


Binaire et représentation des entiers

1. L'information binaire


1.1. Le bit

L'histoire de l'informatique est intrinsèquement dépendante de l'histoire de la maîtrise de l'électricité. Fondamentalement, tout ordinateur est construit à partir de circuits électroniques qui:

- soit ne laissent pas passer le courant électrique (*Off*);
- soit laissent passer le courant électrique (*In*).

 **Exemple**

Un circuit contenant un unique interrupteur admet donc **deux états**, il est **binaire** dans ce sens.

 **Définition : bit**

Un bit est l'unité élémentaire d'information pouvant prendre deux valeurs distinctes, notées 0 et 1 (binaire). Le mot « **bit** » vient de l'anglais « **Binary Digit** », soit littéralement *chiffre binaire*.

La notation internationale pour le bit est `bit`. On parlera alors de `1 bit`, `2 bits`, `4 bits`, `8 bits`, `16 bits`, `32 bits`, `64 bits`, `128 bits`, `256 bits`, `512 bits`, `1024 bits`, `2048 bits`, `4096 bits`, `8192 bits`, `16384 bits`, `32768 bits`, `65536 bits`, `131072 bits`, `262144 bits`, `524288 bits`, `1048576 bits`, `2097152 bits`, `4194304 bits`, `8388608 bits`, `16777216 bits`, `33554432 bits`, `67108864 bits`, `134217728 bits`, `268435456 bits`, `536870912 bits`, `1073741824 bits`, `2147483648 bits`, `4294967296 bits`, `8589934592 bits`, `17179869184 bits`, `34359738368 bits`, `68719476736 bits`, `137438953472 bits`, `274877906944 bits`, `549755813888 bits`, `1099511627776 bits`, `2199023255552 bits`, `4398046511104 bits`, `8796093022208 bits`, `17592186044416 bits`, `35184372088832 bits`, `70368744177664 bits`, `140737488355328 bits`, `281474976710656 bits`, `562949953421312 bits`, `1125899906842624 bits`, `2251799813685248 bits`, `4503599627370496 bits`, `9007199254740992 bits`, `18014398509481984 bits`, `36028797018963968 bits`, `72057594037927936 bits`, `144115188075855872 bits`, `288230376151711744 bits`, `576460752303423488 bits`, `1152921504606846976 bits`, `2305843009213693952 bits`, `4611686018427387904 bits`, `9223372036854775808 bits`, `18446744073709551616 bits`, `36893488147419103232 bits`, `73786976294838206464 bits`, `147573952589676412928 bits`, `295147905179352825856 bits`, `590295810358705651712 bits`, `1180591620717411303424 bits`, `2361183241434822606848 bits`, `4722366482869645213696 bits`, `9444732965739290427392 bits`, `18889465931478580854784 bits`, `37778931862957161709568 bits`, `75557863725914323419136 bits`, `151115727451828646838272 bits`, `302231454903657293676544 bits`, `604462909807314587353088 bits`, `1208925819614629174706176 bits`, `2417851639229258349412352 bits`, `4835703278458516698824704 bits`, `9671406556917033397649408 bits`, `19342813113834066795298816 bits`, `38685626227668133590597632 bits`, `77371252455336267181195264 bits`, `154742504910672534362390528 bits`, `309485009821345068724781056 bits`, `618970019642690137449562112 bits`, `1237940039285380274899124224 bits`, `2475880078570760549798248448 bits`, `4951760157141521099596496896 bits`, `9903520314283042199192993792 bits`, `19807040628566084398385987584 bits`, `39614081257132168796771975168 bits`, `79228162514264337593543950336 bits`, `158456325028528675187087900672 bits`, `316912650057057350374175801344 bits`, `633825300114114700748351602688 bits`, `1267650600228229401496703205376 bits`, `2535301200456458802993406410752 bits`, `5070602400912917605986812821504 bits`, `10141204801825835211973625643008 bits`, `20282409603651670423947251286016 bits`, `40564819207303340847894502572032 bits`, `81129638414606681695789005144064 bits`, `162259276829213363391578010288128 bits`, `324518553658426726783156020576256 bits`, `649037107316853453566312041152512 bits`, `1298074214633706907132624082305024 bits`, `2596148429267413814265248164610048 bits`, `5192296858534827628530496329220096 bits`, `10384593717069655257060992658440192 bits`, `20769187434139310514121985316880384 bits`, `41538374868278621028243970633760768 bits`, `83076749736557242056487941267521536 bits`, `166153499473114484112975882535043072 bits`, `332306998946228968225951765070086144 bits`, `664613997892457936451903530140172288 bits`, `1329227995784915872903807060280344576 bits`, `2658455991569831745807614120560689152 bits`, `5316911983139663491615228241121378304 bits`, `10633823966279326983230456482242756608 bits`, `21267647932558653966460912964485513216 bits`, `42535295865117307932921825928971026432 