

Université Mohammed V
Faculté des Sciences
Département d'Informatique

Cours M6 pour SMIA

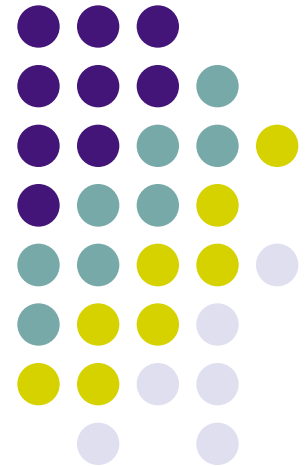
Introduction à l'Informatique

M. El Marraki

N. El Khattabi

2020 – 2021

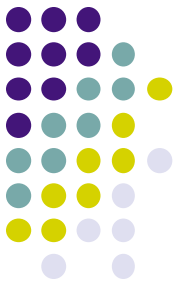
Cours N°8



Sommaire



- I. La Filière SMIA (SMI / SMA)
- II. Histoire de l'informatique et Structure des ordinateurs
- III. Histoires des Langages de programmation
- IV. Algèbre de Boole
- V. **Le codage**
 - Introduction
 - Système de numération décimale, binaire, octale et hexadécimale
 - Codage des nombres entiers
 - **Codage des nombres réels**
 - Codage des caractères
 - Codages des images et du son
- VI. Le langage HTML

