

Binaire

1ère NSI - Travaux dirigés

1. Conversion binaire vers décimal.

Donnez les valeurs entières décimales représentées par les nombres :

- 0b101
- 0b10101
- 0b0101
- 0b00101
- 0b1101 1101
- 0b1001 0111
- 0b1011 1000

2. Examen d'une représentation binaire

On considère $a = 0b1010\ 0110$ et $b = 0b11\ 1101$

1. Lequel des deux est le plus grand ?
2. Ces nombres sont-ils divisibles par 2 ? Pourquoi ?
3. Combien de bits occupe la représentation binaire de $a + b$?

3. Conversion décimal vers binaire.

1. Convertir les nombres suivants en binaire :

- 12
- 23
- 35
- 127
- 211
- 254
- 231

2. Calculer mentalement les puissances de 2 jusque 2^{20} .

3. On considère des entiers représentés sur 1 octet. Quel est le plus grand entier représentable ?

4. Quelle est la représentation binaire d'un nombre de la forme $2^k - 1$? De la forme 2^k ?

5. En remarquant que $2\ 048 = 2^{11}$, donner la représentation binaire de 2022.

```
2048 = 0b 1000 0000 xxxx
2047 = 0b  111 1111 xxxx
  25 = 0b           1 xxxx
2022 = 0b  111 1110 xxxx
```

4. Binaire et python

Python permet d'obtenir la représentation binaire d'un entier à l'aide de la fonction `bin`. Voici ce qu'on obtient avec `help(bin)` :

Help on built-in function bin in module builtins:

```
bin(number, /)
    Return the binary representation of an integer.
```

```
>>> bin(2796202)
'0b101010101010101010101010101010'
```

Inversement, la conversion d'une base b vers la représentation décimale s'obtient en passant à `int` une chaîne de caractères ainsi que la base.

Voici ce qu'on obtient avec `help(int)`

```
class int(object)
| int([x]) -> integer
| int(x, base=10) -> integer
|
| Convert a number or string to an integer, or return 0 if no arguments
| are given. If x is a number, return x.__int__(). For floating point
| numbers, this truncates towards zero.
|
| If x is not a number or if base is given, then x must be a string,
| bytes, or bytearray instance representing an integer literal in the
| given base. The literal can be preceded by '+' or '-' and be surrounded
| by whitespace. The base defaults to 10. Valid bases are 0 and 2-36.
| Base 0 means to interpret the base from the string as an integer literal.
| >>> int('0b100', base=0)
| 4
```

1. Quelle instruction saisir pour obtenir la représentation décimale de `0b1101001` ?
2. x est un entier dont la représentation binaire est `110100`. Donner deux instructions différentes permettant d'obtenir sa représentation décimale.
3. Quel sera le résultat des instructions suivantes ?

```
>>> bin(123)
>>> int("0b1111")
>>> int("0b10101", 2)
>>> bin(0)
>>> int("0b101211", 2)
```

4. Python accepte la notation `0b110` pour représenter un entier, en l'occurrence 6...

Qu'obtient-on pour les opérations suivantes ?

```
>>> 0b101 + 2
>>> bin(0b110 + 0b1110)
```

5. Capacité

1. Parmi les additions suivantes, lesquelles vont provoquer un dépassement de capacité lorsque les nombres sont encodés sur 8 bits ?
 - `1111 1011 + 1001 1111`
 - `1001 1011 + 0111 1011`
 - `0011 1011 + 1001 1001`
 - `1010 1011 + 0001 0100`
2. La taille d'une somme binaire nécessite de connaître les valeurs manipulées. Ce n'est pas le cas d'un produit. Quelle sera le nombre de bits des valeurs suivantes ?
 - `0110 × 1100`

- 1111 0011 × 1101 0101

Opérations bits à bits

Rappelons les opérations bits à bits élémentaires :

Opérations	Notation	Exemple	Remarque
AND bit à bit	&	0b1100 & 0b1010 = 0b1000	
OR bit à bit		0b1100 0b1010 = 0b1110	
XOR bit à bit	^	0b1100 ^ 0b1010 = 0b0110	
Décalage à droite	>>	0b1101 >> 2 = 0b11	Revient à diviser par 2 ⁿ
Décalage à gauche	<<	0b1101 << 2 = 0b110100	Revient à multiplier par 2 ⁿ

6. Mettre un bit à 1

On dispose d'un entier x dont on ne connaît pas la représentation.

