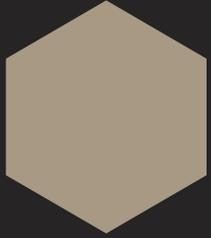




Tu connais ce type ?

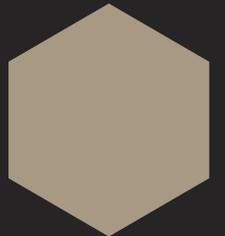




Après des années à arpenter sa stack favorite, on se surprend toujours à découvrir un comportement abscons au détour d'une ligne de code.

Plusieurs jours peuvent passer avant de réaliser qu'on ne comprenait pas vraiment les types de données qu'on manipulait en toute bonne foi.

PRÉAMBULE



SOMMAIRE

- Les nombres
 - Les entiers
 - Les nombres à virgule flottante
- Les dates
- Les chaînes de caractère
 - Unicode
 - Chaînes et langages de programmation
- Les URLs



Les nombres





LES ENTIERS



Des entiers bornés

- Nombre de bits couramment utilisés
 - 8 bits.....256 valeurs
 - 16 bits.....65 536 valeurs
 - 32 bits.....4 294 967 296 valeurs
 - 64 bits.....18 446 744 073 709 551 616 valeurs
~18 milliards de milliards de valeurs
- De nombreuses variantes
 - 7, 10, 12, 15, 18, 20, 24, 36, 48, 80, 128 bits...



Des entiers non bornés (ou presque)

- GNU MP partout...
 - Python
 - Haskell
 - Maple
 - Mathematica
 - etc.
- Ou presque
 - JavaScript
- Avantage
 - peu de limite
- Inconvénients
 - occupe plus de mémoire
 - plus lent
 - gestion mémoire délicate



Le complément à deux

- Technique la plus utilisée pour les entiers signés

	<i>non signé</i>	<i>signé</i>
– 8 bits	[0..255]	[-128..127]
– 16 bits	[0..65 535]	[-32 768..32 767]
– 32 bits	[0..4 294 967 296]	[-2 147 483 648..2 147 483 647]

- Distinction gérée par le programme !



À chaque langage ses variantes

	Signé	Non signé	Borné	Non borné	Bits
C	✓	✓	✓		8, 16, 32, 64
PHP	✓		✓		32 ou 64
Python 2	✓		✓	✓	32 ou 64
Python 3	✓			✓	n/a
Haskell	✓		✓	✓	64
JavaScript	✓		✓		54*

* enfin pas vraiment



Pas d'entier natif en JavaScript

- Utilisation de nombres à virgule flottante
- Fonctionnement parfait pour $-2^{53} \leq x < 2^{53}$
- Rien ne va plus quand on sort de cet intervalle...
 - $2^{53} \leq x < 2^{54}$ *comptage de 2 en 2*
 - $2^{54} \leq x < 2^{55}$ *comptage de 4 en 4*
 - $2^{55} \leq x < 2^{56}$ *comptage de 8 en 8*
 - etc.



La division euclidienne

$$\begin{array}{r|l} \text{dividende } 195 & 8 \text{ diviseur} \\ \underline{16} & \hline 35 & 24 \text{ quotient} \\ \underline{32} & \\ \text{reste } 3 & \end{array}$$

- Comportements spécifiques au langage
 - C entier / entier → entier
 - PHP entier / entier → entier ou flottant
 - Python 2 entier / entier → entier
 - Python 3 entier / entier → flottant
entier // entier → entier
 - Haskell entier / entier → *erreur de compilation!*
div entier entier → entier



Division par zéro

- Comportements spécifiques au langage

- C entier / 0 → *Floating point exception*
- PHP < 8 entier / 0 → INF *PHP Warning: Division by zero*
- PHP ≥ 8 entier / 0 → *DivisionByZeroError*
- Python 2 entier / 0 → *ZeroDivisionError*
- Python 3 entier // 0 → *ZeroDivisionError*
- Haskell div entier 0 → **** Exception: divide by zero*
- JavaScript entier / 0 → Infinity



Pas de débordement

- Les entiers bornés fonctionnent comme des anneaux
 - entiers sur 8 bits \rightarrow anneau $\mathbb{Z}/256\mathbb{Z}$
- Exemples
 - $255 + 1 = 0$ *pour des entiers non signés sur 8 bits*
 - $127 + 1 = -128$ *pour des entiers signés sur 8 bits*
- Attention !
 - $a + b > a$ *pour $b > 0$ n'est pas toujours vrai!*



Attention aux boucles !

```
short i;  
for (i = 0; i < 32700; i += 256) {  
    printf("%d\n", i);  
}
```



Dans cette boucle, `i` sera toujours inférieur à 32700.



Positif ou négatif ?