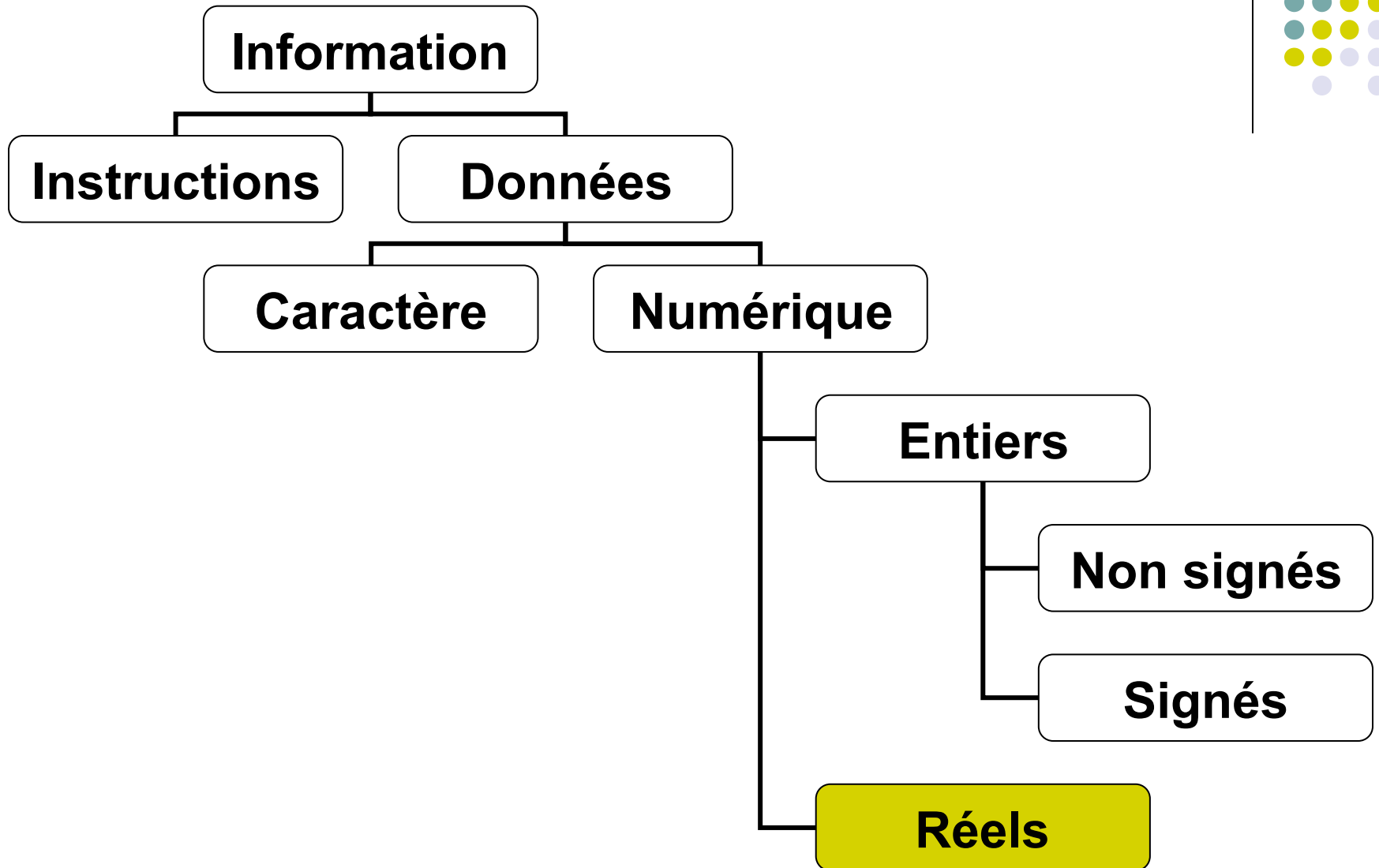


V. Le codage

Introduction

Système d'énumération

Codage des nombres entiers





Codage des nombres réels

Les formats de représentations des nombres réels sont :

- Format **virgule fixe**
 - utilisé par les premières machines
 - possède une partie « **entière** » et une partie « **décimal** » séparés par une virgule. La position de la virgule est fixe d'où le nom.
 - Exemples : **$54,25_{(10)}$** ; **$10,001_{(2)}$** ; **$A1,F0B_{(16)}$**



Codage des nombres réels

Format **virgule flottante** (utilisé actuellement sur machine)

défini par : $\pm m . b^e$

- ✓ un signe $+$ ou $-$
- ✓ une mantisse m (en virgule fixe)
- ✓ un exposant e (un entier relative)
- ✓ une base b (2, 8, 10, 16, ...)
- ✓ Exemple : $0,5425 . 10^2_{(10)}$; $10,1 . 2^{-1}_{(2)}$;
 $A0,B4.16^{-2}_{(16)}$



Codage en Virgule Fixe

Etant donné une base b

➤ un nombre x est représenté par :

- $x = a_{n-1}a_{n-2}\dots a_1a_0, a_{-1}a_{-2}\dots a_{-p} (b)$
- a_{n-1} est le chiffre de poids fort
- a_{-p} est le chiffre de poids faible
- n est le nombre de chiffre avant la virgule
- p est le nombre de chiffre après la virgule

- La valeur de x en base 10 est : $x = \sum_{i=-p}^{n-1} a_i b^i_{(10)}$
- Exemple :

$$101,01_{(2)} = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 5,25_{(10)}$$

Codage en Virgule Fixe

Changement de base 10→2



Le passage de la base 10 à la base 2 est défini par :

- Partie **entière** est codée sur **p bits** (division successive par 2)
- Partie **décimale** est codée sur **q bits** en **multipliant par 2** successivement jusqu'à ce que la partie **décimale** soit **nulle** ou le nombre de bits **q** est atteint.

- **Exemple** : $4,25_{(10)} = ?_{(2)}$ format virgule fixe

✓ $4_{(10)} = 100_{(2)}$

$$0,25 \times 2 = 0,5 \rightarrow 0$$

$$0,5 \times 2 = 1,0 \rightarrow 1$$

✓ donc $4,25_{(10)} = 100,01_{(2)} = 2^2 + 2^{-2}$

Exercice : Coder $7,875_{(10)}$ et $5,3_{(10)}$ avec $p = 8$ et $q = 8$

Codage en Virgule Fixe

Changement de base 10→2



Coder : $7,875_{(10)} = ?_{(2)}$ format virgule fixe $p=q=8$

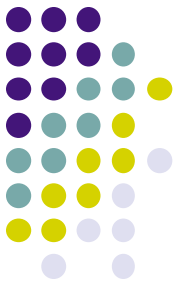
✓ $7_{(10)} = 111_{(2)}$

$0,875 \times 2 = 1,75 \rightarrow 1$

$0,75 \times 2 = 1,5 \rightarrow 1$

$0,5 \times 2 = 1,0 \rightarrow 1$

✓ Donc $7,875_{(10)} = 111,111_{(2)}$ ($= 2^2 + 2^1 + 2^0 + 2^{-1} + 2^{-2}$)



Codage en Virgule Flottante

$$x = \pm M \cdot 2^E$$

où **M** est la mantisse (**virgule fixe**) et **E** l'exposant (signé).

