p.' ou 'by-p' ou 'bp' (utilisé par défaut).

Ce dernier attribut n'est **pas** utilisé pour la ***multiplication*** ou la ***division***, mais uniquement pour l'***addition*** ou la ***soustraction*** afin de choisir l'algorithme qui sera utilisé pour le calcul par la classe PPCM.

- **which-irreducible-fraction-algorithm** peut prendre les valeurs :
- 'euclide' ou 'e' ou ':' ou '÷' (utilisé par défaut);
- 'subtraction' ou 's' ou '-' ou '—';

- ‘factorization’ ou ‘f’ ou ’*‘ou’×’.

Cet attribut est destiné à choisir la méthode qui sera utilisée par la classe **IrreducibleFraction** pour la réduction d'une fraction à sa plus simple expression.

- **which-pgcd-algorithm** peut prendre l'une des valeurs :
- ‘euclide’ ou ‘e’ ou ‘:’ ou ‘÷’ (utilisé par défaut);
- ‘subtraction’ ou ‘s’ ou ‘-’ ou ‘−’;
- ‘factorization’ ou ‘f’ ou ’*‘ou’×’;
- ‘divisors-listing’ ou ‘d’ ou ‘#’ ou ‘/’.

Cet attribut peut être employé pour toutes les opérations; il détermine l'algorithme utilisé par la classe PGCD pour le calcul du PPCM.

La classe possède en outre un **attribut requis** destiné à écrire dans un fichier les informations qui apparaissent à l'écran et qui décrivent les différentes étapes des opérations laissant ainsi un *témoin* du calcul effectué. Celui-ci (pour info) est défini dans le rôle sous-jacent **PrimeFactors**. Il s'agit de **Teeput::Tput $t is required is rw** : il faut lui passer un objet de type **Teeput::Tput** du module **teeput.pm6**. Ses méthodes sont utilisées en remplacement des méthodes ***put*** (tput), ***print*** (tprint) ou ***say*** (tsay). Référez-vous à la doc de ce module pour plus d'informations.

Voici la liste des méthodes de la classe OperationsOnFractions :

calculate-fractions(Str:D $operation –> Pair:D)

Cette méthode est la principale de la classe, qui donne accès à toutes les autres méthodes — bien que celles-là ne soient pas privées et sont donc accessibles isolément — en employant un seul argument, le type d'opération à effectuer. (Voir plus haut)

are-they-prime(Int:D @numerators, Int:D @denominators –> Pair:D)

Cette méthode a pour but de déterminer si les numérateurs et les dénominateurs sont des facteurs premiers. Les arguments correspondant sont des tableaux de 2 ou 3 éléments. Si oui, les attributs de la classe en lecture seule **are-prime-nu** et/ou **are-prime-de** sont passés à **True**. La méthode renvoie une paire constituée de deux valeurs booléenes : la première composante correspondant aux numérateurs et la deuxième au dénominateurs.

reduce-fraction(Int:D $numerator, Int:D $denominator, Str $sign = ” –> Pair:D)

Cette méthode destinée à obtenir la fraction irréductible à partir du numérateur et du dénominateur passés en argument utilise la classe **IrreducibleFraction** du module **irreducible-fraction.pm6** et utilise deux attributs de la classe, soit **which-irreducible-fraction-algorithm** et **which-pgcd-algorithm** qui déterminent le choix pour le premier de l'algorithme de la méthode utilisée en interne par **IrreducibleFraction** pour effectuer les calculs, et pour le deuxième l'algorithme du module **pgdc.pm6** dont **IrreducibleFraction** se servira en interne. Pour gérer correctement les nombres négatifs avec l'opération **multiply** uniquement, il faut mettre le troisième argument **$sign** à '-' afin d'appeler l'une des méthodes

