Formation ISN - A.1

Julien Lefevre, Edouard THIEL et Michel Van Caneghem

Université d'Aix-Marseille (AMU)

Janvier 2012

Les transparents de ce cours sont téléchargeables ici :
http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ens/ISN/

ISN : Informatique et Sciences du Numérique

La réforme du Lycée : s'applique en sept. 2012 pour les terminales.

Les spécialités de terminale S : les 4 actuelles + 1 nouvelle : ISN

- Enseignement : 2 heures hebdomadaires
- Programme : BO spécial n°8 du 13 octobre 2011
- Évaluation : BO spécial n°7 du 6 octobre 2011
  épreuve de 20 mn (exposé 8 mn + questions 12 mn)
- Mise en place : BO n°36 du 6 octobre 2011
- Formation des enseignants volontaires réalisée au niveau des académies.

Intervenants dans cette formation

- Essentiellement des enseignants-chercheurs de l'Université d'Aix-Marseille (AMU) : en informatique, en mathématiques, de l'IREM.
- Quelques enseignants STI du secondaire (projets).

Présentation de votre interlocuteur :

- Professeur d'Informatique à AMU, Campus de Luminy ;
- chercheur au Laboratoire d'Informatique Fondamentale (LIF) ;
- responsable de la Licence d'Informatique AMU.

Pourquoi la spécialité ISN ?

Au 20^e siècle, à l'ère industrielle, développement d'une culture physique-chimie pour comprendre le monde d'alors.

Au 21^e siècle, informatique omniprésente dans notre vie quotidienne : nécessité de développer une culture informatique

- pour comprendre les concepts importants, les enjeux sociétaux, avoir un esprit critique ;
- faire la distinction entre la Science informatique et l'utilisation grand public de l'outil

Comment développer la culture en informatique :
en l'enseignant dès le lycée !

Second objectif : attirer plus de jeunes vers des études d'informatique.

Faire des études en informatique

- C'est le domaine où il y a le plus de débouchés :
  - l'informatique est partout ;
  - la recherche en Science informatique progresse continuellement, faisant avancer aussi la technologie ;
  - elles nourrissent toutes les autres sciences et tous les secteurs d'activité ;
  - nombreux métiers à bac $+3$, $+5$, $+8$, potentiellement bien rémunérés ...
- Comme en médecine ou en droit, les meilleures formations en informatique sont à l'Université.
- Les écoles d'ingénieurs généralistes donnent une formation moins poussée en informatique ; éviter absolument les petites écoles très chères.

Différents cursus d'études en informatique

- Après le bac : CPGE, prépas intégrées, DUT, BTS, Licence info ou Licence math/info, Licence info filière PEI
- Après bac+2 : école ingénieur, licence pro, Licence L3
- Après L3 : école ingénieur sur dossier, Master info
- Après M1 : école ingénieur 2e année sur dossier, Master L2 professionnel ou recherche
- Après Master (ou école) : les meilleurs étudiants font un Doctorat en informatique (PhD)

Licence et Master : orientation progressive, différents parcours, réorientations toujours possibles.

Conditions d'études

- CPGE : stress, classes surchargées, succès aléatoire, pas d'info
- DUT, BTS : horaires très chargés, contenu très scolaire
- BTS : poursuite d'études difficile
- Licence et Master info :
  - classes petits effectifs, excellent encadrement, bon matériel ;
  - moins scolaire, développement personnel ;
  - évaluation chaque semestre + contrôle continu ;
  - programmes remis en cause tous les 4 ans (évaluation par AERES et habilitation du ministère).

La Licence Informatique AMU

2012 : fusion des 3 universités d'Aix-Marseille $\rightarrow$ AMU
Fusion des 3 Licences math/info $\rightarrow$ Licence info + Licence Math

- Première année Tronc commun entre Lic. info et Lic. math ; tutorat, encadrement doublé en TP
- Parcours : Info (L1-L3), PEI (L1-L2), Info-Bio (L2-L3), MIAGE (L3)
- Enseignements dupliqués sur plusieurs sites : Luminy, St-Charles, St-Jérôme (L1), Aix ; télé-enseignement
- 1800h sur 3 ans en 6 semestres, 2 stages
- Parcours Info : 50% info, 20% maths, 8% anglais, 5% UEs transversales, 15% d'options

Programme de la formation pour ISN

Pour l'académie d'Aix-Marseille :

67,5 heures de formation réparties en 5 modules

A. **Information numérique**
B. **Langages et programmation**
C. **Algorithmique**
D. **Machines numériques**
E. **Projet**