Le codage en base 2, format virgule flottante, revient à coder le **signe**, la **mantisse** et l'**exposant**.

Exemple : Codage en base 2, format virgule flottante, de (3,25)

$$\begin{aligned} 3,25_{(10)} &= 11,01_{(2)} && \text{(en virgule fixe)} \\ &= 1,101 \cdot 2^1_{(2)} \\ &= 110,1 \cdot 2^{-1}_{(2)} \end{aligned}$$

Pb : différentes manières de représenter **E** et **M**

→ **Normalisation**

Codage en Virgule Flottante

- Normalisation -



$$x = \pm 1, M \cdot 2^{E_b}$$

Le **signe** (SM) est codé sur **1 bit** ayant le **poids fort** :

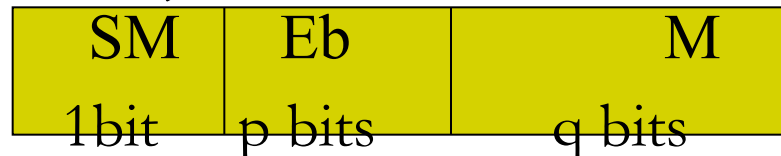
- le signe - : bit 1
- Le signe + : bit 0

Exposant biaisé (Eb)

- placé avant la mantisse pour simplifier la comparaison
- Codé sur **p bits** et biaisé pour être positif (ajout de $2^{p-1}-1$)

Mantisse normalisé(M)

- Normalisé : virgule est placé après le bit à 1 ayant le poids fort
- M est codé sur **q bits**
- Exemple : 11,01 \rightarrow 1,101 donc **M = 101**



Standard IEEE 754 (1985)

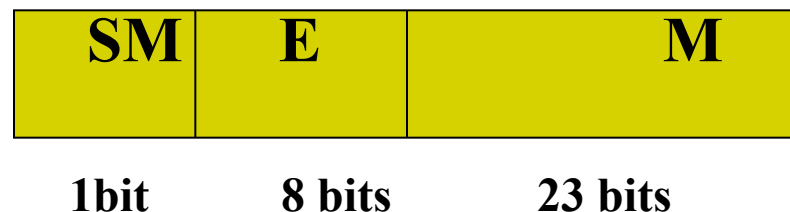


Simple précision sur 32 bits :

1 bit de signe de la mantisse

8 bits pour l'exposant

23 bits pour la mantisse

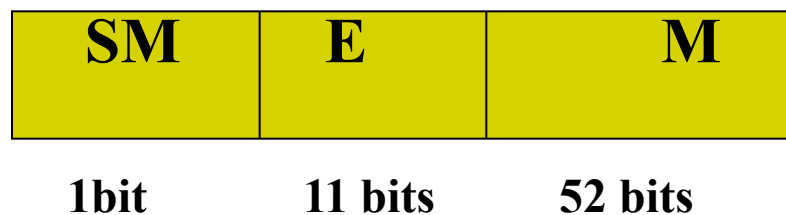


Double précision sur 64 bits :

1 bit de signe de la mantisse

11 bits pour l'exposant

52 bits pour la mantisse



Conversion décimale - IEEE754 (Codage d'un réel)



