Les nombres

1. Les différentes classes

Comme nous l'avons vu précédemment, tous les nombres sont représentés par des objets. Observons les classes employées :

```
a=10
puts a.class
a=10.0
puts a.class
```

Nous obtenons pour notre entier `Fixnum` et pour notre décimal `Float`.

Regardons maintenant la classe pour un nombre de grande taille en utilisant l'opérateur de puissance `**`.

```
a=10
puts ( a**100 ).class
```

Nous obtenons maintenant une classe `Bignum`. Pour les décimaux, il n'y a pas de changement de classe, mais la valeur deviendra infini (`Infinity`).

Voici quelques représentations de nombre :

```
-10
3.14
10e15
0xFF
# Forme hexadØcimale
-0b101
# Forme binaire
?a
# NumØro de caractŁre
```

2. Fixnum

a. Opérateurs

Cette classe a pour parent `Integer` (elle-même ayant pour parent `Numeric`).

Voici la liste des opérateurs :

- `+` : Addition
- `-` : Soustraction
- `*` : Multiplication
- `/` : Division
- `%` : Modulo
- `**` : Puissance

Des opérations binaires sont également disponibles par ces opérateurs :

- `~` : Inversion de bits
- `|` : Ou binaire
- `&` : Et binaire|
- `^` : Ou exclusif binaire (xor)
- `<<` : Décalage à gauche
- `>>` : Décalage à droite

L'opérateur `<=>` sert à comparer deux entiers avec en retour `-1`, `0` ou `+1` selon le résultat de la comparaison.

Exemple :

```
puts 1 <=> 2
puts 2 <=> 1
puts 1 <=> 1
```

Nous obtenons en sortie :

```
-1
1
0
```

b. Représentation interne

L'instruction `size` sert à déterminer le nombre d'octets nécessaires à la représentation du nombre. En utilisant les crochets, il est possible d'accéder à la valeur d'un bit sachant que l'indice 0 correspond au bit de poids faible.

Exemple :

```
a=101
```

```
puts a.size
( a.size * 8 - 1 ).downto(0) { |i| print a[i]  }
```

Nous obtenons en sortie 4 octets et la représentation binaire de 101 :

```
4
00000000000000000000000001100101
```

Observons également les différentes occupations mémoire avec un nombre de taille croissante :

```
a=10
for i in 2..10
  puts (a**i).class
  puts (a**i).size
end
```

Nous obtenons en sortie :

```
Fixnum
4
```

```
Fixnum
4
Fixnum
4
Fixnum
4
Fixnum
4
Fixnum
4
Fixnum
4
Fixnum
4
Bignum
8
```

c. Conversions

Pour effectuer une conversion en décimal ou chaîne de caractères, nous avons les instructions respectives `to_f` et `to_s`. Lors de la conversion en chaînes, il est possible de fournir en argument la base attendue (16

par exemple pour une conversion en hexadécimal).

Exemples :

```
puts 10.to_f
puts 10.to_s(2)
puts 10.to_s(16)
puts 10.to_s
```

Nous obtenons en sortie :

```
10.0
1010
a
10
```

La conversion vers un caractère s'effectue par la méthode `chr`.

Exemple :

```
puts 66.chr
```

Ce qui donne `B` en résultat.

Pour faire la conversion d'une chaîne en entier, nous utilisons la fonction `to_i`.

Exemple :

```
a = "10".to_i
puts a+1
```

Ce qui donne bien `11` en résultat.

Cette conversion d'une chaîne en entier fonctionne toujours, si la conversion n'est pas possible c'est la valeur zéro qui sera retournée.

d. Parcours

Il existe un certain nombre de méthodes acceptant des blocs d'instructions pour effectuer un parcours sur une valeur.

- `step( limit, step )` : parcours jusqu'à la limite en effectuant une augmentation de `step`

valeurs.

- `times` : parcours de 0 au nombre en cours - 1.
- `downto( limit )` : c'est un parcours descendant jusqu'à la limite incluse.
- `upto( limit )` : c'est un parcours ascendant jusqu'à la limite incluse.

