

Opérations *bitwise*

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.

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:

```
▢ x49 = 49
  y81 = 81
```

Ce qui nous donne comme décomposition binaire

```
\begin{array}{rccccc} x49 & = & 49 & = & 32 + 16 + 1 & \rightarrow (0,1,1,0,0,0,1) \\ y81 & = & 81 & = & 64 + 16 + 1 & \rightarrow (1,0,1,0,0,0,1) \end{array}
```

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

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

$$\quad \quad \quad + \text{0} \cdot 2^2 + \text{0} \cdot 2^1 + \text{1} \cdot 2^0$$

Et logique : opérateur &

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

```
▢ x49 & y81
```

Et en effet

```
\begin{array}{rcl} x49 & \rightarrow & (0,1,1,0,0,0,1) \\ y81 & \rightarrow & (1,0,1,0,0,0,1) \\ \hline x49 \& y81 & \rightarrow & (0,0,1,0,0,0,1) \end{array}
```

Ou logique : opérateur |

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

```
▢ x49 | y81
```

On s'y retrouve parce que

$$\begin{array}{rcl} x_{49} & \& \rightarrow & (0,1,1,0,0,0,1) \setminus y_{81} & \& \rightarrow & (1,0,1,0,0,0,1) \setminus \\ x_{49} \setminus \setminus y_{81} & \& \rightarrow & (1,1,1,0,0,0,1) \end{array} \rightarrow 64+32+16+1 \rightarrow 113$$

Ou exclusif : opérateur ^

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

```
▮ x49 ^ y81
```

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 $2^4=16$

```
▮ x49 << 4
```

$$\begin{array}{rcl} x_{49} & \& \rightarrow & (0,1,1,0,0,0,1) \setminus x_{49} << 4 & \& \rightarrow & \\ (0,1,1,0,0,0,1,0,0,0,0) & \& \rightarrow & 512+256+16 \end{array} \rightarrow 784$$

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)

```
▮ x49 >> 4
```

$$\begin{array}{rcl} x_{49} & \& \rightarrow & (0,1,1,0,0,0,1) \setminus x_{49} >> 4 & \& \rightarrow & (0,0,0,0,0,1,1) \\ \& \rightarrow & 2+1 \end{array} \rightarrow 3$$

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 type `str`

```
▮ 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

```
▮ x49bis = 0b110001
| x49bis == x49
```

Pour en savoir plus

Section de la documentation python (<https://docs.python.org/2/library/stdtypes.html#bitwise-operations-on-integer-types>)