- **reduce-fraction-with-euclide-algorithm($sign)**;
- **reduce-fraction-with-subtraction-algorithm($sign)**
- **reduce-fraction-with-factorization-algorithm(**$sign) $**$; $Lesigne + n'estpascrit, ilestgrparlavaleurpardfautde$ $*$**sign**, une chaîne vide.

add-up(Pair:D $pair1, Pair:D $pair2, Int $times = 0 –> Pair:D)

Cette méthode renvoie l'addition des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1** et **nudepair2**. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe **reduce-fraction**(Int:D $numerator, Int:D $denominator, Str $sign = ” –> Pair:D) {} par l'entremise de laquelle sont appelées les méthodes du module **irreducible-fraction.pm6**.

Le paramètre **$times** s'il est mis à 1 et que l'attribut de la classe **reduce-last-on**_c_e**** est passé à **True**, la méthode ne réduira pas la fraction résultante lors d'un premier appel, mais la réduira lors d'un appel subséquent lors de l'effectuation d'un calcul sur trois fractions par la méthode **calculate-fractions**(Str:D $operation –> Pair:D) {} Elle renvoie une **paire** constituée du numérateur et du dénominateur de la fraction résultante.

add-upx2(Pair:D $pair1, Pair:D $pair2, Pair:D $pair3 –> Pair:D)

Cette méthode renvoie l'addition des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1**, **nudepair2** et **nudepair3**, pour calculer la somme de trois fractions. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe : **reduce-fraction($numerator,**

$denominator). Elle renvoie une **paire** constituée par le numérateur et le dénominateur de la fraction résultante.

subtract(Pair:D $pair1, Pair:D $pair2, Int $times = 0 –> Pair:D)

Cette méthode renvoie la soustraction des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1** et **nudepair2**. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe **reduce-fraction($numerator,** *denominator*)∗∗.*Leparamtre*∗∗**times** a le même effet que pour l'opération **add-up** (voir plus haut). Elle renvoie une paire constituée du numérateur et du dénominateur de la fraction résultante.

subtractx2(Pair:D $pair1, Pair:D $pair2, Pair:D $pair3 –> Pair:D)

Cette méthode renvoie la soustraction des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1**, **nudepair2** et **nudepair3** pour calculer la différence de trois fractions. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe : **reduce-fraction($numerator, $denominator)**. Elle renvoie une **paire** constituée par le numérateur et le dénominateur de la fraction résultante.

add-up-subtract(Pair:D $pair1, Pair:D $pair2, Pair:D $pair3 –> Pair:D)

Cette méthode renvoie l'addition des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1** et **nudepair2**, et la soustraction des numérateurs et dénominateurs passés à l'argument **nudepair3** pour calculer la somme des deux premières fractions et la différence du résultat et de la troisième fraction. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe : **reduce-fraction($numerator, $denominator)**. Elle renvoie une **paire** constituée par le numérateur et le dénominateur de la fraction résultante.

subtract-add-up(Pair:D $pair1, Pair:D $pair2, Pair $pair3:D –> Pair:D)

Cette méthode est l'inverse de la précédente, c'est-à-dire qu'elle renvoie la soustraction des numérateurs et dénominateurs passés aux attributs de la classe, c'est-à-dire, **nudepair1**, **nudepair2** pour calculer la différence des deux premières fractions et la somme du résultat et de la troisième fraction dont l'attribut

de classe est **nudepair3**. Elle utilise les modules **ppcm.pm6** et **pgcd.pm6**, ainsi que la méthode de la classe : **reduce-fraction($numerator, $denominator)**. Elle renvoie une **paire** constituée par le numérateur et le dénominateur de la fraction résultante.

breakdown-factors(Int:D @array-of-factors, Str:D $nu'de –> Array:D)