On voudrait mettre à 1 le bit de position n en partant de la fin (et en comptant à partir de 0)

Par exemple avec $n = 4$:

```

                bit 4 à 1
xxxx xxxx xxxx ----->  xxxx xxx1 xxxx

```

Proposer une opération bit à bit qui réalise cet exploit.

7. Alternier un bit

Même point de départ, un entier x quelconque.

On veut *inverser* le bit de position n en partant de la fin (et en comptant à partir de 0)

Par exemple :

```

                retourne bit 4
xxxx xxx0 xxxx ----->  xxxx xxx1 xxxx
xxxx xxx1 xxxx ----->  xxxx xxx0 xxxx

```

Les autres bits sont inchangés.

Proposer une opération bit à bit.

8. Retourner la queue

On veut échanger les derniers bits d'un entier, à partir de son dernier 1 :

```

                retourner la queue
1101 1100 ----->  1101 1000

```

Les bits précédents le dernier 1 sont inchangés.

Proposer une opération bit à bit.

On pourra commencer par comparer les représentations binaires de x et $x - 1$.

9. Extraire une partie

On considère un entier sur 8 bits comme :

```

nombre = 0b_1011_0111
indices =   0123 4567

```

On souhaite récupérer les bits numéro 2, 3 et 4 du nombre (en comptant à partir de 0) donc : 110

1. On vire les derniers bits, il y en a 3 :

```
x = nombre >> 3
```

2. On masque avec 111 pour ne garder que trois bits : bits = nombre & 0b_111

Questions

1. Détailler les calculs précédents et vérifier le résultat.
2. Extraire les bits de position 4, 5 et 6 du nombre 0b1110 1100 1001 (en comptant à partir de 0)

10. Dénombrer les bits à 1 d'un entier

Nous allons étudier deux algorithmes qui répondent à la même question : **compter les bits valant 1 dans un entier**

Exemple : $13 = 0b\ 1101$ donc `nombre_bits_a_un(13) = 3`

Méthode 1

- On converti l'entier en binaire (exemple : `bin(13) = "0b1101"`)
- On itère sur la représentation et compte les "1".

1. Écrire une fonction Python qui implémente cet algorithme
2. La faire tourner sur 5, 9 et 14.

Méthode 2, de Brian Kernighan – Accrochez vous.

Remarques initiales :

- Retirer 1 à un entier inverse tous les bits après le dernier bit à 1. Par exemple :

décimal	binaire
10	1010
$9 = 10 - 1$	1001
$8 = 9 - 1$	1000
$7 = 8 - 1$	0111
$6 = 7 - 1$	0110
$5 = 6 - 1$	0101

- Donc, si on soustrait 1 et qu'on fait un ET bit à bit avec lui même $n \& (n - 1)$, on passe à 0 tous les derniers bits...

décimal	binaire
10	1010
$9 = 10 - 1$	1001
$10 \& 9$	1000

- Si on fait $n = (n \& (n - 1))$ jusqu'à avoir $n == 0$ et qu'on compte les tours, on a le nombre de bits à 1 dans un entier.

décimal	binaire
Tour 1	
$n = 10$	1010
$n-1 = 9$	1001
$n = (n \& (n-1))$	
$10 \& 9 = 8$	1000
Tour 2	
$n = 8$	1000
$n-1 = 7$	0111
$n = (n \& (n-1))$	
$8 \& 7 = 0$	0000

L'algorithme s'arrête parce que n vaut 0

Donc 10 comporte deux bits à 1.

1. Écrire une fonction Python qui implante cet algorithme
2. La faire tourner sur 5, 9 et 14.