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

void main() {
    if (abs(INT_MIN) < 0)
        printf("NEGATIVE\n");
    else
        printf("POSITIVE\n");
}
```



Quand ça dépasse



- **C**..... fonctionne comme un anneau
- **PHP**..... **integer** devient **double**
- **Python 2**..... **int** devient **long int**
- **Python 3**..... **int** est en réalité **long int**
- **Haskell**..... fonctionne comme un anneau
- **JavaScript**..... perte de précision, plafonne à **Infinity**

Un entier qui n'en est pas un

- **Des chiffres n'impliquent pas un nombre**

Numéro de sécurité sociale, numéro de carte bleu, code secret de carte bleu, numéro de compte bancaire, code postal, numéro de guichet, numéro de téléphone, référence d'article...

- **Les opérations mathématiques ne les concernent pas**

Addition, soustraction, multiplication, division...



Alors, que faire ?

- **Lire les spécifications**
Du langage utilisé, des formats d'échange, de leurs évolutions...
- **Tester les cas aux limites**
La plupart des langages disposent de constantes indiquant leurs limites
- **Utiliser les types adéquats**
JavaScript: *BigInt*, Haskell: *Integer*, Python 2: *long*, générique: *GNU GMP*...
- **Typer fortement, encapsuler**
Signature des fonctions, assertions, exceptions, chasse aux avertissements
- **Ne pas se reposer sur des hypothèses !**





**LES NOMBRES À VIRGULE FLOTTANTE
OU FLOTTANTS**



IEEE 754, pour les gouverner tous

- Norme omniprésente et qui évolue
 - IEEE 754-1985.....norme initiale, base 2
 - IEEE 854-1987.....base 10
 - IEEE 754-2008.....IEEE 754-1985 + IEEE 854-1987
 - IEEE 754-2019.....révision mineure
- IEEE 754 n'est pas l'ensemble \mathbb{R} des maths
- Chacun fait c'qui lui plaît...



Approximations

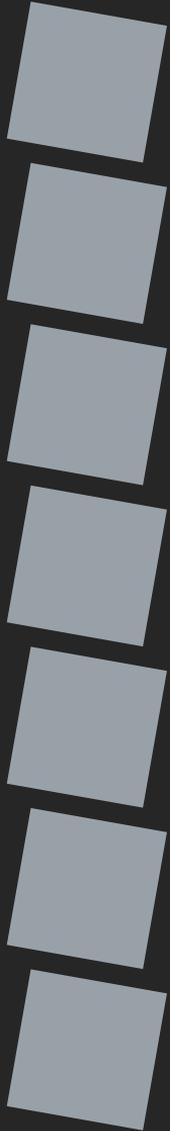
- $0.1 + 0.2 \neq 0.3$
- 0.1, 0.2 et 0.3 ne peuvent pas s'écrire en base 2
 - $0.1_{10} = 0.0\overline{0011}_2 = 0.0001100110011001100110011_2\dots$
 - $0.2_{10} = 0.\overline{0011}_2 = 0.0011001100110011001100110_2\dots$
 - $0.3_{10} = 0.01\overline{0011}_2 = 0.0100110011001100110011001_2\dots$
- 4 arrondis cumulés
 - 3 conversions (0.1, 0.2 et 0.3) et 1 addition



Les vraies valeurs de 0.1 à 0.9



0.10000000000000000000055511151231257827021181583404541015625
0.2000000000000000000011102230246251565404236316680908203125
0.299999999999999999988897769753748434595763683319091796875
0.400000000000000000002220446049250313080847263336181640625
0.5
0.59999999999999999997779553950749686919152736663818359375
0.6999999999999999999555910790149937383830547332763671875
0.80000000000000000000444089209850062616169452667236328125
0.900000000000000000002220446049250313080847263336181640625



Les erreurs s'accumulent

- `for(let i=0, x=0; i<N; i++) x = x + 0.3;`
 - N = 100 x = 30.000000000000005
 - N = 1000 x = 300.00000000000056
 - N = 10000 x = 3000.0000000003583
 - N = 100000 x = 29999.999999950614
 - N = 1000000 x = 299999.99999434233
 - N = 10000000 x = 2999999.9996692175
 - N = 100000000 x = 30000000.049996



L'addition n'est pas associative

$$(0.1 + 0.2) + (0.1 + 0.2) = 0.600000000000000001$$

Véritable valeur : 0.6000000000000000088817841970012523233890533447265625

$$0.1 + (0.2 + 0.1 + 0.2) = 0.6$$

Véritable valeur : 0.59999999999999997779553950749686919152736663818359375



Parfois $x \times 1 \div x \neq 1$