Cette méthode décompose un tableau de facteurs en facteurs premiers. (remarquez le séparateur 'du deuxième paramètre qui est parfaitement valide en Perl 6). Elle utilise la méthode **breakdown** du rôle **PrimeFactors**. L'argument **$nu'de** peut contenir les valeurs **nu** ou **de** (pour 'numerator'/'denominator') pour indiquer à la fonction qu'elle doit retourner soit un tableau des numérateurs, soit un tableau des dénominateurs. Consultez la documentation du module **prime-factors.pm6** pour plus d'informations.

breakdown-numerator(Int:D $numerator –> Array:D)

Cette méthode permet d'ajouter de la granularité aux opérations en réduisant en facteurs premiers uniquement le ou les numérateurs de chaque fraction prise isolément. Elle utilise les attributs **breakdown-numerator1** et/ou **breakdown-numerator2** et/ou **breakdown-numerator3**. Si l'un de ces trois attributs est passé à b, cela désactivera automatiquement l'attribut global **breakdown-numerators**.

breakdown-denominator(Int:D $numerator –> Array:D)

Cette méthode permet d'ajouter de la granularité aux opérations en réduisant en facteurs premiers uniquement le ou les denominateurs de chaque fraction prise isolément. Elle utilise les attributs **breakdown-denominator1** et/ou **breakdown-denominator2** et/ou **breakdown-denominator3**. Si l'un de ces trois attributs est passé à b, cela désactivera automatiquement l'attribut global **breakdown-denominators**.

fractions-product-sign(Pair:D $p1, Pair $p2:D, Pair $p3? –> Str:D)

Cette méthode est utilisée pour déduire le signe du résultat du produit des fractions passées en arguments aux attributs de la classe que sont **nudepair1**, **nudepair2** et facultativement **nudepair3** avant d'effectuer les calculs. Cette méthode est employée uniquement pour des opérations de **multiplication** ou

de **division**. Elle retourne '+' ou '−' qui seront convertis en ''ou'-' pour leur utilisation effective.

multiply(Pair:D $pair1, Pair:D $pair2, Pair $pair3? –> Pair:D)

Cette méthode est utilisée pour multiplier deux ou trois fractions données en arguments sous forme de paires numérateur => dénominateur passées aux attributs de classe **nudepair1**, **nudepair2** et facultativement **nudepair3**. Elle retourne une nouvelle **paire**.

reduce-fractions-prime-factors(Int:D @numerators, Int:D @denominators, Int $return-array = 1 –> Array:D)

Cette méthode appartient au rôle **PrimeFactors** : elle consiste à produire les facteurs qui sont dans l'un des tableaux mais pas dans l'autre. Il faut passer l'argument 1 (c'est la valeur par défaut) à l'argument **$return-array** pour retourner le premier tableau, c'est-à-dire les numérateurs ou 2 pour retourner le deuxième tableau, celui des dénominateurs. Elle est implémentée ici directement dans le code et n'est donc pas disponible comme méthode du module **operations-on-fractions** mais seulement comme méthode du module **prime-factors.pm6** dont je rappelle qu'il définit le rôle **PrimeFactors** employé par la classe **OperationsOnFractions**. Consultez aussi la documentation du module **prime-factors.pm6**.

divide(Pair:D $pair1, Pair:D $pair2 –> Pair:D)

Cette méthode est utilisée pour diviser deux fractions données en arguments sous forme de paires numérateur => dénominateur passées aux attributs de classe **nudepair1** et **nudepair2**. Elle retourne une nouvelle **paire**. Remarquez que l'attribut **nudepair3** n'est pas utilisé contrairement aux autres opérations et sera donc ignoré.

deliver-fraction-sign(Int:D $key, Int:D $value –> Str:D)

Cette méthode permet de distribuer le signe de chacune des fractions prises isolément. Elle est utilisée si l'un des attributs **reduce-fraction1** et/ou **reduce-fraction2** et/ou **reduce-fraction3** sont passés à **True** dans la méthode **calculate-fractions**(Str:D $operation –> Pair:D); je rappelle ici que ces trois champs sont à **False** par défaut.