Programme de la formation pour ISN

A. **Information numérique**
   - A.1 Codage numérique (3h)
   - A.2 Représentation numérique des données analogiques (3h)
   - A.3 Théorie de l'information (3h)
   - A.4 Information structurée 1 (3h)
   - A.5 Information structurée 2 (1,5h)
   - A.6 Contrôle de l'information (3h)
   - A.7 Informatique et société (3h)

B. **Langages et programmation**

C. **Algorithmique**

D. **Machines numériques**

E. **Projet**

Programme de la formation pour ISN

A. **Information numérique**

B. **Langages et programmation**

B.1	Programmation 1	(1,5h)
B.2	Programmation 2	(3h)
B.3	Programmation 3	(3h)
B.4	Programmation 4	(3h)
B.5	Langages formels	(1,5h)
B.6	Langages du web	(3h)

C. **Algorithmique**

D. **Machines numériques**

E. **Projet**

Programme de la formation pour ISN

A. **Information numérique**

B. **Langages et programmation**

C. **Algorithmique**

C.1	Algorithmique 1	(1,5h)
C.2	Algorithmique 2	(3h)
C.3	Algorithmique 3	(3h)
C.4	Algorithmique avancée 1	(3h)
C.5	Algorithmique avancée 2	(3h)

D. **Machines numériques**

E. **Projet**

Programme de la formation pour ISN

A. **Information numérique**

B. **Langages et programmation**

C. **Algorithmique**

D. **Machines numériques**

- D.1 Architecture des ordinateurs 1 (3h)
- D.2 Architecture des ordinateurs 2 (3h)
- D.3 Réseaux (4,5h)
- D.4 Autre module (3h)

E. **Projet**

- E.1 Initiation à la pédagogie de projet (6h)

A Information numérique

A.1 Codage numérique

1. Opérations binaires et logique booléenne
2. Bit, octet, mot, unités
3. Codage binaire des entiers et opérations
4. Représentation des nombres réels
5. Codage numérique du texte (standards)

1 - Opérations binaires et logique booléenne

- Quantité élémentaire d'information : le bit (BInary digiT).

Deux valeurs : 0 ou 1

Selon le contexte, peuvent correspondre à : nombres 0 ou 1 (numérique), faux ou vrai (logique), ouvert ou fermé (interrupteur), nord ou sud (magnétique), noir ou blanc (optique), absence ou présence de trou (carte perforée), etc.

- Algèbre de Boole (George Boole, milieu 19e siècle) :
  - partie des mathématique qui s'intéresse aux opérations et aux fonctions sur les variables logiques.
  - En logique, domaine définissant les lois formelles du raisonnement, utilisée pour le calcul des propositions ($\wedge, \vee, \neg, \exists, \forall, \Rightarrow, =$);
  - utilisée dans la conception des circuits électroniques.

Opérateurs booléens

Opérateurs booléens : not (négation), and (et), or (ou), xor (ou exclusif).

Tables de vérité :

x	not x
false	true
true	false

x	y	x and y	x or y	x xor y
false	false	false	false	false
false	true	false	true	true
true	false	false	true	true
true	true	true	true	false

Numériquement, avec false $= 0$ et true $= 1$:

$$\begin{aligned} \text{not } x &= 1 - x \\ x \text{ and } y &= x * y \\ x \text{ xor } y &= (x + y) \bmod 2 \\ x \text{ or } y &= ? \end{aligned}$$

Propriétés (1/3)

Complémentarité :

$$\text{non } (\text{non } x) = x$$

Idempotence :

$$x \text{ and } x = x$$
$$x \text{ or } x = x$$

Élément neutre :

$$x \text{ and true} = x$$
$$x \text{ or false} = x$$

Exercices : preuve avec les tables de vérité ;
calculer x and false, x or true

Propriétés (2/3)

Commutativité :

$$\begin{aligned} x \text{ and } y &= y \text{ and } x \\ x \text{ or } y &= y \text{ or } x \\ x \text{ xor } y &= y \text{ xor } x \end{aligned}$$

Associativité :

$$\begin{aligned} (x \text{ and } y) \text{ and } z &= x \text{ and } (y \text{ and } z) \\ (x \text{ or } y) \text{ or } z &= x \text{ or } (y \text{ or } z) \\ (x \text{ xor } y) \text{ xor } z &= x \text{ xor } (y \text{ xor } z) \end{aligned}$$

Distributivité :

$$\begin{aligned} x \text{ and } (y \text{ or } z) &= (x \text{ and } y) \text{ or } (x \text{ and } z) \\ x \text{ or } (y \text{ and } z) &= (x \text{ or } y) \text{ and } (x \text{ or } z) \end{aligned}$$

Exercices : preuves avec tables de vérité ; distributivité avec xor ?

