Afin de pouvoir discuter avec la machine, il faut traduire le langage des hommes en langage machine. Le codage est l'une de ces étapes.

I Introduction

Rappelons ce que l'on avait vu à la 1ère scéance. Nous avions vu que $(\sqrt{2})^2 = 2{,}000\,000\,000\,000\,000\,4$!

Nous l'avons expliquer par le fait qu'une variable nécessitait une certaine taille mémoire selon son type.

◇ pour coder un entier on utilise . octets (... bits),

◇ pour coder un réel on utilise . octets(... bits).

? Quelles informations est mise dans ces cases mémoires ? Et comment elles y sont codées ?

II Majoration des erreurs de calculs

Puisque certains nombres (*e.g.* $\sqrt{2}$) ne sont pas codés de manière exacte, il existe une erreur entre la valeur exacte et la valeur utilisée. Comment cette erreur est-elle répercutée dans les calculs ?

expl : $a = \sqrt{2}$ est la valeur exacte mais imaginons que `python` prenne pour valeur $\tilde{a} = 1{,}414$. L'erreur entre la valeur exacte ($a = 1{,}414213\ldots$) et la valeur utilisée par `python` ($\tilde{a} = 1{,}414$) est donc $|a - \tilde{a}|$ qui est inférieure à 3.10^{-4}, *i.e.* $|a - \tilde{a}| \leqslant 3.10^{-4}$ (l'erreur est d'au plus 3.10^{-4}).

De même supposons que $b = \sqrt{3} = 1{,}732050\ldots$ et que `python` prenne $\tilde{b} = 1{,}7321$, l'erreur est $|b - \tilde{b}| \leqslant 5.10^{-5}$.

◇ Si on somme $\tilde{a}$ et $\tilde{b}$ pour faire le calcul de $a + b$, on aura une erreur au plus égale à : $|a + b| \leqslant |a| + |b|$ et $|ab| = |a| \times |b|$, on a $|\tilde{a} + \tilde{b} - \widetilde{(a + b)}| \leqslant 3.10^{-4} + 5.10^{-5} = 35.10^{-5}$.

◇ Tandis que pour la multiplication $\tilde{a} \times \tilde{b}$ au lieu de $a \times b$, on aura une erreur $|\tilde{a}\tilde{b} - ab| = |\tilde{a}\tilde{b} - b\tilde{a} + b\tilde{a} - ab| = |\tilde{a}(\tilde{b} - b) + b(\tilde{a} - a)| \leqslant |\tilde{a}| \times |b - \tilde{b}| + |b| \times |\tilde{a} - a| = 1{,}414 \times 5.10^{-5} + \sqrt{3} \times 3.10^{-4} \approx 59.10^{-5}$ (rq : majoration plus importante que pour l'addition)

Exo 1 : calculer une majoration des erreurs faites lors des calculs somme et multiplication entre :

◇ $a = \sqrt{5} = 2{,}23606\ldots$ et $\tilde{a} = 2{,}2361$, $b = \sqrt{331} = 18{,}193405\ldots$ et $\tilde{b} = 18{,}1934$

◇ $\tilde{a} = 24{,}04163$ (erreur de 7.10^{-5}) et $\tilde{b} = 10578{,}4563$ (erreur de 1.10^{-7}).

III Codage Décimal

De manière usuelle, nous utilisons 10 symboles (0 ;1 ; 2 ; ... 9) car nous avons .....
On dit que l'on écrit en base ....

Rq : cela n'a pas toujours été le cas : les Babyloniens utilisaient la base 60 qui était extrèmement pratique pour les calculs (60= .....).
nb : il reste encore des traces de la numération en base 60 dans notre quotidien (*cf.* .....).

Lorsque que l'on écrit un nombre en base 10, quelles opérations fait-on à partir des symboles (les 10 chiffres) et de la base pour interpréter et obtenir la valeur du nombre ? En d'autres termes, comment décomposer un nombre à l'aide des chiffres et de 10 ?

expl : 2513 =

On voit que la suite de symboles 2, 5, 1, 3 forment le nombre .....
On a donc décomposé le nombre à l'aide des puissances de 10 et des symboles 0, 1, ... 9.