Exemples :

```
1.step( 10, 5 ) { |i| puts "Etape #{i}"  }
```

Donne en sortie :

```
Etape 1
Etape 6

3.times() { |i| puts "Etape #{i}"  }
```

Donne en sortie :

```
Etape 0
Etape 1
```

```
Etape 2

3.downto(1) { |i| puts "Etape #{i}"  }
```

Donne en sortie :

```
Etape 3
Etape 2
Etape 1

1.upto(3) { |i| puts "Etape #{i}"  }
```

Donne en sortie :

```
Etape 1
Etape 2
Etape 3

1.upto(255) { |i|
  print i.chr
}
```

Affiche les deux cent cinquante-cinq caractères dans la console.

e. Fonctions utilitaires

Il existe également quelques fonctions utilitaires d'intérêt variable comme :

- `abs` : valeur absolue.
- `between?` : indique si le nombre est dans une borne.
- `next` ou `succ` : le prochain nombre.
- `zero?` : indique si la valeur est 0.

Exemples :

```
puts ( 1 - 6 ).abs
puts 1.between?( 0, 2 )
puts 1.next
puts 1.succ
puts 1.zero?
```

Nous obtenons en sortie :

```
5
true
```

```
2
2
false
```

f. Exceptions

La division par zéro peut être interceptée par l'exception `ZeroDivisionError` (voir le chapitre Programmation Objet pour les détails sur le mécanisme des exceptions).

Exemple :

```
begin
  1/0
  rescue ZeroDivisionError => r
    puts r.message
end
```

Nous obtenons en sortie :

```
divided by 0
```

3. Float

`Float` est une classe qui hérite de `Numeric`. Les opérateurs sont les mêmes qu'avec la classe `Fixnum`. La représentation décimale accepte l'exposant.

Exemples :

```
puts 1.0E1
puts 1.0E-1
```

Avec en sortie :

```
10.0
0.1
```

a. Arrondis

L'arrondi inférieur, supérieur et au plus juste est obtenu respectivement par `floor`, `ceil` et `round`.

Exemples :

```
puts 1.6.floor
```

1 en sortie

```
puts 1.2.ceil
```

2 en sortie

```
puts 1.2.round
```

1 en sortie

```
puts 1.5.round
```

2 en sortie

b. Valeurs infinies

La fonction `infinite?` retourne respectivement `nil`, `-1`, `+1` pour un décimal fini, infini négatif et infini positif. La fonction `finite?` retourne un booléen vrai (`true`) pour un état fini.

Exemples :

```
puts 1.0.infinite?
#nil en sortie
puts (1.0/0.0).infinite?
#1 en sortie
puts (-1.0/0.0).infinite?
#-1 en sortie
```

La fonction `nan?` retourne vrai pour un nombre flottant incorrect.

Exemples :

```
puts 1.0.nan?
#false en sortie
puts (1.0/0.0).nan?
#false en sortie
puts (0.0/0.0).nan?
#true en sortie
```

c. Conversions

Les fonctions `to_i` et `to_s` servent respectivement à faire une conversion en entier et en chaîne. La conversion en entier applique un arrondi inférieur.

Exemples :

```
puts (1.0).to_s
puts (2.9).to_i
```

Donne en sortie :

```
1.0
2
```

La conversion d'une chaîne vers un décimal se fait par la méthode `to_f`.

Exemple :

```
puts "1.0E1".to_f
```

Ce qui donne en sortie :

```
10.0
```

4. Bignum

La classe `Bignum` est dérivée de la classe `Integer` (ce qui est aussi le cas de `Fixnum`), donc la grande majorité des opérateurs et méthodes que nous avons vu pour les entiers sera également valable.

À noter que la conversion en Float peut donner un résultat infini si le décimal n'a pas la capacité de recevoir le nombre.

Exemples :

```
puts 10**100.to_f
puts 10**1000.to_f
```

Nous obtenons alors :

```
1.0e+100
Infinity
```