Propriétés (3/3)

Lois de De Morgan (milieu 19e siècle) :

$$\begin{aligned} \text{not } (x \text{ or } y) &= (\text{not } x) \text{ and } (\text{not } y) \\ \text{not } (x \text{ and } y) &= (\text{not } x) \text{ or } (\text{not } y) \end{aligned}$$

Négation des inégalités :

$$\begin{aligned} \text{not } (x < y) = x \geq y \quad ; \quad \text{not } (x \leq y) = x > y \\ \text{not } (x > y) = x \leq y \quad ; \quad \text{not } (x \geq y) = x < y \end{aligned}$$

Exercices : simplifier

$(x < 7)$ and $(y < 3)$ or not $((x \geq 7)$ or $(y < 3))$

not $(x$ and $(\text{not } x$ or $y))$ or y

Exercice : montrer que $x \Rightarrow y$ s'écrit not x or y.

2 - Bit, octet, mot, unités

Les bits sont regroupés par paquets adjacents pour représenter de l'information.

Un octet (byte) est constitué de 8 bits.

1	1	0	1	0	1	1	0
7	6	5	4	3	2	1	0

Si l'on numérote les bits de 0 à 7, le bit numéro 0 est le bit de poids faible, et le bit numéro 7 est le bit de poids fort.

Si l'on considère un octet comme un nombre écrit en base 2, sa valeur numérique est

$$\sum_{i=0}^{7} b_i \, 2^i$$

Ici, on a $11010110_2 = 2^7 + 2^6 + 2^4 + 2^2 + 2^1 = 214_{10}$

Bases 8 et 16

La base 2 est peu pratique à écrire, et la base 10 oblige à faire un calcul. Deux autres bases sont souvent employées :

- La base 8, octal : chiffres de 0 à 7 ; préfixe 0
- La base 16, hexadécimal : chiffres de 0 à 9 puis A à F ; préfixe 0x

Pour convertir de base 2 en base 8, il suffit de faire des groupes de 3 bits.

1	1	0	1	0	1	1	0
3			2			6	

11010110_2 s'écrit donc 0326 en octal.

Pour convertir de base 2 en base 16, il suffit de faire des groupes de 4 bits.

1	1	0	1	0	1	1	0
D				6			

11010110_2 s'écrit donc 0xD6 en hexadécimal.

Les mots

Un mot (machine) est la quantité de bits standard manipulée par un microprocesseur (CPU). La taille du mot s'exprime en bits (ou en octets).

Un microprocesseur est d'autant plus performant que ses mots sont long, car les données qu'il traite à chaque cycle sont plus nombreuses. C'est pourquoi on classe les microprocesseurs par la taille de leur mot : 8, 16, 32, 64 bits (soit 1, 2, 4 ou 8 octets).

Valeur max d'un mot :

$2^{16} - 1 = 65\,535 = 0\text{xFFFF}$
$2^{32} - 1 = 4\,294\,967\,295 = 0\text{xFFFF FFFF}$
$2^{64} - 1 = 18\,446\,744\,073\,709\,551\,615 = 0\text{xFFFF FFFF FFFF FFFF}$

Unités officielles

Standard SI		Unités binaires	
Unité	Valeur	Unité	Valeur
kilobit (kb)	10^3	kibibit (Kibit)	2^{10}
mégabit (Mb)	10^6	mébibit (Mibit)	2^{20}
gigabit (Gb)	10^9	gibibit (Gibit)	2^{30}
térabit (Tb)	10^{12}	tébibit (Tibit)	2^{40}
pétabit (Pb)	10^{15}	pébibit (Pibit)	2^{50}
kilooctet (ko)	10^3	kibioctet (Kio)	2^{10}
mégaoctet (Mo)	10^6	mébioctet (Mio)	2^{20}
gigaoctet (Go)	10^9	gibioctet (Gio)	2^{30}
téraoctet (To)	10^{12}	tébioctet (Tio)	2^{40}
pétaoctet (Po)	10^{15}	pébioctet (Pio)	2^{50}

Usage traditionnel $\neq$ notations officielles : 1 ko $=$ 1024 octets ...

Exercices

- Calculer le rapport en % entre unités SI et unités binaires.
  Quelles unités les marchands ont-ils intérêt à utiliser ?