IV Codage Binaire

1 Écriture en base 2

L'ordinateur est composé de câbles dans lesquels circulent un courant électrique. On code l'information à l'aide de . états : ..... .

L'ordinateur utilise un système avec les symboles ..... On dit qu'il s'agit d'un système en base ....

? Comment écrire un nombre en base 2 ?

Comme pour la base 10, on va chercher à décomposer le nombre à l'aide des puissances de 2 et des symboles 0 & 1.

On peut décomposer un nombre comme la somme de puissance de 2 :

Rappel :

i	0	1	2	3	4	5	6	7	8	9	10
2^i	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

$$N = \sum_{i=0}^{n} a_i 2^i = a_n 2^n + \dots a_1 2^1 + a_0 2^0$$

On dit que la suite de symboles $a_n \dots a_1 a_0$ est l'écriture ..... et on la note $\overline{a_n \dots a_1 a_0}^2$ ou $(a_n \dots a_1 a_0)_2$. Chacun des symboles s'appelle des .....

rq : comme pour la base 10 où l'on rassemble les symboles par paquets de 3, les bits sont rassemblés par paquet de 4 (les quartets) ou par paquets de 8 (les octets).

rq : pour tout nombre entier, cette décomposition à l'aide des puissances de 2 existe et est unique, ainsi l'écriture en base 2 existe et est unique.

2 base 2 → base 10

expl : retrouver en base 10 les nombres

$(101)_2 = \dots$

$(101\ 1010)_2 = \dots$

expl : Quel nombre en base 10 code chacun de ces .....

$(1001\ 0101)_2$

$(0011\ 0010)_2$

Exo 2 : Convertir en base 10 ces nombres binaires : $(1001)_2$; $(10\ 1101)_2$; $(0110\ 0101)_2$; $(1001\ 1100)_2$; $(100\ 1010\ 0011)_2$

? Quelques limites du codage en base 2.

Quel est le plus grand entier que l'on puisse coder à l'aide de 3 symboles - en base dix ? - en base deux ?

À l'aide de n symboles - en base dix ? - en base deux ?

Exo 3 : On se propose de coder un algorithme qui permette à partir d'une chaîne de caractère `NBinaire` formant un nombre en base 2, de renvoyer la valeur en base 10.

Variables

NBinaire : chaîne de caractères
ValDecimale : entier
b, i, n : entier

Algorithme

AFFICHER "Entrer un nombre écrit en base 2"
SAISIR *NBinaire*
n **PREND LA VALEUR** longueur de NBinaire
ValDecimale **PREND LA VALEUR** ...
POUR i **ALLANT DE** 0 **À** ... **FAIRE**
  b **PREND LA VALEUR NUMÉRIQUE** du i^e caractère de *NBinaire*
  ValDecimale **PREND LA VALEUR** .....
FIN-POUR
AFFICHER "Ce nombre s'écrit en base ... : "
AFFICHER .....

Fin Algorithme

1. compléter l'algorithme
2. implémenter en langage `python`.

3 base 10 → base 2

Faisons l'inverse, à partir d'un nombre entier, calculons son écriture en base 2.

base 10	0	1	2	3	4	5	6	7	8
base 2									

base 10	9	10	11	12	13	14	15	16	17
base 2									

expl : écrire en base 2 les nombres suivants
$1 = \ldots$
$5 = \ldots$
$12 = \ldots$

On peut remarquer que tous les nombres pairs .....

◇ méthode 1 : décomposition du nombre en puissances de 2

	2^5	2^4	2^3	2^2	2^1	2^0
14						
49						

◇ métohde 2 : les restes de la division euclidienne par 2
11 : 11/2=5 reste $\underline{1}$; 5/2=2 reste $\underline{1}$; 2/2=1 reste $\underline{0}$; 1/2=0 reste $\underline{1}$ ainsi en lisant de d. à g. (←) : $11 = (1011)_2$
Cette suite d'opérations s'écrit aussi $11 = (5)\times 2+1 = ((2)\times 2+1)\times 2+1 = ((\underline{1}\times 2+\underline{0})\times 2+\underline{1})\times 2+\underline{1} = (1011)_2$
14 :

34 :

49 :

rq : il est difficile de programmer la méthode 1 sans utiliser des fonctions mathématiques avancées. On lui préférera la méthode 2 qui ne fait appel qu'à des calculs .....