bits`, `85070591730234615865843651857942052864 bits`, `170141183460469231731687303715884105728 bits`, `340282366920938463463374607431768211456 bits`, `680564733841876926926749214863536422912 bits`, `1361129467683753853853498429727072845824 bits`, `2722258935367507707706996859454145691648 bits`, `5444517870735015415413993718908291383296 bits`, `10889035741470030830827987437816582766592 bits`, `21778071482940061661655974875633165533184 bits`, `43556142965880123323311949751266331066368 bits`, `87112285931760246646623899502532662132736 bits`, `174224571863520493293247799005065324265472 bits`, `348449143727040986586495598010130648530944 bits`, `696898287454081973172991196020261297061888 bits`, `1393796574908163946345982392040522594123776 bits`, `2787593149816327892691964784081045188247552 bits`, `5575186299632655785383929568162090376495104 bits`, `11150372599265311570767859136324180752990208 bits`, `22300745198530623141535718272648361505980416 bits`, `44601490397061246283071436545296723011960832 bits`, `89202980794122492566142873090593446023921664 bits`, `178405961588244985132285746181186892047843328 bits`, `356811923176489970264571492362373784095686656 bits`, `713623846352979940529142984724747568191373312 bits`, `1427247692705959881058285969449495136382746624 bits`, `2854495385411919762116571938898990272765493248 bits`, `5708990770823839524233143877797980545530986496 bits`, `11417981541647679048466287755595961091061972992 bits`, `22835963083295358096932575511191922182123945984 bits`, `45671926166590716193865151022383844364247891968 bits`, `91343852333181432387730302044767688728495783936 bits`, `182687704666362864775460604089535377456991567872 bits`, `365375409332725729550921208179070754913983135744 bits`, `730750818665451459101842416358141509827966271488 bits`, `1461501637330902918203684832716283019655932542976 bits`, `2923003274661805836407369665432566039311865085952 bits`, `5846006549323611672814739330865132078623730171904 bits`, `11692013098647223345629478661730264157247460343808 bits`, `23384026197294446691258957323460528314494920687616 bits`, `46768052394588893382517914646921056628989841375232 bits`, `93536104789177786765035829293842113257979682750464 bits`, `187072209578355573530071658587684226515959365500928 bits`, `374144419156711147060143317175368453031918731001856 bits`, `748288838313422294120286634350736906063837462003712 bits`, `1496577676626844588240573268701473812127674924007424 bits`, `2993155353253689176481146537402947624255349848014848 bits`, `5986310706507378352962293074805895248510699696029696 bits`, `11972621413014756705924586149611790497021399392059392 bits`, `23945242826029513411849172299223580994042798784118784 bits`, `47890485652059026823698344598447161988085597568237568 bits`, `95780971304118053647396689196894323976171195136475136 bits`, `191561942608236107294793378393788647952342390272950272 bits`, `383123885216472214589586756787577295904684780545900544 bits`, `766247770432944429179173513575154591809369561091801088 bits`, `1532495540865888858358347027150309183618739122183602176 bits`, `3064991081731777716716694054300618367237478244367204352 bits`, `6129982163463555433433388108601236734474956488734408704 bits`, `12259964326927110866866776217202473468949912977468817408 bits`, `24519928653854221733733552434404946937899825954937634816 bits`, `49039857307708443467467104868809893875799651909875269632 bits`, `98079714615416886934934209737619787751599303819750539264 bits`, `196159429230833773869868419475239575503198607639501078528 bits`, `392318858461667547739736838950479151006397215279002157056 bits`, `784637716923335095479473677900958302012794430558004314112 bits`, `1569275433846670190958947355801916604025588861116008628224 bits`, `3138550867693340381917894711603833208051177722232017256448 bits`, `6277101735386680763835789423207666416102355444464034512896 bits`, `12554203470773361527671578846415332832204710888928069025792 bits`, `25108406941546723055343157692830665664409421777856138051584 bits`, `50216813883093446110686315385661331328818843555712276103168 bits`, `100433627766186892221372630771322662657637687111424552206336 bits`, `200867255532373784442745261542645325315275374222849104412672 bits`, `401734511064747568885490523085290650630550748445698208825344 bits`, `803469022129495137770981046170581301261101496891396417650688 