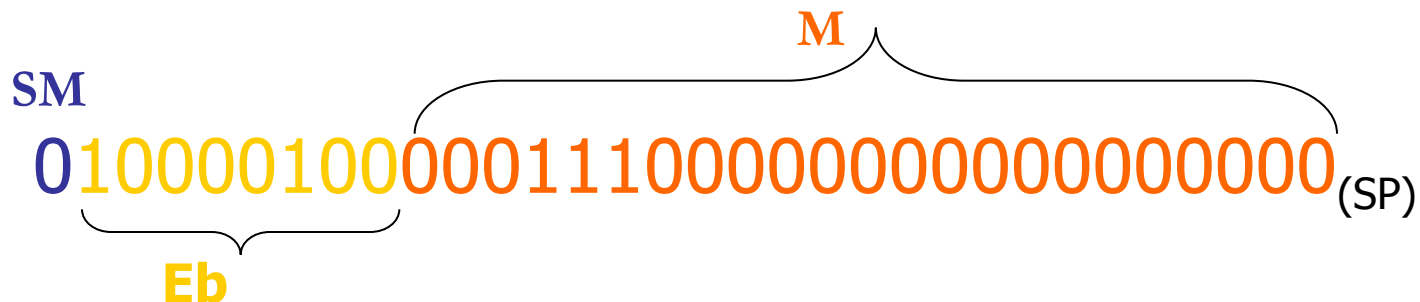
$$35,5_{(10)} = ?_{(\text{IEEE 754 simple précision})}$$

Nombre positif, donc SM = 0

$$\begin{aligned} 35,5_{(10)} &= 100011,1_{(2)} \quad (\text{virgule fixe}) \\ &= 1,000111 \cdot 2^5_{(2)} \quad (\text{virgule flottante}) \end{aligned}$$

$$\text{Exposant} = E_b - 127 = 5, \text{ donc } E_b = 132$$

$$1, M = 1,000111 \quad \text{donc } M = 00011100\dots$$



Conversion décimale - IEEE754 (Codage d'un réel)



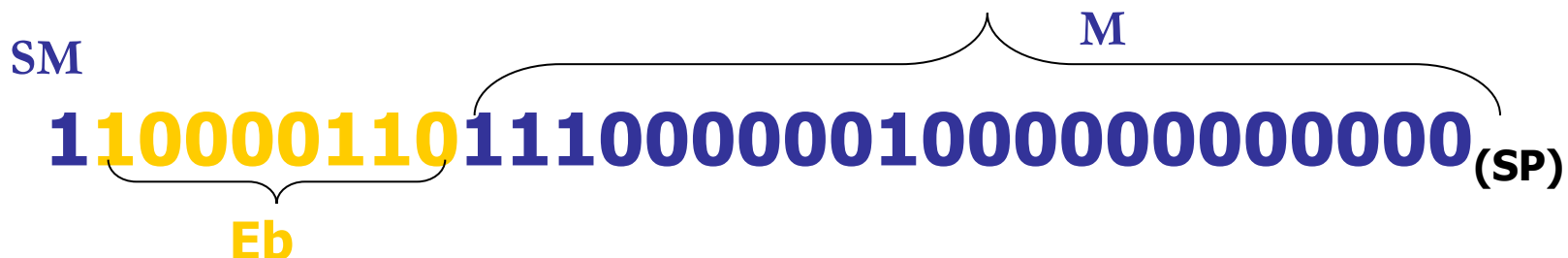
$$\textcolor{red}{-240.125}_{(10)} = ?_{(SP)}$$

Nombre négatif, donc SM = 1

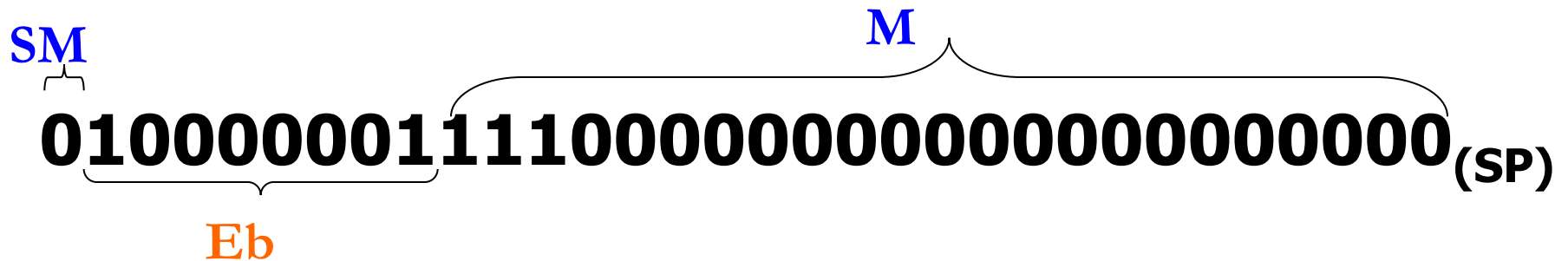
$$\begin{aligned} -240,125_{(10)} &= -11110000,001_{(2)} \text{ (virgule fixe)} \\ &= 1,1110000001 \cdot 2^7_{(2)} \text{ (virgule flottante)} \end{aligned}$$