```
for (let x = 0; x < 1000; x++) {  
  if (x * (1 / x) !== 1) {  
    console.log(x);  
  }  
}
```

0	49	98	103	107	161	187	196	197	206	214	237	239	249
253	322	347	374	389	392	394	412	417	425	428	443	474	478
479	491	498	499	501	503	506	509	561	569	644	685	691	694
725	729	735	737	748	753	765	778	779	784	788	789	797	809
817	823	824	829	833	834	837	841	849	850	853	856	857	886
895	927	929	941	947	948	949	956	958	969	982	996	998	

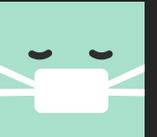
Division par zéro

- Comportements spécifiques au langage
 - C flottant / 0 → inf
 - PHP < 8 flottant / 0 → INF *PHP Warning: Division by zero*
 - PHP ≥ 8 flottant / 0 → *Fatal error*
 - Python flottant / 0 → *ZeroDivisionError*
 - Haskell flottant / 0 → Infinity
 - JavaScript flottant / 0 → Infinity



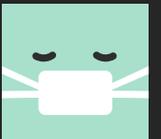
Mélange des genres : multiplication

- $1.0 * \text{"foo"}$
 - PHP < 8 0
 - PHP \geq 8 *Fatal error: Uncaught DivisionByZeroError*
 - Python *Can't multiply sequence by non-int of type 'float'*
 - Haskell *No instance for (Fractional [Char])*
 - C *error: invalid operands to binary **
 - JavaScript NaN
- $1.0 * \text{"1.0"}$ est valide en PHP!



Mélange des genres : division

- `1.0 / "foo"`
 - **PHP < 8** *INF PHP Warning: Division by zero*
 - **PHP ≥ 8** *Fatal error: Uncaught DivisionByZeroError*
 - **Python** *Unsupported operand type(s) for /: 'float' and 'str'*
 - **Haskell** *No instance for (Fractional [Char])*