bits`, `1606938044258990275541962092341162602522202993782792835301376 bits`, `3213876088517980551083924184682325205044405987565585670602752 bits`, `6427752177035961102167848369364650410088811975131171341205504 bits`, `12855504354071922204335696738729300820177623950262342682411008 bits`, `25711008708143844408671393477458601640355247900524685364822016 bits`, `51422017416287688817342786954917203280710495801049370729644032 bits`, `102844034832575377634685573909834406561420991602098741459288064 bits`, `205688069665150755269371147819668813122841983204197482918576128 bits`, `411376139330301510538742295639337626245683966408394965837152256 bits`, `822752278660603021077484591278675252491367932816789931674304512 bits`, `1645504557321206042154969182557350504982735865633579863348609024 bits`, `3291009114642412084309938365114701009965471731267159726697218048 bits`, `6582018229284824168619876730229402019930943462534319453394436096 bits`, `13164036458569648337239753460458804039861886925068638906788872192 bits`, `26328072917139296674479506920917608079723773850137277813577744384 bits`, `52656145834278593348959013841835216159447547700274555627155488768 bits`, `105312291668557186697918027683670432318895095400549111254310977536 bits`, `210624583337114373395836055367340864637790190801098222508621955072 bits`, `421249166674228746791672110734681729275580381602196445017243910144 bits`, `842498333348457493583344221469363458551160763204392890034487820288 bits`, `1684996666696914987166688442938726917102321526408785780068975640576 bits`, `3369993333393829974333376885877453834204643052817571560137951281152 bits`, `6739986666787659948666753771754907668409286105635143120275902562304 bits`, `13479973333575319897333507543509815336818572211270286240551805124608 bits`, `26959946667150639794667015087019630673637144422540572481103610249216 bits`, `53919893334301279589334030174039261347274288845081144962207220498432 bits`, `107839786668602559178668060348078522694548577690162289924414440996864 bits`, `215679573337205118357336120696157045389097155380324579848828881993728 bits`, `431359146674410236714672241392314090778194310760649159697657763987456 bits`, `862718293348820473429344482784628181556388621521298319395315527974912 bits`, `1725436586697640946858688965569256363112777243042596638790631055949824 bits`, `3450873173395281893717377931138512726225554486085193277581262111899648 bits`, `6901746346790563787434755862277025452451108972170386555162524223799296 bits`, `13803492693581127574869511724554050904902217944340773110325048447598592 bits`, `27606985387162255149739023449108101809804435888681546220650096895197184 bits`, `55213970774324510299478046898216203619608871777363092441300193790394368 bits`, `110427941548649020598956093796432407239217743554726184882600387580788736 bits`, `220855883097298041197912187592864814478435487109452369765200775161577472 bits`, `441711766194596082395824375185729628956870974218904739530401550323154944 bits`, `883423532389192164791648750371459257913741948437809479060803100646309888 bits`, `1766847064778384329583297500742918515827483896875618958121606201292619776 bits`, `3533694129556768659166595001485837031654967793751237916243212402585239552 bits`, `7067388259113537318333190002971674063309935587502475832486424805170479104 bits`, `14134776518227074636666380005943348126619871175004951664972849610340958208 bits`, `28269553036454149273332760011886696253239742350009903329945699220681916416 bits`, `56539106072908298546665520023773392506479484700019806659891398441363832832 bits`, `113078212145816597093331040047546785012958969400039613319782796882727665664 bits`, `226156424291633194186662080095093570025917938800079226639565593765455331328 bits`, `452312848583266388373324160190187140051835877600158453279131187530910662656 bits`, `904625697166532776746648320380374280103671755200316906558262375061821325312 bits`, `1809251394333065553493296640760748560207343510400633813116524750123642650624 bits`, `3618502788666131106986593281521497120414687020801267626233049500247285301248 bits`, `7237005577332262213973186563042994240829374041602535252466099000494570602496 bits`, `14474011154664524427946373126085988481658748083205070504932198000989141204992 bits`, `28948022309329048855892746252171976963317496166410141009864396001978282409984 bits`, `57896044618658097711785492504343953926634992332820282019728792003956564819968 bits`