$$\begin{aligned} \text{Exposant : } E_b &= 127 + 7 = 134 = 128 + 6 \\ &= 10000110_2. \end{aligned}$$

● 1 10000110 111000000100...0



Conversion IEEE754 - décimale (Évaluation d'un réel)



0 10000001 111000000000000000000000

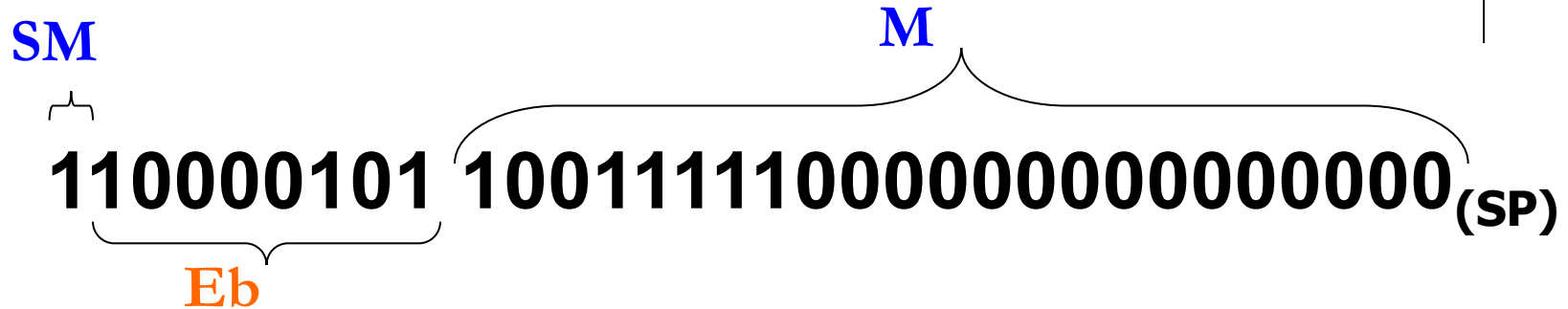
SM = 0, donc nombre positif

Eb = 129, donc exposant = Eb-127 = 2

1,M = 1,111

+ 1,111 . 2²₍₂₎ = 111,1₍₂₎ = 7,5₍₁₀₎

Conversion IEEE754 - décimale (Évaluation d'un réel)



0 10000001 111000000000000000000000

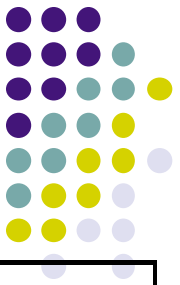
$$E_b = 10000101_2 = 128 + 5 = 133,$$

$$\text{donc } E = 133 - 127 = 6$$

$$x = -1,10011111 \times 2^6 = -1100111,11$$

$$1100111_2 = 103 \text{ et } 0,11_2 = 0,75$$

$$\text{donc } x = -103,75$$



1. Évaluer le nombre réel en format IEEE 754 simples précisions :

1 10000101 100111110000000000000000

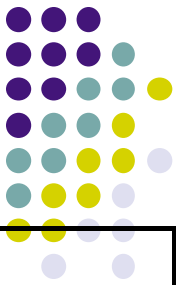
$$E_b = 10000101_2 = 128 + 5 = 133,$$

$$\text{donc } E = 133 - 127 = 6$$

$$x = -1,10011111 \times 2^6 = -1100111,11$$

$$1100111_2 = 103 \text{ et } 0,11_2 = 0,75$$

$$x = -103,75$$



1. Convertir le nombre réel **-240.125** dans le format IEEE 754 **simples précisions**.

$$240 = 120 \times 2 = 11110000_2.$$

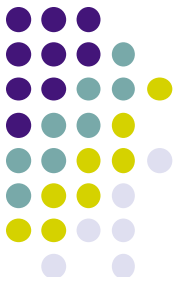
$$0,125 = 0,001_2.$$

$$\begin{aligned} -240,125 &= -11110000,001 \\ &= -1,1110000001 \times 2^7. \end{aligned}$$