 - **C** *error: invalid operands to binary /*
 - **JavaScript** NaN
- `1.0 / "1.0"` est valide en PHP!



Petite subtilité: -0 et $+0$

- Division

- $1 / -0 == -\text{Infinity}$

- $1 / 0 == \text{Infinity}$

- $\text{Infinity} / -0 == -\text{Infinity}$

- Addition

- $-0 + -0 == -0$

- $0 + -0 == 0$

- Égalité

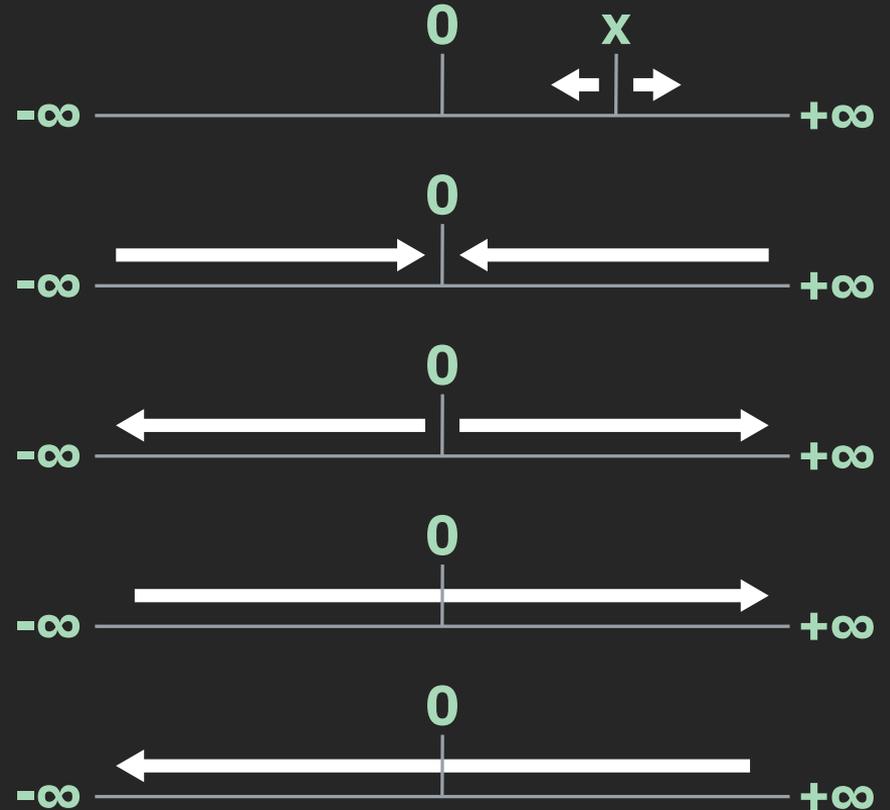
- $-0 == +0$



Arrondir les angles



- 10 types d'arrondis
 - inférieur / supérieur
 - vers ou à l'écart de zéro
 - à l'unité supérieure / inférieure
 - à la moitié vers ou à l'écart de zéro
 - à l'entier pair ou impair



Attention à la fonction round !

	#	-1.5	-0.5	0.5	1.5	2.5	3.5	4.5
C	<i>a</i>	-2	-1	1	2	3	4	5
PHP	<i>a</i>	-2	-1	1	2	3	4	5
Python	<i>b</i>	-2	0	0	2	2	4	4
Haskell	<i>b</i>	-2	0	0	2	2	4	4
JavaScript	<i>c</i>	-1	-0	1	2	3	4	5
Ruby	<i>a</i>	-2	-1	1	2	3	4	5

a: arrondi arithmétique

b: arrondi au nombre pair le plus proche

c: arrondi arithmétique au nombre le plus grand



Alors, que faire ?

- **Connaître les flottants**
Précision, comportements aux limites, arrondis, cas particuliers...
- **Tester les cas aux limites**
Division par 0, signe de zéro, opérations sur l'infini...
- **Utiliser les types adéquats**
Nécessité des nombres à virgule flottante, arrondis
- **Typer fortement, encapsuler**
Signature des fonctions, assertions, exceptions, chasse aux avertissements



Pour aller plus loin

- **Falsehoods about numbers**