Nombre en binaire	Nombre en décimal
110	6
111	7
1000	8
...	...

1.3. Grouper les bits

Une information binaire est donc une suite de 0 et de 1. Cette information peut-être de différente nature, tout dépend de la **norme d'encodage** utilisée. En soit, la même suite binaire peut signifier des choses totalement différentes comme :

- un nombre entier ;
- un nombre flottant ;
- un caractère ;
- une note de musique ;
- ...

Exemple

L'écriture binaire signifie :

- dans les nombres entiers non signés (c'est-à-dire positifs) ;

```
>>> int('101010',2)
42
```

- Le caractère * en UTF-8 :

```
>>> chr(int('101010',2))
'*'
```

Bits et quantités d'informations

Avec un unique bit, on ne peut stocker que deux informations (deux nombres, deux caractères, deux notes de musiques,...). La quantité d'information différentes pouvant être représentées dépend donc du nombre de bits utilisés :

Avec bits :

Done

Avec bits :

Done

Avec bits :

Done

Avec un système à bits, on peut représenter informations différentes.

1.4. Octets

Octets

Pour quantifier les informations binaires, on utilise souvent le mot **octet** (abusivement appelé aussi *byte* dans le monde anglo-saxon).

Un octet est un groupement de bits. Il permet de représenter informations différentes.

La notation internationale pour l'octet est . On parlera alors de (), (), etc, mais aussi de , , etc...

Remarque

Les préfixes *kilo*, *Mega*, *Giga* ..., sont bien ceux du système international, c'est-à-dire ceux pour , , ... On verra dans la partie suivante qu'ils sont parfois confondus avec les préfixes binaires (*kibi*, *Mibi*, *Gibi*...), pour lesquels le facteur de changement d'unité n'est pas mais . Ainsi .

2. Ecritures en d'autres bases

2.1. Les entiers en base décimale

Rappels : Base décimale

Un nombre entier écrit dans une base décimale (base) vérifie les conditions suivantes :

- il est écrit avec les dix chiffres arabes : ;
- chaque chiffre possède un poids, représentant une puissance de , le poids augmentant de la droite vers la gauche en partant d'un exposant .

Exercice

Enoncé

- 1.
- 2.

Solution

- 1.
- 2.

2.2. La base 2 (système binaire)

Système binaire

Un nombre entier écrit dans le système binaire vérifie les conditions suivantes :

- il est écrit avec les deux chiffres : et .
- chaque chiffre possède un poids représentant une puissance de , le poids augmentant de la droite vers la gauche en partant d'un exposant .

Remarques et notations

L'écriture possède aussi bien un sens en binaire (un-zéro-un) qu'en décimal (cent-un). Pour lever l'ambiguïté , on écrira :

- ou simplement pour le nombre en base ;
- pour le nombre en base .

? Exercice**Enoncé**

Considérons le nombre . Convertissez ce nombre binaire en décimal :

Solution**? Exercice : Conversions de la base vers la base****Enoncé**

Ecrire les nombres suivants en base :

- 1.
- 2.
- 3.

Solution

A venir !

? Exercice : Un peu de Python

Enoncé

Compléter la fonction suivante, **sans utiliser** `int(x, 2)`, afin qu'elle renvoie en base 10 le nombre passé en argument en base , sous la forme d'une chaîne de caractères. *Vous pouvez cependant utiliser la fonction built-in `int` afin de convertir une chaîne de caractère en un entier.*

```
def bin2dec(x) :
    """fonction convertissant le nombre (x)_2 en base 10
    >>> bin2dec('0')
    0
    >>> bin2dec('1')
    1
    >>> bin2dec('11')
    3
    >>> bin2dec('1000')
    8
    >>> bin2dec('11111111')
    255
    """
```

2.3. Conversions de la base vers la base

✎ Méthode : Algorithme de conversion du nombre en base

En langage naturel

```
fonction dec2bin(n) :
    base2 <- chaîne de caractère vide
    Tant que n!=0 :
        base2 <- caractere(n%2)+base2
        n <- n//2
    Renvoyer base2
```

Sous forme de diagramme

```
flowchart TB
    A(Nombre n positif en décimal) --> B(base2 chaîne de caractère vide)
    B --> C{n != 0}
    C --> |Non| F(base2 est la représentation de n en binaire)
    C --> |Oui| D[Ajouter n%2 au début de base2];
    D --> E[n prend la valeur n//2]
    E --> C
    end
```

Exemple : Conversion de en base

A venir !