- "Programmer" à l'aide d'un tableur la conversion sur un octet

a)	base 2 $\rightarrow$ base 10	b)	base 10 $\rightarrow$ base 2
c)	base 2 $\rightarrow$ base 8	d)	base 8 $\rightarrow$ base 2
e)	base 2 $\rightarrow$ base 16	f)	base 16 $\rightarrow$ base 2

Idées :

a) Somme des puissances de 2, ou Horner
b) Div par 128, reste, div par 64, reste, etc
c) et e) En regroupant les bits
d) et f) Pour chaque chiffre, div et reste

Pour la base 16, étape intermédiaire 10 $\leftrightarrow$ A, 11 $\leftrightarrow$ B, etc.
Syntaxe : =SI(test;alors_valeur;sinon_valeur)

3 - Codage binaire des entiers et opérations

Pour les nombres entiers positifs, il est naturel de coder leur valeur en base 2.

Pour des raisons de performances, un tel entier est codé sur un mot. On peut donc représenter les entiers des intervalles

- mot de 16 bits : $[0, 2^{16} - 1]$
- mot de 32 bits : $[0, 2^{32} - 1]$
- mot de 64 bits : $[0, 2^{64} - 1]$

On ne peut donc représenter qu'un petit sous-ensemble de $\mathbb{N}$ avec ce système.

Addition des nombres positifs

Comme en base 10, avec retenue : $1 + 1 \rightarrow 0$ et on retient 1

Sens de calcul : du poids faible vers le poids fort

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

(vérification en base 10 : $86 + 27 = 113$)

La suite des opérations à effectuer constitue un *algorithme*.

Exercices : "programmer" l'addition à l'aide d'un tableur :

- avec des divisions et restes entiers ;
- avec des opérateurs logiques.

Dépassement de capacité

Quelque soit la longueur du mot, la dernière retenue dans une somme peut être perdue.

Exemple sur un octet :

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

On obtient $150 + 126 = 20$.
Ce n'est pas une erreur ; en fait le résultat est juste modulo 256.

$\rightarrow$ L'addition est faite dans $\mathbb{Z}/2^n\mathbb{Z}$ où n est la taille du mot en bits.

Entiers négatifs

Pour représenter les nombres négatifs, la première idée est d'utiliser le bit de poids fort pour le signe : 0 positif, 1 négatif.

Pour un mot de taille n, on peut représenter l'intervalle $[-2^{n-1}, 2^{n-1}]$

Inconvénient : deux représentations de zéro : $+0$ et -0

Problème : somme positif + négatif

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

On obtient $38 - 27 = -65$

Entiers négatifs : complément à 2

La solution usuelle pour représenter les entiers négatifs s'appelle le complément à 2 :

Pour un mot de taille n, on code un entier x par

- x si $x \geq 0$
- $2^n + x$ si $x < 0$ (i.e $2^n - |x|$)

On peut ainsi représenter les entiers dans $[\,-2^{n-1}\,,\ 2^{n-1}-1\,]$

Exemple : $38 - 27 = (38 + 229) \bmod 256 = 11$

	0	0	1	0	0	1	1	0
+	1	1	1	0	0	1	0	1
=	0	0	0	0	1	0	1	1

Exercices

- Calculer les intervalles représentables pour les mots de taille 8, 16, 32, 64 (pour 64, problème de précision ...).

- Trouver une méthode simple pour calculer le complément à 2 d'un entier x.

  Solution : inverser tous les bits puis ajouter 1.

- Quel est le complément à 2 de zéro ?

- "Programmer" à l'aide d'un tableur le complément à 2 en base 2.

Grands entiers

Appelons *petits entiers* les entiers représentables sur un mot et *grands entiers* ceux qui ne le sont pas.

Toutes les opérations sur les petits entiers sont "câblées" dans les microprocesseurs, de manière très efficace
($*$ et $/$ sont plus compliquées que $+$ et $-$).

Les grands entiers sont représentés par une suite de mots dans une base B, telle que $B-1$ soit représentable sur un mot de taille n.

Exemple : $B = 2^n$ (sur entiers non signés) ou $B = 2^{n-1}$ (signés).

On a alors $x = \sum_{i=0}^{k} x_i B^i$ où k est le nombre de mots nécessaires pour représenter x.

- Avantage : on peut utiliser les opérations câblées sur les x_i.
- Attention aux débordements !
- En pratiques, $\exists$ logiciels tout faits, très efficaces.

Utilisation des grands entiers

Exemples :

- Cryptographie : fabrication de grands nombres premiers (tests de primalité probabilistes de Solovay-Strassen et de Miller-Rabin).
- Cryptanalyse : factorisation de grands nombres premiers.
- Calcul des décimale de π.
- Calcul des décimales de $\sqrt{2}$.