<https://gist.github.com/joezeng/d5d562ff34b390f20c22405b6bc9e99e>

- **Deterministic cross-platform floating point arithmetics**

<http://christian-seiler.de/projekte/fpmath/>

- **Erreurs de calcul des ordinateurs**

<https://www.irisa.fr/sage/jocelyne/cours/precision/precision-2016.pdf>

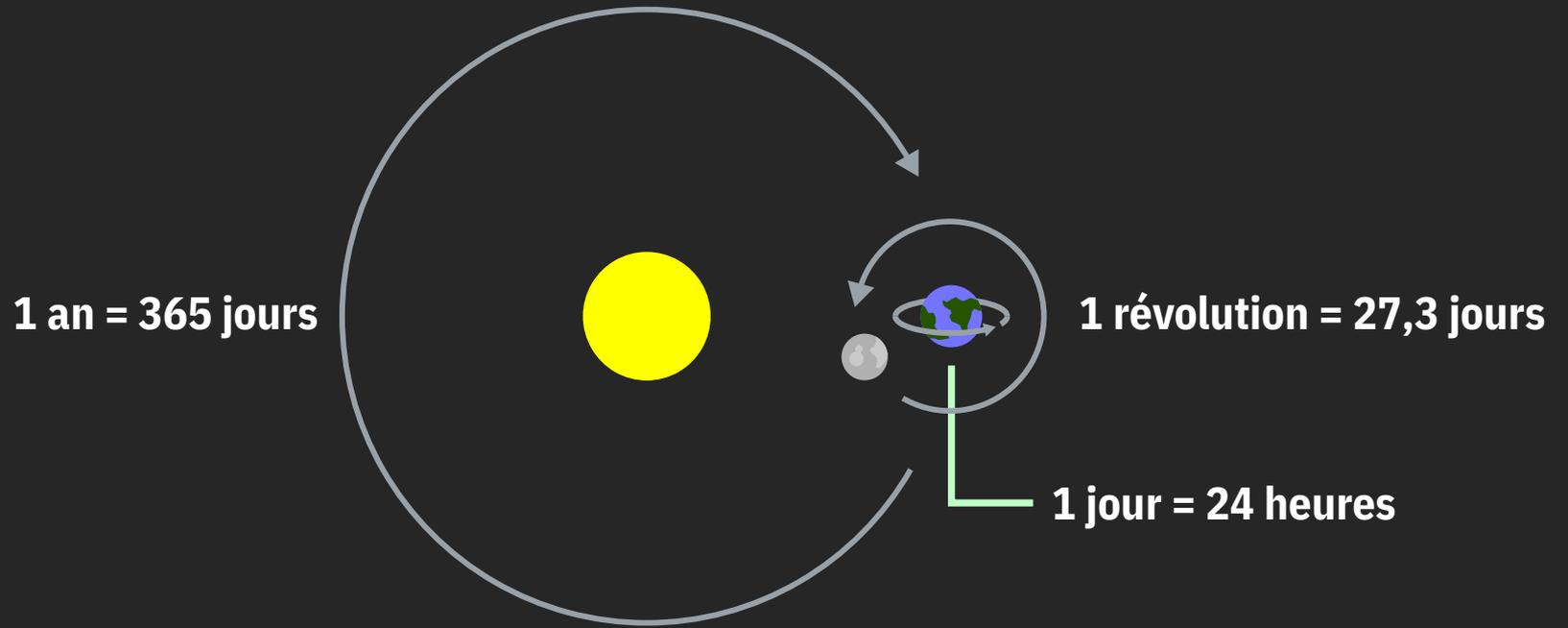
- **Rounding**

<https://en.wikipedia.org/wiki/Rounding>



Les dates



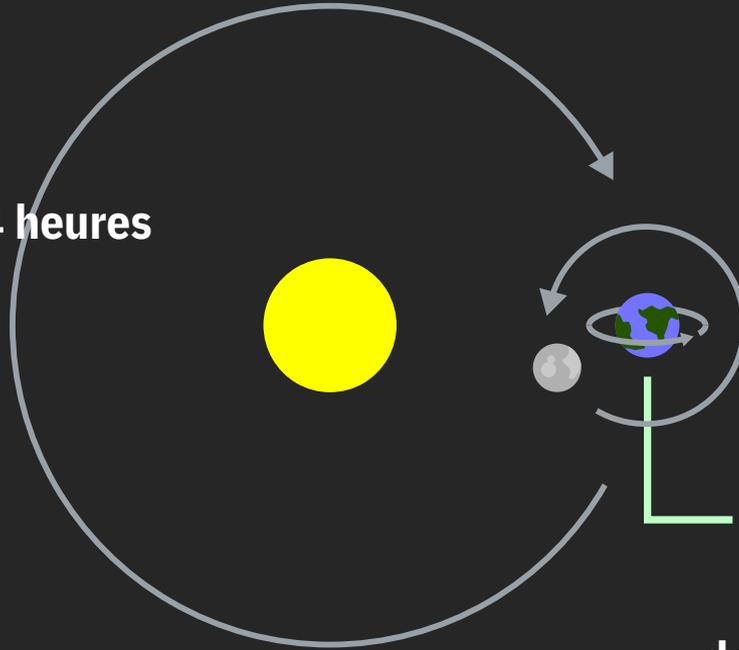


Le Soleil, la Terre, la Lune



1 année sidérale
~365,256363 jours de 24 heures

1 an = ~~365~~ jours



1 révolution
27,321582 jours

1 révolution = ~~27,3~~ jours

1 jour = ~~24~~ heures

1 jour solaire
de 23 h 59 min. 39 sec.
à 24 h 0 min. 30 sec.

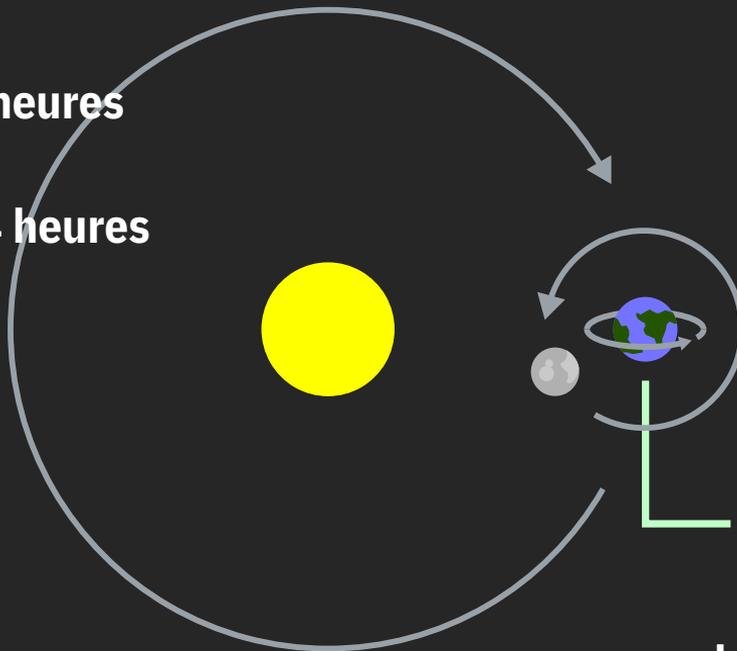
Question de révolution



1 année tropique
~365,2422 jours de 24 heures

1 année sidérale
~365,256363 jours de 24 heures

1 an = 365 jours



1 période synodique
29,530589 jours

1 révolution
27,321661 jours

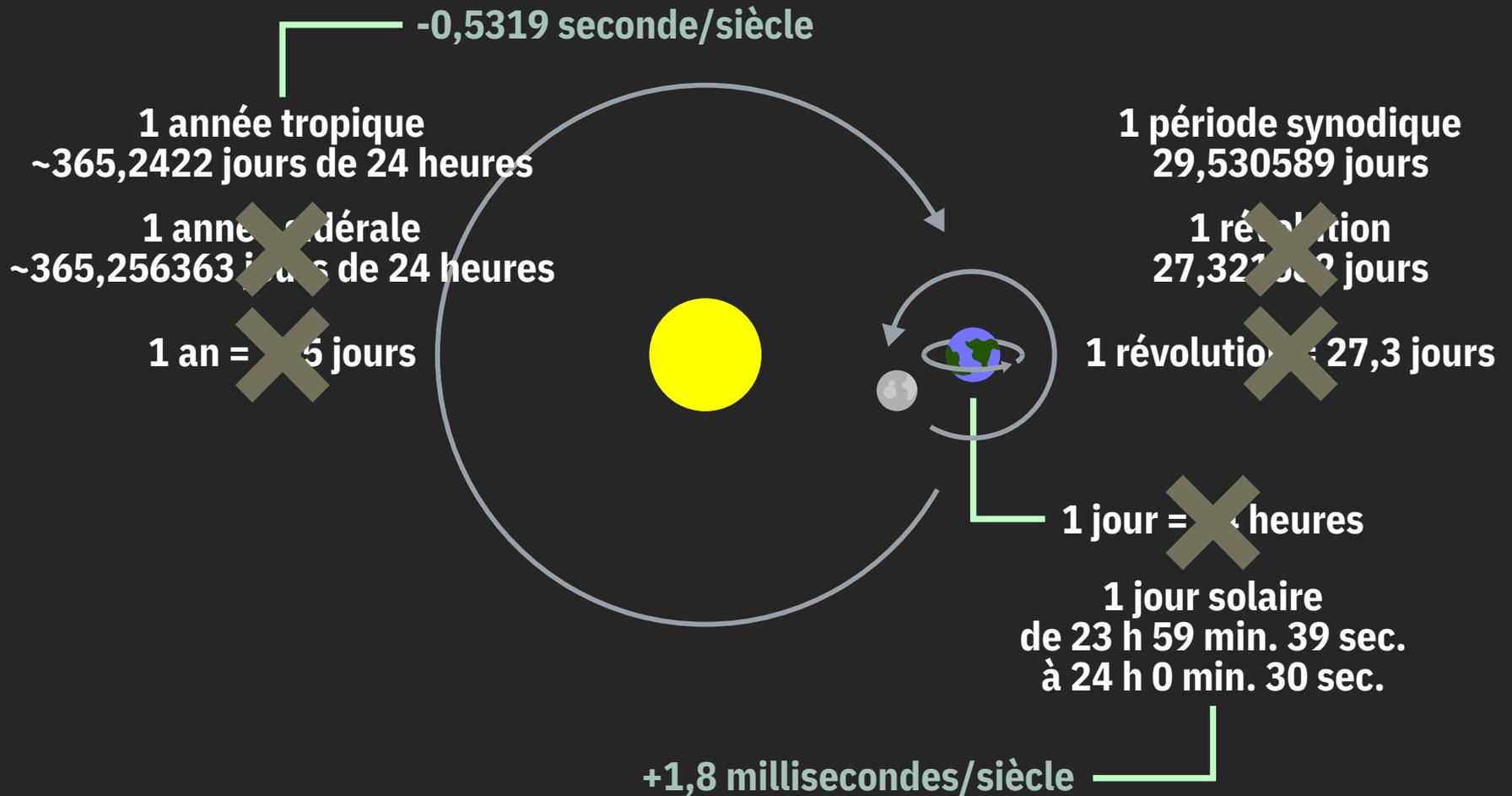
1 révolution = 27,3 jours

1 jour = 24 heures

1 jour solaire
de 23 h 59 min. 39 sec.
à 24 h 0 min. 30 sec.

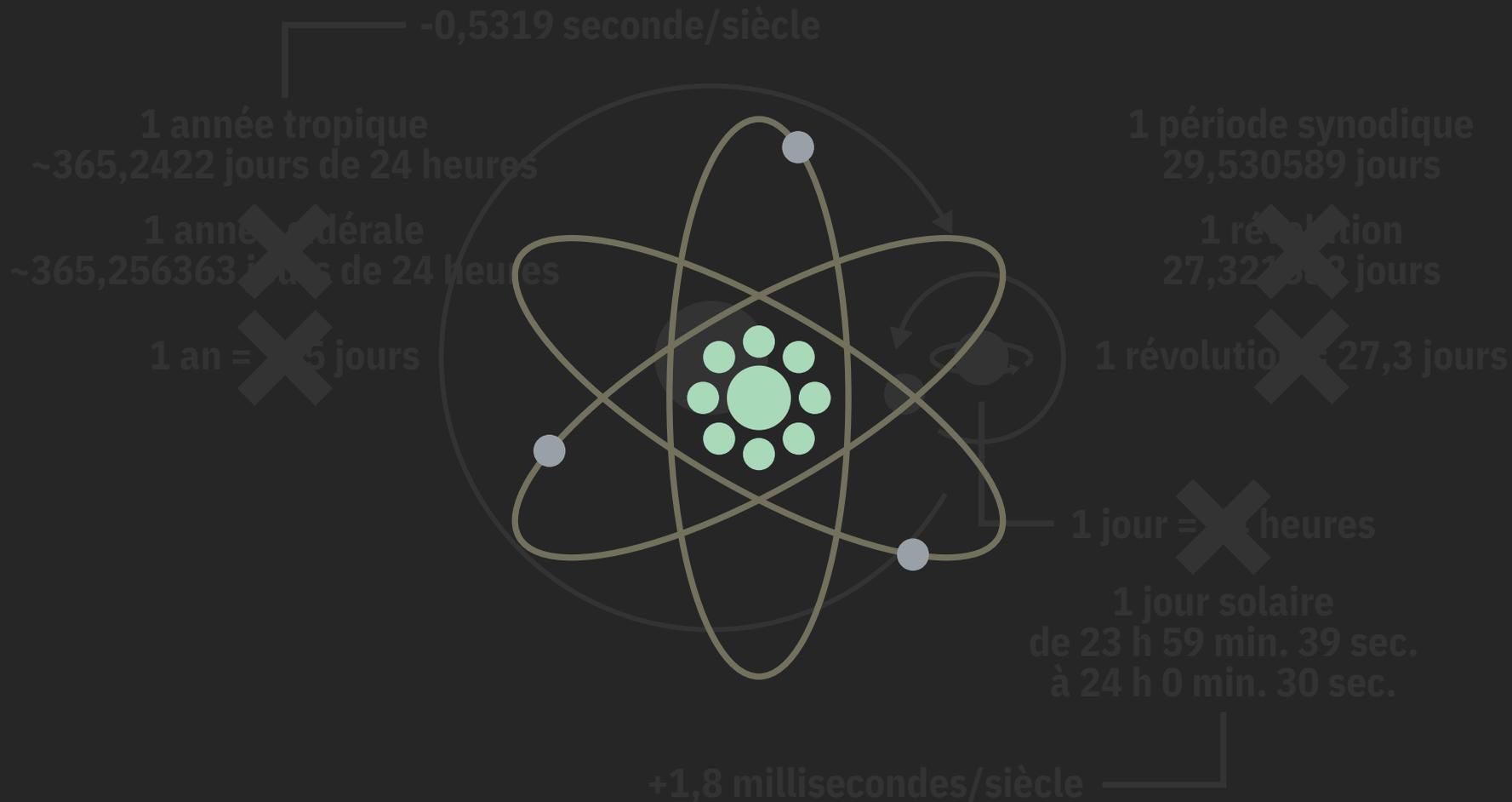
Prendre la bonne révolution



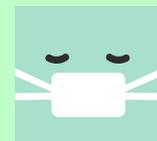


Rien n'est figé dans le marbre





Et l'horloge atomique dans tout ça ?



Temps universel coordonné (UTC)

- Temps

- Temps universel solaire..... UT1
- Temps atomique international..... TAI
- Temps universel coordonné..... UTC
