**Nombres numériques?**

**Introduction**

Ouvrir un interpréteur Python. Nous allons l'utiliser comme une calculatrice. Noter pour chaque calcul le résultat tel qu'il s'affiche à l'écran.

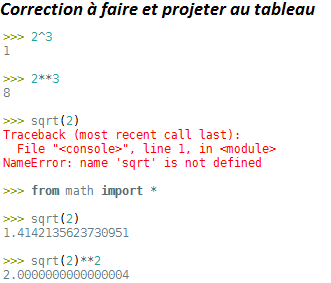
>>> 5+3

>>> 2 - 9

>>> 7 + 3 \* 4 #*les priorités des opérations mathématiques sont elles respectée ?*

*#que faire pour que l’addition soit effectuée en premier ?*

>>> 20 / 3 #*était-ce ce que vous attendiez ?*

Quels types de nombres connaissez-vous en mathématiques? Quels types de nombres semble connaître la machine?

On souhaite calculer des puissances et des racines.

Comment essayez vous de faire calculer 23 et √2 ?

Calculer √2 2? Que constatez vous ?

L'ordinateur ne connait que deux principaux types de nombres: les entiers et les "flottants" (nombres à virgule flottante). Il ne manipule les nombres non-entiers qu'avec une précision limitée. Il ne comprend pas ce qu'est √2 mais en fournit une valeur approchée. Cette erreur d'approximation peut se propager dans les calculs.

**Nombres entiers en binaire**

Nous comptons habituellement avec 10 symboles (de 0 à 9), parce que nous avons 10 doigts. Cela n’a historiquement pas toujours été le cas (il existe des systèmes à 12, 20 chiffres ou 60, c’est joli car c’est divisible par 2, 3, 4, 5, et 6, voire plus complexe comme le système romain).

L’ordinateur n’a pas de doigts. Les ordinateurs à l'heure actuelle sont basés sur l'électronique. Ils sont composés de fils dans lesquels il y a du courant (ou plutôt de la tension) (1) ou pas (0). Ils n'ont donc à leur disposition que deux symboles 0 et 1. Cet élément élémentaire d'information est appelé BIT (abréviation de Binary Digit).

Comment écrire un nombre comme 23 en binaire?

en décimal : 1789 signifie 1 x 1000 + 7x100 + 8x10 + 9x1

On accole des chiffres (entre 0 et 9) qui correspondent aux puissances de 10.

en binaire, on va accoler des chiffres ( … ou …) qui correspondront aux puissances de …

Compléter le tableau suivant des puissances de 2 puis compléter la ligne correspondant à la décomposition de 23.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  | Nombre décimal |
|  |  |  | 1 | 0 | 1 | 1 | 1 | 23 |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

décomposer en binaire

(36)d = (153)d = (88)d =

retrouver la valeur décimale de:

(00110110)b = (10110011)b =

Pourquoi a-t-on mis 8 colonnes?

Pour des raisons d'efficacité de traitement, les bits ont très vite été regroupés. Une association de 8 bits s'appelle un octet (attention, cela se dit byte en anglais). Les ordinateurs actuels ont des processeurs 64 bits (32 bits pour les ordinateurs un peu moins performants/récents) ce qui signifie qu'un "mot" élémentaire pour notre ordinateur comprend 8 (ou 4 octets). L'humain continue de les représenter par octet pour des raisons de lisibilité.

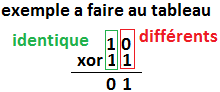
Quel est le plus petit et le plus grand nombre que l'on peut écrire avec 8 chiffres en binaire? Combien de nombres peut on représenter en tout?

Avec un octet on peut représenter tous els nombres entre 0 et 255 (255 = 128 + 64 + 32 +16 +8 +4 +2 +1). Cela représente 256 nombres possibles.

*remarque:* les opérations classiques additions, soustractions, multiplications et divisions s'appuient sur les mêmes algorithmes en binaire et en décimal. On peut remarquer que les tables de multiplication sont minimales (0x0=0, 0x1=1 et 1x1=1 )

Il existe aussi de drôles d'opérations dites bit à bit comme le ET (AND il faut que les deux bits comparés vaillent 1 pour que le résultat soit 1, sinon c'est 0), le OU (OR, il faut que l'un au moins des deux bits vaille 1 pour que le résultat soit 1) et le OU exclusif (XOR, si les deux bits sont identiques c'est 0, s'ils sont différents c'est 1).

Le résultat étonnant 2^3=1 de Python tient au fait que le symbole ^ en Python est associé à l'opération XOR.



**Nombres flottants**

Lorsque nous écrivons un nombre décimal comme 0,125 cela signifie :

0,125 = 0 + 1x1/10 + 2x1/100 + 5x1/1000 = 0 + 1x10-1 + 2x10-2+ 5x10-3

De la même manière, les nombres "flottant" sont représentés dans la machine par leur décomposition suivant les puissances de 2 négatives.

par exemple : 0,125 = 0 + 0x1/2 + 0x1/4 + 1x 1/8 = 0+ 0x1/21 +0x 1/22 + 1x1/23 s'écrit (0,001)b

De même que dans le système décimal, certains nombres n'ont pas d'écriture décimale finie comme 1/3 (≈0,3333…) ou trop longue pour être recopiée, dans le système binaire, certains nombres n'ont pas de représentation binaire finie (ou suffisamment courte pour être encodée).

C'est le cas de (0,1)d = (0.0001100110011001100110011001100110011001100110011...)b

Les langages de programmation réservent une place fixe et limitée pour encoder chaque nombre flottant. Ils ne stockent donc pas vraiment 0,1 mais une approximation. Pour voir laquelle, retourner sur l'interpréteur Python:



Vérifier quelle est la valeur stockée pour 0,125 et pour 0.2 ?

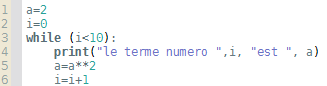
Calculer 0.1+0.2

Comprenez vous d'où vient la bizarrerie?

Comme beaucoup d'utilisateurs n'ont pas besoin de plus de précisions, les erreurs d'approximations sont masquées en n'affichant que les premières décimales. Mais parfois, l'erreur se propage jusqu'à être visible.

**Quel est le plus grand nombre que l'on peut représenter?**

On donne le script Python suivant.



Le recopier et l'exécuter

Exprimer la valeur de a en fonction de i

Quel est le type de la variable a? entier

Modifier le code pour faire afficher les termes jusqu'au numéro 15. Que constatez vous?

Les nombres entiers sont très grands, peu lisibles. Les résultats sont exacts et rapide (si on voit un ralentissement c'est du à l'affichage, mais pas au calcul en lui même)

Si on avait initialisé a par a=2.0

* quel aurait été le type de a? flottant
* Que constatez vous? ça plafonne à , après c'est trop grand pour être encodé

video synthèse sur le binaire du MOOC SNT partie S telechargeable, projetable en selectionnant les bons morceaux