Le calcul de π (1/3)

Babylonians	2000? BCE	1	$3.125 = 3 + 1/8$
Egyptians	2000? BCE	1	3.16045
China	1200? BCE	1	3
Bible (1 Kings 7:23)	550? BCE	1	3
Archimedes	250? BCE	3	3.1418 (ave.)
Hon Han Shu	130 AD	1	$3.1622 = \sqrt{10}$?
Ptolemy	150	3	3.14166
Tsu Ch'ung Chi	480?	7	3.1415926
Aryabhata	499	4	3.14156
Brahmagupta	640?	1	$3.162277 = \sqrt{10}$?
Al-Khowarizmi	800	4	3.1416
Fibonacci	1220	3	3.141818
Al-Kashi	1429	14	
Viete	1593	9	3.1415926536 (ave.)
Van Ceulen	1615	35	
Newton	1665	16	
Machin	1706	100	
De Lagny	1719	127	(112 corrects)

Le calcul de π (2/3)

Rutherford	1824	208	(152 corrects)
Shanks	1874	707	(527 corrects)
Ferguson	1946	620	
Smith and Wrench	1949	1 120	
Reitwiesner et al. (ENIAC)	1949	2 037	
Nicholson and Jeenel	1954	3 092	
Genuys	Jan. 1958	10 000	
Shanks and Wrench	1961	100 265	
Guilloud and Dichampt	1967	500 000	
Guilloud and Bouyer	1973	1 001 250	
Kanada, Yoshino and Tamura	1982	16 777 206	
Ushiro and Kanada	Oct. 1983	10 013 395	
Kanada, Tamura, Kubo et al	Jan. 1987	134 217 700	
Chudnovskys	Aug. 1991	2 260 000 000	
Takahashi and Kanada	Oct. 1995	6 442 450 938	
Kanada	Dec. 2002	1 241 100 000 000	
Fabrice Bellard	Dec. 2009	2 700 000 000 000	PC standard
A.J. Yee and Shigeru Kondo	Aug. 2010	5 000 000 000 000	PC 24 cœurs

Le calcul de π (3/3)

Quelques réflexions :

- Calcul et vérifications de F. Bellard : 131 jours sur un simple PC ; espace de stockage nécessaire 1137 Go.
- En 60 ans, on est passé de 2.10^3 à 5.10^{12} décimales : un facteur 2.10^9 !
- On est passé de super-calculateurs extrêmement chers à un simple PC.
- Progrès énormes dans le matériel, les algorithmes, les optimisations, les formules mathématiques employées (formules de Machin, de Chudnovsky ...).

$\sqrt{2}$ avec 10 000 décimales (1/2)

On va utiliser la formule classique :

$$x_{n+1} = \frac{1}{2}(x_n + \frac{2}{x_n})$$

Convergence rapide (quadratique). Si $x_n = \sqrt{2} + \epsilon_n$ alors :

$$\epsilon_{n+1} = \frac{1}{2}\frac{\epsilon_n^2}{\sqrt{2} + \epsilon_n}$$

En fait on calcule $u_n = 10^{10000} x_n$. La formule s'écrit alors :

$$u_{n+1} = \frac{1}{2}(u_n + \frac{2 \times 10^{20000}}{u_n})$$

On peut prendre $u_0 = 10^{10000}$ et dans ce cas 14 itérations suffisent.

$\sqrt{2}$ avec 10 000 décimales (2/2)

Voici le programme en Python (supporte les grands entiers).

```
def rac2(ndec,niter):
    u = 10**ndec
    deux = 2*10**(2*ndec)
    for i in range(niter):
        u = (u + deux/u)/2   # Division entière
    return u

print rac2(10000,14)
```

$\rightarrow$ Résultat immédiat :

14142135623730950488016887242096980785696718753769480731766797379907324784621070388503875343276415727350138462309122970249248360558507372126441214970999358314132226659275...

4 - Représentation des nombres réels

Commençons par les rationnels, codés comme un couple d'entier :

- Calculs exacts.
- Problème des fractions irréductibles : à chaque calcul il faut simplifier à l'aide du pgcd.
- Difficulté du calcul dû à la croissance des nombres dans des opérations simples.

Comment coder les irrationnels ? Par un rationnel proche ?

On ne peut pas coder *tous* les réels ; par contre, on peut découper $\mathbb{R}$ en une suite finie d'intervalles, puis travailler avec ces intervalles. Par exemple : $[0]\,,\ ]0\ldots1[\,,\ [1]\,,\ ]1\ldots+\infty[$

La représentation décimale

On peut représenter les rationnels par la suite des chiffres décimaux.