Attention toutefois, il faut ..... des restes pour obtenir l'écriture en base 2.

Exo 4 : à l'aide d'une des méthodes, convertir en base 2 l'écriture de ces nombres : 6 ; 37 ; 73 ; 300.
Exo 5 : implémentation de la méthode n° 2 (exo 9 maison)

4 Opérations sur les binaires

la somme ou le produit de deux binaires se fait de la même manière qu'une addition ou qu'une multiplication en primaire sauf que $1 + 1 = (10)_2$ (il y a une retenue dès que l'on fait 1+1 !)

expl :

$$\begin{array}{rccc} & 1 & 0 & 1 \\ + & & 1 & 1 \\ \hline & & & \\ \hline \end{array}$$

$$\begin{array}{rccc} & & 1 & 0 & 1 \\ \times & & & 1 & 1 \\ \hline & & & & \\ + & & & & \bullet \\ \hline \end{array}$$

expl : calculer $(1011)_2 + (110)_2$: et $(1011)_2 \times (110)_2$:

Vérifier vos résultats en convertissant vos opérandes et résultats en base 10.
Exo 6 : calculer la somme et le produit des nombres $(10111)_2$ et $(101)_2$. Vérifier en convertissant en base 10 les opérandes et les résultats.

Travail à faire pour la prochaine séance

Exo 7 : (retour exo 2) Implémenter l'algorithme de l'exo 2 avec une boucle `while` plutôt que `for`
Nb : si vous avez eu des difficultés, la solution de l'exo 2 est similaire à cette partie de code :

```
print("Entrer un nombre ecrit en base 2")
NBinaire = ...
n = ...(NBinaire)
...
for i in ...:
    b = ...(NBinaire[...])
    ValDecimale = ...+...*2**(...)
print(...)
```

Pour les exos suivants, on rappelle que

- ◇ `%2` renvoie le reste de la division euclidienne par 2 (*e.g.* `3%2`)
- ◇ `//2` renvoie le quotient par 2 (*e.g.* `3//2`)
- ◇ on peut "coller" deux chaînes de caractères à l'aide de l'opérateur de concaténation (`+`) (*e.g.* `"to"+"to"`)

Exo 8 :

1. Afficher le reste de la division euclidienne de 5, 17, 246 par 2 puis par 5.
2. Afficher le quotient de la division euclidienne de 5, 17, 246 par 2 puis par 5.
3. Proposer alors une fonction `divEuclide()` qui demande à l'utilisateur d'entrer un nombre p et un quotient q puis affiche la division euclidienne de p par q.

e.g. si $p = 15$, $q = 4$ `divEuclide()` affiche "$15 = 4 \times 3 + 3$"

Exo 9 :

1. demander à l'utilisateur son nom puis demander son prénom. Définir alors une variable `NomPrenom` qui soit la concaténation de son nom et de son prénom séparés par une virgule. Afficher alors la variable `NomPrenom`.
2. Proposer une fonction `inverse(mot)` qui créer une nouvelle chaîne de caractères dans laquelle la première lettre est devenue la dernière, la deuxième est devenue l'avant-dernière, etc.

e.g. `inverse("elephant")` renverra `"tnahpele"`.

Exo 10 : (retour exo 4)

1. En vous aidant des exo 7 & 8, proposer un algorithme qui fasse la conversion du décimal au binaire.
2. Proposer une implémentation `python`.

Si vous bloquez, vous pouvez vous inspirer du code ci-dessous.

```
print("Entrer un nombre ecrit en base 10")
ValDec = ...                # il faut que ValDec soit un entier
NBin = ""                   # NBin est une chaine de caracteres vide
while (ValDec > ...):
   NBin = str(...)+...
   ValDec = ...
print("Ce nombre s'ecrit en base 2 : {}".format(NBin))
```