Et pas CUT pour Coordinated Universal Time afin de ne pas froisser les Français...

- Synchronisation et rattrapage

- **1958**: synchronisation UT1 et TAI
- **1963**: Recommandation n°374, création d'UTC
- **1972**: 10 secondes de décalage, mise à jour d'UTC



Seconde intercalaire (leap second)

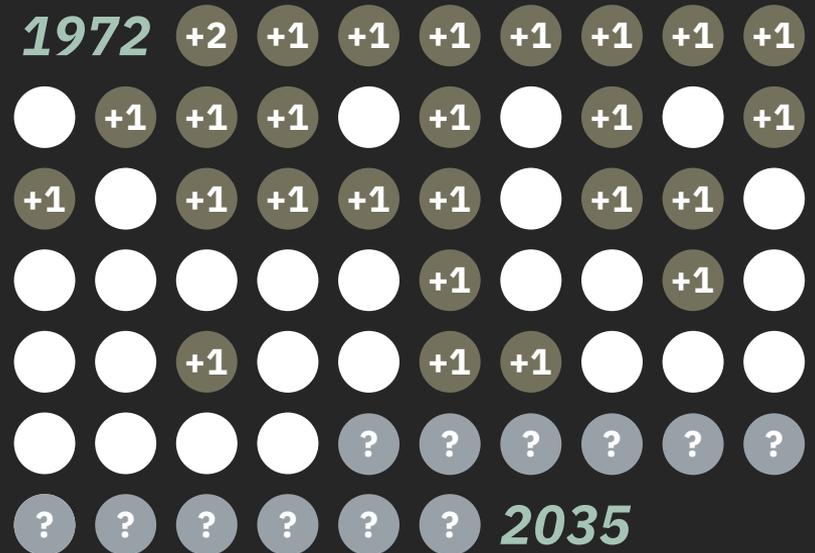
- Synchronisation de UTC et TAI

- 30 juin, 31 décembre

- on ajoute 1 seconde
- on retranche 1 seconde
- on ne fait rien

- 23:59:60 ≠ 24:00:00

- Une seconde appelée à disparaître en 2035?



Calendrier grégorien

- **Calendrier romain**..... -450
Calendrier approximativement lunaire, débute en mars à partir de -153.
- **Calendrier julien**..... -46
Calendrier solaire, années bissextiles approximatives, débute au 1^{er} janvier.
- **Calendrier grégorien**.....1582
Calendrier solaire, années bissextiles plus justes, supprime 11 jours.
Adoption progressive: catholiques → 16^e, protestants → 18^e, orthodoxes → 20^e.



Erreurs historiques

- **Pas d'année zéro**
21^e siècle débutant en 2001
- **Annus domini de Denys le Petit au 6^e siècle**
Naissance aujourd'hui estimée entre -7 et -5, mais beaucoup se sont déjà trompés...
- **Bug de l'an 2000 et années codées sur 2 chiffres**
Problème soulevé dès 1958 par Bob Berner, co-inventeur du code ASCII
- **Bug de l'an 2038 et systèmes 32 bits**
Fichiers ZIP, système de fichiers FAT, systèmes d'exploitation, horloges temps réel...