Suivant la base, la suite des chiffre décimaux est finie ou infinie. Si le nombre est rationnel, alors cette suite est périodique.

- $1/2 = 0{,}5$
- $1/7 = 0{,}142857\ 142857\ 142857\ 142857\ ....$

Problèmes :

- La période peut être très grande $\rightarrow$ grands entiers.
- 2 et 1,99999999999... sont identiques.
- Si toutes les décimales sont connues, le nombre n'est pas forcément rationnel : 0,123456789101112131 4....

Représentation en virgule fixe

Représentation par un entier divisé par une constante fixée S.

Par exemple, $1,23$ peut être représenté par 1230 si $S = 1000$.

En général S est une puissance de 2 (pour le calcul par ordinateur), une puissance de 10 (pour l'humain), parfois 3600 pour le calcul en secondes.

Lorsque S est une puissance de 2, le premier bit de la partie décimale représente $1/2$, le second $1/4$, etc.

Utilisé pour les calculs en précisions fixe, et dans les processeurs à faible coût.

Représentation en virgule flottante

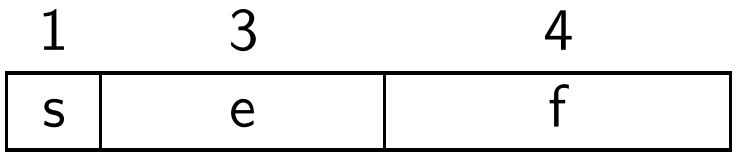
Pour les besoins du calcul numériques, il faut beaucoup de nombres autour de 0.

Les flottants sont un ensemble fini de rationnels, de la forme $a/2^n$; on considère que ces flottants séparent $\mathbb{R}$ en intervalles, dont ils sont les bornes inférieures.

Ils sont choisis de telle manière qu'il y ait autant de flottants entre 0 et 1 qu'entre 1 et 2, entre 2 et 4, entre 4 et 8, etc.

Codage binaire par un bit de signe, un exposant e et une mantisse f.

Les flottants — un exemple (1/2)



exposant e	mantisse f	valeur v	type de v
$0 < e < 7$		$v = (-1)^s \times 2^{(e-3)} \times (1.f)$	Normalisé
$e = 0$	$f \neq 0$	$v = (-1)^s \times 2^{(-2)} \times (0.f)$	Dénormalisé
$e = 0$	$f = 0$	$v = (-1)^s \times 0$	Zéro
$e = 7$	$f = 0$	$v = (-1)^s \times \mathit{infini}$	Infini
$e = 7$	$f \neq 0$	v n'est pas un nombre	NaN

Normalisé signifie que l'on insère 1 devant la mantisse

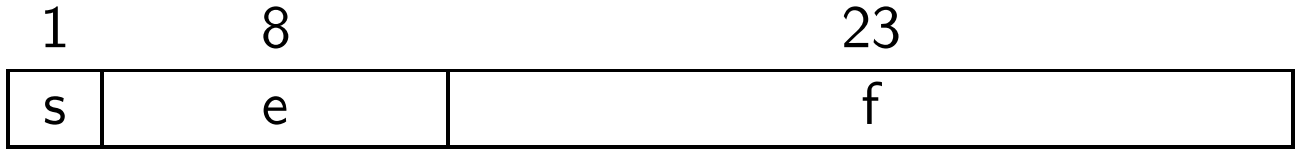
	valeur exacte	valeur décimale
Nombre maximum :	$2^4 - 2^{-1}$	15.5
Nombre minimum (normalisé) :	2^{-2}	0.25
Nombre minimum (dénormalisé) :	2^{-6}	0.015625
Entier maximum :	$2^4 - 1$	15

Les flottants — un exemple (2/2)

f	$e=0$	$e=1$	$e=2$	$e=3$	$e=4$	$e=5$	$e=6$	$e=7$
0	0	16/64	16/32	1	2	4	8	$+\infty$
1	1/64	17/64	17/32	17/16	17/8	17/4	17/2	NaN
2	2/64	18/64	18/32	18/16	18/8	18/4	9	NaN
3	3/64	19/64	19/32	19/16	19/8	19/4	19/2	NaN
4	4/64	20/64	20/32	20/16	20/8	5	10	NaN
5	5/64	21/64	21/32	21/16	21/8	21/4	21/2	NaN
6	6/64	22/64	22/32	22/16	22/8	22/4	11	NaN
7	7/64	23/64	23/32	23/16	23/8	23/4	23/2	NaN
8	8/64	24/64	24/32	24/16	3	6	12	NaN
9	9/64	25/64	25/32	25/16	25/8	25/4	25/2	NaN
10	10/64	26/64	26/32	26/16	26/8	26/4	13	NaN
11	11/64	27/64	27/32	27/16	27/8	27/4	27/2	NaN
12	12/64	28/64	28/32	28/16	28/8	7	14	NaN
13	13/64	29/64	29/32	29/16	29/8	29/4	29/2	NaN
14	14/64	30/64	30/32	30/16	30/8	30/4	15	NaN
15	15/64	31/64	31/32	31/16	31/8	31/4	31/2	NaN

