W2-S2-C4-entiers-bit-a-bit

December 15, 2014

1 Opérations *bitwise*

1.1 Compléments - niveau avancé

Les compléments ci-dessous expliquent des fonctions évoluées sur les entiers; les débutants en programmation peuvent sans souci sauter cette partie en cas de difficultés.

1.1.1 Opérations logiques : ET &, OU | et OU exclusif ^

Il est possible aussi de faire des opérations "bit-à-bit" sur les nombres entiers. Le plus simple est de penser à l'écriture du nombre en base 2

Considérons par exemple deux entiers constants dans cet exercice:

```
In []: x49 = 49
       y81 = 81
```

Ce qui nous donne comme décomposition binaire

$x49 \quad = \quad 49 \quad = \quad 32+16+1 \quad \rightarrow \quad (0,1,1,0,0,0,1)$

$y81 \quad = \quad 81 \quad = \quad 64+16+1 \quad \rightarrow \quad (1,0,1,0,0,0,1)$

Pour comprendre comment passer de $32+16+1$ à $(0,1,1,0,0,0,1)$ il suffit d'observer que

$32+16+1 = \mathbf{0}*2^6 + \mathbf{1}*2^5 + \mathbf{1}*2^4 + \mathbf{0}*2^3 + \mathbf{0}*2^2 + \mathbf{0}*2^1 + \mathbf{1}*2^0$

Et logique : opérateur & L'opération logique & va faire un 'et' logique bit à bit entre les opérandes, ainsi

```
In []: x49 & y81
```

$$\begin{array}{rrcl} & x49 & \rightarrow & (0,1,1,0,0,0,1) \\ \text{Et en effet} & y81 & \rightarrow & (1,0,1,0,0,0,1) \\ & x49 \;\&\; y81 & \rightarrow & (0,0,1,0,0,0,1) \rightarrow 17 \end{array}$$

Ou logique : opérateur | De même, l'opérateur logique | fait simplement un 'ou' logique, comme ceci

```
In []: x49 | y81
```

On s'y retrouve parce que

$$\begin{array}{rcl} x49 & \rightarrow & (0,1,1,0,0,0,1) \\ y81 & \rightarrow & (1,0,1,0,0,0,1) \\ x49 \mid y81 & \rightarrow & (1,1,1,0,0,0,1) \rightarrow 64+32+16+1 \rightarrow 113 \end{array}$$

Ou exclusif : opérateur ^ Enfin on peut également faire la même opération à base de 'ou exclusif' avec l'opérateur '^'

```
In []: x49 ^ y81
```

1.1.2 Décalages

Un décalage 'à gauche' de, par exemple, 4 positions, revient à décaler tout le champ de bits de 4 cases à gauche (les 4 nouveaux bits insérés sont toujours des 0); c'est donc équivalent à une multiplication par $2^4 = 16$

```
In []: x49 << 4
```

$$\begin{aligned} x49 &\rightarrow (0,1,1,0,0,0,1) \\ x49 << 4 &\rightarrow (0,1,1,0,0,0,1,0,0,0,0) \rightarrow 512 + 256 + 16 \rightarrow 784 \end{aligned}$$

De la même façon le décalage à droite de n revient à une division par 2^n (en fait, plus précisément, le quotient de la division)

```
In []: x49 >> 4
```

$$\begin{aligned} x49 &\rightarrow (0,1,1,0,0,0,1) \\ x49 >> 4 &\rightarrow (0,0,0,0,0,1,1) \rightarrow 2 + 1 \rightarrow 3 \end{aligned}$$

1.1.3 Une astuce

On peut utiliser la fonction *built-in* `bin` pour calculer la représentation binaire d'un entier; attention la valeur de retour est une chaîne de caractères de ytpe `str`

```
In []: bin(x49)
```

Dans l'autre sens, on peut aussi entrer un entier directement en base 2 comme ceci; ici comme on le voit $x49bis$ est bien un entier

```
In []: x49bis = 0b110001
       x49bis == x49
```

1.1.4 Pour en savoir plus

Section de la documentation python