- **Et tant d'autres...**



Fuseaux horaires, heure d'été...

- Le fuseau horaire et l'heure d'été varient
En fonction du pays, de la date de mise en place, des évolutions historiques
- Le décalage horaire peut être de 30 ou 45 minutes!
Iran +3h30, Inde +5h30, Îles Chatham + 12h45...



Précision du type de données

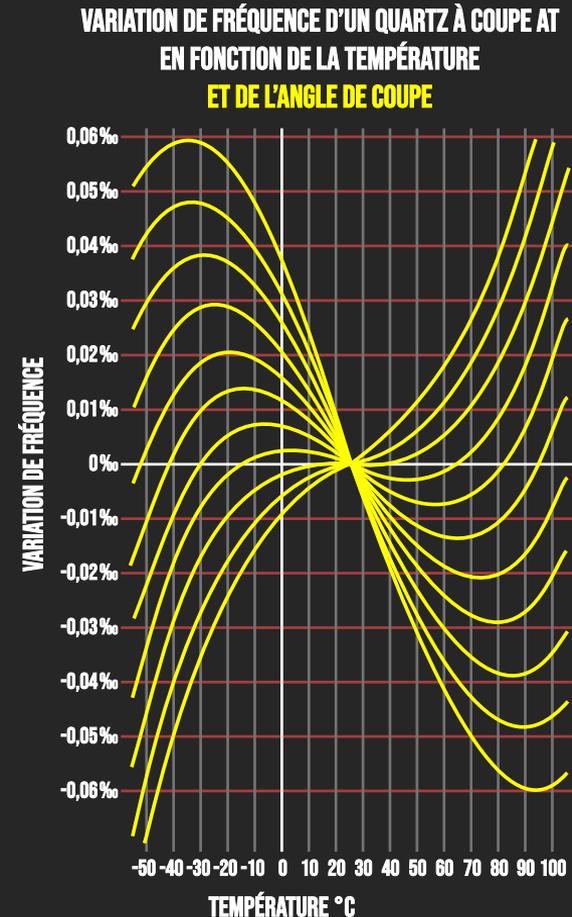
- Java 8.....1 ns
- Python.....1 μ s
- JS Date.....1 ms
- JS Performance... 5 μ s
- ISO 8601.....1 μ s ?
- FAT..... 2 s
- EXT3.....1 s
- EXT4.....1 ms
- NTFS..... 0,1 μ s



Précision réelle

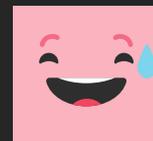


- Atténuation
 - d'attaques Meltdown, Spectre...
 - de prise d'empreinte
- Instabilité de l'horloge système
 - précision et âge du quartz
 - variations avec la température
 - coordination NTP
- Et dans une VM suspendue ?



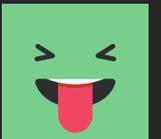
Calcul de dates

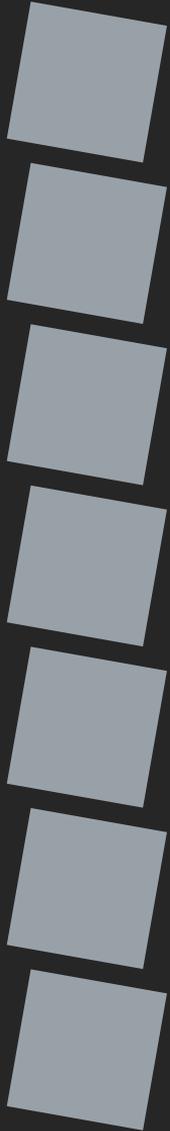
- Des calculs « simples »
 - mois/semaine suivante
 - jour de la semaine
 - durée entre deux dates
- Des calculs pas simples
 - dates religieuses lunaires
- Des notions locales
 - numéro de la semaine
 - premier jour de la semaine
 - jours ouvrés



C'est quoi un mois ?

- Ça dépend à **qui** on pose la question
 - PHP..... 28, 29, 30 ou 31 jours
 - Google.....30,4167 jours
 - Convertlive.com.....30,4375 jours
- Ça dépend **comment** on pose la question
 - calcul naïf
 - fonctions dédiées





PHP et mois