IEEE Single

Norme IEEE 754 pour les flottants simple précision (32 bits) et double précision (64 bits).



exposant e	mant. f	valeur v	type de v
$0 < e < 255$		$v = (-1)^s \times 2^{(e-127)} \times (1.f)$	Normalisé
$e = 0$	$f \neq 0$	$v = (-1)^s \times 2^{(-126)} \times (0.f)$	Dénormalisé
$e = 0$	$f = 0$	$v = (-1)^s \times 0$	Zéro
$e = 255$	$f = 0$	$v = (-1)^s \times \textit{infini}$	Infini
$e = 255$	$f \neq 0$	v n'est pas un nombre	NaN

	valeur exacte	valeur décimale
Nb maximum :	$2^{128} - 2^{104}$	3.402823466E+38
Nb minimum (normalisé) :	2^{-126}	1.175494351E-38
Nb minimum (dénormalisé) :	2^{-149}	1.401298464E-45
Ent. maximum :	$2^{24} - 1$	16 777 215

Pour réfléchir (1/2)

On veut calculer u_{100} avec:

$$u_n = 4u_{n-1} - 1 \qquad u_0 = 1/3$$

```
def recurrence(n):
    u0 = 1.0/3
    u1 = 0
    for i in range(n):
        u1 = 4*u0 - 1
        u0 = u1
    return u1

print recurrence(100)
```

On trouve : -2,97343269313741 63e+43 : Pourquoi?

Pour réfléchir (2/2)

0	0	24	0.328125	48	-1.4660155037e+12
1	0.333333333333	25	0.3125	49	-5.86406201480e+12
2	0.333333333333	26	0.25	50	-2.34562480592e+13
3	0.333333333333	27	0.0	51	-9.38249922369e+13
4	0.333333333333	28	-1.0	52	-3.75299968948e+14
5	0.333333333333	29	-5.0	53	-1.50119987579e+15
6	0.333333333333	30	-21.0	54	-6.00479950316e+15
7	0.333333333333	31	-85.0	55	-2.40191980126e+16
8	0.333333333332	32	-341.0	56	-9.60767920506e+16
9	0.333333333328	33	-1365.0	57	-3.84307168202e+17
10	0.333333333314	34	-5461.0	58	-1.53722867281e+18
11	0.333333333256	35	-21845.0	59	-6.14891469124e+18
12	0.333333333023	36	-87381.0	60	-2.45956587649e+19
13	0.333333332092	37	-349525.0	61	-9.83826350598e+19
14	0.333333328366	38	-1398101.0	62	-3.93530540239e+20
15	0.333333313465	39	-5592405.0	63	-1.57412216096e+21
16	0.33333325386	40	-22369621.0	64	-6.29648864383e+21
17	0.333333015442	41	-89478485.0	65	-2.51859545753e+22
18	0.333332061768	42	-357913941.0	66	-1.00743818301e+23
19	0.33332824707	43	-1431655765.0	67	-4.02975273205e+23
20	0.333312988281	44	-5726623061.0	68	-1.61190109282e+24
21	0.333251953125	45	-22906492245.0	69	-6.44760437128e+24
22	0.3330078125	46	-91625968981.0	...	...
23	0.33203125	47	-3.66503875925e+11	100	-2.97343269314e+43

5 - Codage numérique du texte (standards)

Le texte est constitué de caractères. Chaque caractère est représenté par un entier (sur un certain nombre d'octets).

