Stéphane KELLER – Lycée agricole Louis Pasteur

CHAPITRE 7 : CALCUL AVEC DES NOMBRES DÉCIMAUX ET FRACTIONNAIRES

I. Calculs avec des nombres décimaux avec le module decimal.

1.1 Mise en situation.

```
>>> 0.5*0.5
0.25
>>> 0.7*0.7
0.48999999999999994
>>> 0.1+0.2
0.30000000000000004
>>> 0.7*0.7==0.49
False
>>> [0.5**n for n in range(1,8)]
[0.5, 0.25, 0.125, 0.0625, 0.03125, 0.015625, 0.0078125]
>>>
```

➢ Les "erreurs" ci-dessus ne sont que des conséquences de la représentation en base 2 des flottants : même un nombre aussi inoffensif que 0.1 = 1 / (2 * 5) n'est pas représenté de façon exacte en mémoire (contrairement à $0.5 = 2^{-1}$).
➢ Le module decimal permet d'effectuer des calculs exacts sur les nombres décimaux, dans les limites d'une précision fixée par l'utilisateur (mais par défaut égale à 28 chiffres significatifs).
➢ Les "nombres décimaux" (au sens du module decimal) sont obtenus par application du constructeur Decimal (noter le D majuscule) appliqué à un entier ou à une chaîne de caractères (elle-même une représentation d'un flottant, ce qui permet à l'utilisateur de former des valeurs décimales avec une précision donnée).

```
>>> from decimal import *
>>> [Decimal('0.5')**n for n in range(1,5)]
[Decimal('0.5'), Decimal('0.25'), Decimal('0.125'), Decimal('0.0625')]
>>> [Decimal('0.1')**n for n in range(1,5)]
[Decimal('0.1'), Decimal('0.01'), Decimal('0.001'), Decimal('0.0001')]
>>> Decimal('0.1')+Decimal('0.2')
Decimal('0.3')
>>>
```

1.2 Précision des calculs.

Tout cela n'est peut-être pas très impressionnant, alors on va augmenter la précision.
L'instruction getcontext().prec = n fixe la précision à n chiffres significatifs.

```
>>> getcontext().prec = 50 #augmente la précision à 50 chiffres
>>> Decimal(1) / Decimal(19)
Decimal('0.052631578947368421052631578947368421052631578947368')
>>> Decimal(3).sqrt() #racine de 3 avec 50 chiffres significatifs
Decimal('1.7320508075688772935274463415058723669428052538104')
>>> Decimal(1).exp() # le nombre e avec 50 chiffres significatifs
```

```
Decimal('2.718281828459045235360287471352662497757247093700 0')
>>> Decimal(2).ln() # ln 2 avec 50 chiffres significatifs.
Decimal('0.693147180559945309417232121458176568075500134360 26')
>>>
```

Le nombre de fonctions mathématiques qui sont compatibles avec ce module est essentiellement limité au logarithme, à l'exponentielle et à la racine carrée.

1.3 Affichage d'un nombre décimal :

```
>>> from decimal import *
>>> a = Decimal('0.125')
>>> a
Decimal('0.125')
>>> print(a)
0.125
>>>
```

```
>>a = [print(Decimal('0.1')**n) for n in range(5)]
1
0.1
0.01
0.001
0.0001
>>>
```

1.4 Utilisation du module *decimal* dans un programme.

```
from decimal import *

a = Decimal(input('Entrer un nombre décimal : ')) #a est un decimal
b = Decimal(input('Entrer un nombre décimal : ')) #b est un decimal

print('La somme est =',a+b)
```

```
>>>
Entrer un nombre décimal : 0.1
Entrer un nombre décimal : 0.2
La somme est = 0.3
>>>
```

II. Calculs avec des nombres fractionnaires avec le module fractions.

2.1 Exemples.

Le module fractions permet d'effectuer des calculs exacts sur les nombres rationnels.
Un nombre rationnel s'obtient par le constructeur Fraction qui prend en argument deux entiers (le numérateur, puis le dénominateur qui par défaut vaut 1) ou une chaîne (par exemple '12/17') ou un flottant (sous forme décimale).

```
>>> from fractions import * # importe le module fractions
>>> Fraction(1, 2) + Fraction(1, 3) #Fraction (1, 2) peut s'écrire aussi Fraction('0.5')
```

```
Fraction(5, 6)
>>> float(_)
0.8333333333333334 #convertit ce résultat en flottant
>>> from math import pi # importe la valeur _ depuis le module math
>>> Fraction(pi) # approximation rationnelle de pi
Fraction(884279719003555, 281474976710656)
>>> Fraction(pi).numerator #extrait le numérateur
884279719003555
>>> Fraction(pi).denominator #extrait le dénominateur
281474976710656
```

2.2 Affichage d'un nombre rationnel.

```
>>> from fractions import * # importe le module fractions
>>> a = Fraction('5/49') #attention à ne pas mettre d'espaces
>>> a
Fraction(5, 49)
>>> print(a)
5/49
>>>
```

2.3 Transformation d'un nombre décimal sous forme de fraction.

```
>>> from fractions import * # importe le module fractions
>>> Fraction(0.1)
Fraction(3602879701896397, 36028797018963968)
```

limit_denominator([*max_denominator*]) : recherche et renvoie la fraction la plus proche de soi ayant un dénominateur au plus égal à l'argument max_denominator. Cette méthode est utile pour trouver des approximations rationnelles d'un nombre à virgule flottante donné,
ou pour récupérer un nombre rationnel représenté comme un flottant

```
>>> Fraction(0.1).limit_denominator()
Fraction(1, 10)
>>> Fraction(0.333333333333).limit_denominator()
Fraction(1, 3)
>>> Fraction(pi)
Fraction(884279719003555, 281474976710656)
>>> Fraction(pi).limit_denominator()
Fraction(3126535, 995207)
>>> Fraction(pi).limit_denominator(100)
Fraction(311, 99)
```

2.4 Utilisation du module *fractions* dans un programme.

```
a = Fraction(input('Entrer un nombre rationnel : ')) #a est du type fraction
b = Fraction(input('Entrer un nombre rationnel : ')) #b est du type fraction

print(a, '+', b, '=', a + b)
```

```
>>>
Entrer un nombre rationnel : 1/2
```

```
Entrer un nombre rationnel : 2/3
1/2
2/3
7/6
>>>
```

CALCUL AVEC DES NOMBRES DÉCIMAUX ET FRACTIONNAIRES TESTS

I. Calculer les sommes suivantes. On affichera la valeur exacte et une valeur approchée de chaque somme.

a) $1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{10}$.	b) $1+\frac{1}{3}+\frac{1}{5}+\frac{1}{7}+...+\frac{1}{51}$.
c) $1+\frac{1}{3}+\frac{1}{6}+\frac{1}{12}+...+\frac{1}{768}$	d) $1+\frac{1}{2}+\frac{1}{2\times3}+\frac{1}{2\times3\times4}+...+\frac{1}{2\times3\times...\times10}$

II. Concevoir le programme permettant de calculer la longueur d'une courbe quelconque. On prendra, par exemple, le calcul de la longueur d'un demi-cercle et on utilisera le théorème de Pythagore.