$$\begin{aligned} E_b &= 127 + 7 = 134 = 128 + 6 \\ &= 10000110_2. \end{aligned}$$

$$1 \ 10000110 \ 111000000100...0$$

Caractéristiques des nombres flottants au standard IEEE

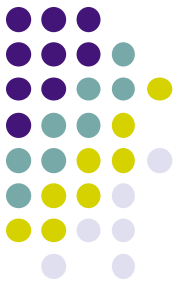


	Simple précision	Double précision
Bit de signe	1	1
Bit d'exposant	8	11
Bit de mantisse	23	52
Nombre total de bits	32	64
Codage de l'exposant	Excédant 127	Excédant 1023
Variation de l'exposant	-126 à +127	-1022 à +1023
Plus petit nombre normalisé	2^{-126}	2^{-1022}
Plus grand nombre normalisé	Environ 2^{+128}	Environ 2^{+1024}
Echelle des nombre décimaux	Environ 10^{-38} à 10^{+38}	Environ 10^{-308} à 10^{+308}
Plus petit nombre dénormalisé	Environ 10^{-45}	Environ 10^{-324}

Sommaire



- I. La Filière SMIA (SMI / SMA)
- II. Histoire de l'informatique et Structure des ordinateurs
- III. Histoires des Langages de programmation
- IV. Algèbre de Boole
- V. **Le codage**
 - Introduction
 - Système de numération décimale, binaire, octale et hexadécimale
 - Codage des nombres entiers
 - Codage des nombres réels
 - **Codage des caractères**
 - Codages des images et du son
- VI. Le langage HTML