? Conversions manuelles**Enoncé**

Convertir les nombres suivants en base :

- 1.
- 2.
- 3.
- 4.

Solution

A venir !

? Exercice : De l'algorithme au programme**Enoncé**

Compléter la fonction suivante en Python, qui renvoie la chaîne de caractères correspondant à l'écriture de en base .

```
def dec2bin(n) :  
    """fonction convertissant le nombre n en base 2,  
    et renvoyant la chaîne de caractères correspondante  
>>> dec2bin(0)  
'0'  
>>> dec2bin(1)  
'1'  
>>> dec2bin(3)  
'11'  
>>> dec2bin(8)  
'1000'  
>>> dec2bin(255)  
'11111111'  
    """
```

Solution

A venir !

Nombre de bits nécessaires

Soit un nombre écrit en base .

Si est l'entier tel que , alors le nombre nécessitera bits pour être représenté en binaire.

? Exercice : Nombre de bits nécessaires

Enoncé

Déterminer pour chacun des nombres ci-dessous le nombre de bits minimal qui seront nécessaire dans l'écriture binaire de ce nombre :

- 1.
- 2.
- 3.
- 4.

Solution

A venir !

2.4. Une autre base utile : l'hexadécimal

Outre que la lecture des nombres en écriture binaire par un humain est très compliquée, il faut remarquer que, de par la construction de ces nombres, la quantité de symboles utilisés en base est largement supérieur à celui utilisé en base - **fois plus grand** en moyenne sur les premiers entiers.

Il peut donc être utile de trouver un compromis entre la base , utile pour l'ordinateur, et la base , plus compréhensible par un être humain.

Ce compromis peut-être trouvé avec le système **hexadécimal**, c'est-à-dire un système de **base 16**.

Clé WIFI

Typiquement, une clé de sécurité WIFI est un nombre binaire à 128 bits (ou 256 bits). Lorsque vous vous connectez à un nouveau réseau WIFI, vous devez taper la clé sur votre ordinateur/smartphone (on élimine ici la possibilité d'un FlashCode), mais celle ci est présentée souvent sous la forme

38326f53685c2c256164712838, déjà particulièrement pénible à taper, et les erreurs de recopiage sont faciles à faire et difficiles à retrouver.

Ce nombre, sous sa forme binaire est celui-ci :

```
111000001100100110111101010011011010000101110000101100001001010110000101100100011100010010100000011000
```

De quoi faire encore plus d'erreurs en le recopiant...

Base hexadécimale

Un nombre entier écrit dans une base hexadécimale (base 16) vérifie les conditions suivantes :

- il est écrit avec les seize chiffres hexadécimaux : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F ;
- chaque chiffre possède un poids représentant une puissance de 16, le poids augmentant de la droite vers la gauche en partant d'un exposant 0.

Exemple

Le nombre hexadécimal 38326f53685c2c256164712838 correspond donc en décimal à :

A compléter

Solution

? Exercice

Enoncé

Convertir de l'hexadécimal vers le décimal :

-
-
-

✎ Méthode : Convertir vers l'hexadécimal depuis le décimal

Pour convertir vers l'hexadécimal depuis le décimal, on utilise la même méthode qu'en binaire mais en divisant par .

📌 Exemple

Enoncé

Convertir le nombre en hexadécimal.

Solution

A venir !

i Remarques

- A un chiffre dans la base , correspond exactement chiffres dans la base .
- Cela signifie que pour écrire un octet en hexadécimal, **deux chiffres hexadécimaux suffisent**. C'est pour cette raison que les octets écrits en base deux sont **groupés par 4 chiffres** :
- Ce système de notation est très pratique pour noter les codes des couleurs (par exemple en RGB : `#7455BA` signifie que l'octet représentant le canal rouge a pour valeur `74`, celui du canal vert `55`, et celui du canal bleu `BA`), pour les clés de chiffrement (code Wifi par exemple), ...

i Roue numérique



3. Opérations élémentaires sur les nombres binaires

3.1. Sommes de nombres binaires

Méthode : Additionner deux nombres entiers en base 2

La technique d'addition de deux nombres binaire est la même que pour des nombres en écriture décimale :

En décimal

En binaire

3.2. Produits de nombres binaires

✎ Méthode : Multiplier deux nombres entiers en base 2

La technique de multiplication de deux nombres binaire est la même que pour des nombres en écriture décimale - mais la retenue peut se propager parfois plus loin que le rang immédiatement supérieur :

En décimal

En binaire