Il existe de nombreux codages des caractères ; les principaux codages pour les occidentaux sont :

- Le code ASCII (ISO 646)
- Les codes ISO 8859-1 / ISO 8859-15 (symbole €)
- Le code Unicode
- Les codes UTF-8 / UTF-16 / UTF-32

ASCII/ISO 646

- ASCII (American Standard Code for Information Interchange), Bob Bemer en 1961. C'est un code sur 7 bits, le premier bit étant 0 (Dans le temps on utilisait ce bit comme bit de parité).
- Les caractères de 0 à 31 ainsi que le 127 ne sont pas affichables, et correspondent à des directives de terminal. Le caractère 32 est l'espace blanc. Les autres correspondent aux chiffres, aux lettres majuscules et minuscules et à quelques symboles de ponctuation. Problème LF/CR.
- ASCII : norme américaine, la norme officielle en France = ISO 646 (pas de variantes nationales).
- Problèmes : Pas d'accents et de nombreuses variantes pour utiliser le 8e bit (Mac, IBM, ...)

ISO 8859-1

- L'ISO 8859-1 [Ou Latin-1] (1992) recouvre les caractères utilisés par les langues européennes suivantes : albanais, allemand, anglais, basque, catalan, danois, gaélique écossais, espagnol, féringien, finnois, français. Code sur 8 bits.
- Code 191 caractères. Mais les caractères suivants ont été oubliés : œ, Œ et Ÿ. Et bien sûr il manquait le caractère € de l'euro !! Pour corriger cela $\rightarrow$ code ISO 8859-15
- Mais il fallait tout une collection de codes pour chaque groupe de pays, par exemple ISO 8859-2 (latin-2 ou européen central), ISO 8859-7 (grec), ... Ne parlons pas du Japon et de la Chine !!
- En 2004, le groupe de normalisation a décidé d'abandonner ce code au profit de l'Unicode et de l'UTF-8.

Unicode

- Unicode, dont la première publication remonte à 1991, a été développé dans le but de remplacer l'utilisation de pages de code nationales.
- Jeu de caractères abstraits : La couche la plus élevée est la définition du jeu de caractères. Par exemple, Latin-1 a un jeu de 256 caractères et Unicode normalise actuellement près de 100 000 caractères.
- Jeu de caractères codés : on ajoute à la table précédente un index numérique. Notons bien qu'il ne s'agit pas d'une représentation en mémoire, juste d'un nombre (de 4 à 6 caractères hexadécimaux).
- Exemples :
  - é : *latin small letter e with acute* (hexa = E9) é or é [ISO 8859-15 : E9]
  - œ: *latin small ligature oe* – (hexa = 153) œ or œ [ISO 8859-15 : BD]

UTF-8

Codage des caractères Unicode sur plusieurs octets. (Il existe d'autres formats comme UTF-16 et UTF-32).

0vvvvvvv	1 octet codant 1 à 7 bits
110vvvvv 10vvvvvv	2 octets codant 8 à 11 bits
1110vvvv 10vvvvvv 10vvvvvv	3 octets codant 12 à 16 bits

Quelques exemples :

é	E9	1110 1001	11000011 10101001
œ	153	1 0101 0011	11000101 10010011
€	20AC	10 0000 1010 1100	11100010 10000010 10101100

- Astucieux : compatible ASCII 7 bits.
- On ne peut plus déduire la longueur d'une chaîne, en comptant le nombre d'octets.
- Il faut connaître le type de codage d'un texte.

MIME

Multipurpose Internet Mail Extensions (MIME) est un format de données permettant d'introduire dans les messages SMTP (courriels) différents types de fichiers multimédias (et en particulier des textes).

On peut utiliser base64 : codage de l'information utilisant 64 caractères (lettres majuscules, lettres minuscules, chiffres, '+' et '/') et terminé par '='.

- On organise la suite d'octets par tranches de trois octets (24 bits).
- Chaque groupe de 3 octets est traduit par 4 caractères.

C'est ainsi que l'on code par exemple les images dans les mails (augmentation de la taille par un facteur 4/3) ; on peut aussi coder des textes en ISO 8859 ou UTF-8.

Un exemple

Indication du codage : entête XML/HTML

```
<?xml version="1.0" encoding="UTF-8"?>

<meta http-equiv="content-type" content="text-html;
    charset=UTF-8">
```

Regardons différents codages du mot œuvrée :

HTML	œuvrée
ISO 8859-15	BD 75 76 72 E9 65
UTF-8	C5 93 75 76 72 C3 A9 65
base64	xZN1dnLDqWU=

Conclusion : interprétations du même mot

Le mot

11000101 10010011 01110101 01110110
01110010 11000011 10101001 01100101

peut être interprété comme :

hexadécimal	0xC593757672C3A965
entier	14236851998640286053
entier signé	-4209892075069265563
flottant	-1.5055547159432955 E27
ISO 8859-15	Å“uvrÃ©e
UTF-8	œuvrée

→ Connaître le codage binaire d'un mot n'est pas suffisant, il faut aussi connaître la représentation et le codage employés.