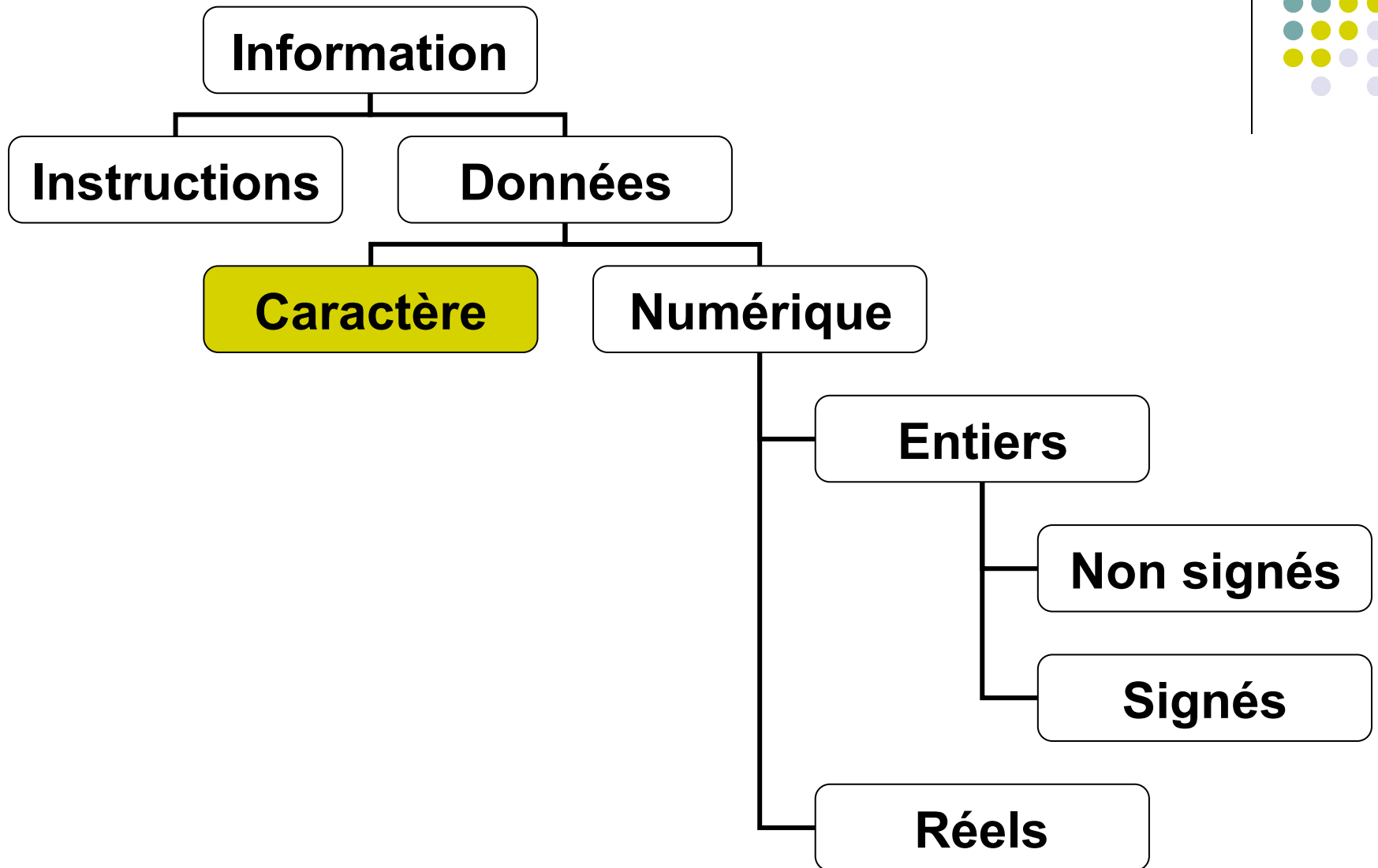
V. Le codage

Introduction

Système d'énumération

Codage des nombres réels

Codage des caractères





Codage des caractères

Caractères : **Alphabétique** (A-Z , a-z), **numérique** (0 ,..., 9), **ponctuation, spéciaux** (&, \$, %, ...) ...
etc.

Données **non numérique** (addition n' a pas de sens)

Comparaison ou **tri** → **très utile**

Codage revient à créer une **Table de correspondance** entre les **caractères** et **des nombres**.

Codage des caractères

Les standards



Code (ou Table) **ASCII** (American Standard Code for Information Interchange)

- **7 bits** pour représenter **128** caractères (0 à 127)
- **48 à 57** : **chiffres** dans l'ordre (0,1,...,9)
- **65 à 90** : les **alphabets majuscules** (A ,..., Z)
- **97 à 122** : les **alphabets minuscule** (a ,..., z)

Codage des caractères

Les standards



Table **ASCII Etendu**

- 8 bits pour représenter 256 caractères (0 à 255)
- Code les caractères accentués : à, è,...etc.
- Compatible avec ASCII

Code **Unicode** (mis au point en 1991)

- 16 bits pour représenter 65 536 caractères (0 à 65 535)
- Compatible avec ASCII
- Code la plupart des alphabets : Arabe, Chinois,
- On en a défini environ 50 000 caractères pour l' instant






















Code ASCII Etendu



DECIMAL VALUE	➡	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
↩	HEXA DECIMAL VALUE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	BLANK (NULL)	▶	SP	0	@	P	‘	p	Ç	É	á	⋮	⌌	⌌	∞	≡
1	1	😊	◀	!	1	A	Q	a	q	ü	æ	í	⋮	⌌	⌌	β	±
2	2	😬	↑	"	2	B	R	b	r	é	Æ	ó	⋮	⌌	⌌	Γ	≥
3	3	♥	!!	#	3	C	S	c	s	â	ô	ú	⋮	⌌	⌌	π	≤
4	4	♦	¶	\$	4	D	T	d	t	ä	ö	ñ	⋮	⌌	⌌	Σ	∫
5	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	⋮	⌌	⌌	σ	∫
6	6	♠	—	&	6	F	V	f	v	å	û	ä	⋮	⌌	⌌	μ	÷
7	7	BEL	↓	'	7	G	W	g	w	ç	ù	ó	⋮	⌌	⌌	τ	≈
8	8	BS	↑	(8	H	X	h	x	ê	ÿ	ı	⋮	⌌	⌌	ϕ	°
9	9	HT	↓)	9	I	Y	i	y	ë	Ö	⌌	⋮	⌌	⌌	θ	•
10	A	LF	→	*	:	J	Z	j	z	è	Ü	⌌	⋮	⌌	⌌	Ω	•
11	B	VT	←	+	;	K	I	k	{	ï	ç	½	⋮	⌌	⌌	δ	√
12	C	FF	FS	,	<	L	\	l	!	î	£	¼	⋮	⌌	⌌	∞	n
13	D	CR	GS	—	=	M	J	m	}	ì	¥	ı	⋮	⌌	⌌	φ	²
14	E	♪	RS	.	>	N	^	n	~	Ä	Ŕ	«	⋮	⌌	⌌	€	■
15	F	⚙	US	/	?	O	—	o	Δ	Å	ƒ	»	⋮	⌌	⌌	∩	NAME

Unicode

پ	ڈ	غ	گ	ق	ی	ُ	۔
0680	0690	06A0	06B0	06C0	06D0	06E0	06F0
خ	ز	ف	ح	ج	پ	و	ا
0681	0691	06A1	06B1	06C1	06D1	06E1	06F1
ط	ر	ب	ک	ل	م	ن	ی
0682	0692	06A2	06B2	06C2	06D2	06E2	06F2

						
2600	2610	2620	2630	2640	2650	2660
						
2601	2611	2621	2631	2641	2651	2661
						
2602	2612	2622	2632	2642	2652	2662

Є	Д	Φ	д	φ	€	Љ	ѐ
0404	0414	0424	0434	0444	0454	0464	0474
Š	Е	Х	е	х	š	ћ	ѝ
0405	0415	0425	0435	0445	0455	0465	0475
І	Ж	Ц	ж	ц	і	А	ѐ
0406	0416	0426	0436	0446	0456	0466	0476

Ce ne sont que des bits !!!



01001001 01001110 01000110 01001111
01010010 01001101 01000001 01010100
01001001 01010001 01010101 01000101

Ce ne sont que des bits !!!



Caractères codés en ASCII Etendu (8 bits)

01001001 01001110 01000110 01001111 01010010 01001101
01000001 01010100 01001001 01010001 01010101 01000101

INFORMATIQUE

Ce ne sont que des bits !!!



Entiers codés en binaire pur sur 1 octets

01001001 01001110 01000110 01001111 01010010 01001101
01000001 01010100 01001001 01010001 01010101 01000101

73 ; 78 ; 70 ; 79 ; 82 ; 77 ;
65 ; 84 ; 73 ; 81 ; 85 ; 69 (base 10)

Ce ne sont que des bits !!!



Entiers codés en binaire pur sur 2 octets

01001001 01001110 01000110 01001111 01010010 01001101
01000001 01010100 01001001 01010001 01010101 01000101

18766 ; 17999 ; 21069 ;
16724 ; 18769 ; 21829 (base 10)

Ce ne sont que des bits !!!



Entiers codés en binaire pur sur 4 octets

01001001 01001110 01000110 01001111 01010010 01001101
01000001 01010100 01001001 01010001 01010101 01000101

1 229 866 575 ;

1 380 794 708 ;

1 230 067 013 (base 10)

Ce ne sont que des bits !!!



Nombres en flottant simple précision (32 bits)

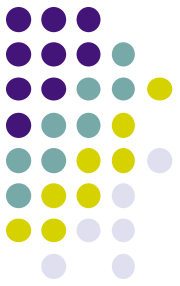
01001001 01001110 01000110 01001111 01010010 01001101
01000001 01010100 01001001 01010001 01010101 01000101

$$\begin{aligned} &+ (1,10011100100011001001111) \cdot 2^{19} ; \\ &+ (1,10011010100000101010100) \cdot 2^{37} ; \\ &+ (1,10100010101010101000101) \cdot 2^{19} ; \end{aligned}$$

844 900,9375;

220 391 079 936 ;

857 428,3125 (base 10)



Fin du cours