```
> $oneMonth = new DateInterval("P1M");
```

```
> $date = new DateTime("2022-03-30");
```

```
> print($date->sub($oneMonth)->format("Y-m-d"));
```

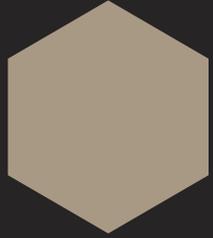
```
2022-03-02
```

```
> $date = new DateTime("2024-03-30");
```

```
> print($date->sub($oneMonth)->format("Y-m-d"));
```

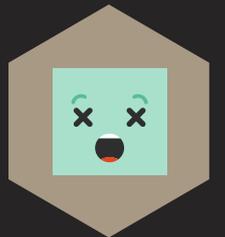
```
2024-03-01
```

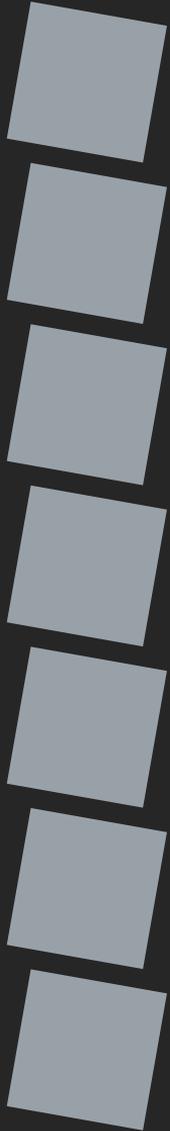




*Pâques est
le dimanche qui suit
le 14^e jour de la Lune
qui atteint cet âge le 21 mars
ou immédiatement après*

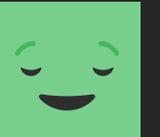
CONCILE DE NICÉE, 325





ISO 8601:2004

- Chaîne de caractères
 - difficulté à parser
- Calendrier grégorien **proleptique**
 - année zéro
 - flou avant 1582
 - seconde intercalaire
 - date ou intervalle de dates



Alors, que faire ?

- **Considérer les problématiques**
Années bissextiles, secondes intercalaires, référentiel, précision...
- **Utiliser normes, standards et bibliothèques**
ISO 8601, UTC..., JavaScript: Luxon...
- **Mettre à jour régulièrement**
Les secondes intercalaires ne peuvent pas être prévues à l'avance
- **Multiplier les tests**
Les cas aux limites sont très nombreux : décalage horaire, heure d'été...



Pour aller plus loin

- **Précision de la synchronisation de NTP**

https://kb.meinbergglobal.com/kb/time_sync/time_synchronization_accuracy_wit

- **Falsehoods programmers believe about time**

<https://www.wired.com/2012/06/falsehoods-programmers-believe-about-time/>

- **Calcul de la date de Pâques**

https://fr.wikipedia.org/wiki/Calcul_de_la_date_de_P%C3%A2ques

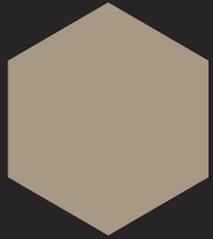
- **Résolution 4 de la 27^e CGPM (2022)**

<https://www.bipm.org/fr/cgpm-2022/resolution-4>



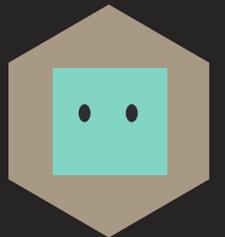
Les chaînes de caractères





*Une chaîne de caractères est à la fois
conceptuellement
une suite ordonnée de caractères
et physiquement
une suite ordonnée d'unités de code*

CHAÎNE DE CARACTÈRES, WIKIPÉDIA



Concepts nés avant l'ordinateur



- **15^e siècle**..... typographie
Caractères, signes, ligatures, espaces
- **16^e siècle**..... apparition des accents en français
- **19^e siècle**..... télégraphe
Premières codifications et automatisations, codes de contrôle
- **1930**..... téléscripneur
Codes de contrôle, séparateurs, augmentation du nombre de caractères
- **Depuis le 19^e siècle**..... standardisation
Code Baudot, Western Union, EBCDIC, ASCII, ISO-8859-*...

Caractère est un concept

- Représentation visuelle
- Un caractère peut représenter
 - lettre, ligature, sinogramme, emoji, symbole...
 - diacritique (*suscrit, souscrit, prescrit, adscrit, inscrit, circonscrit*)
 - signe de ponctuation
 - séparateur (*espace, tabulation, retour à la ligne...*)
 - opération spéciale (*sonnerie, effacement, déplacement...*)



Glyphe, point de code et encodage

- **Glyphe**..... représentation visuelle du caractère
Défini par la police de caractères
- **Point de code**..... identifiant numérique
- **Encodage**..... représentation physique du point de code
ASCII, ISO-8859-*, UCS2, UTF-8, UTF-16, UTF-32...

U+0041

Point de code

LATIN CAPITAL LETTER A

Désignation du caractère

UNICODE



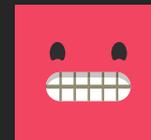
Encodages reconnus par l'IANA



US-ASCII ISO_8859-1:1987 ISO_8859-2:1987 ISO_8859-3:1988 ISO_8859-4:1988 ISO_8859-5:1988 ISO_8859-6:1987 ISO_8859-7:1987
ISO_8859-8:1988 ISO_8859-9:1989 ISO-8859-10 ISO_6937-2-add JIS_X0201 JIS_Encoding Shift_JIS
Extended_UNIX_Code_Packed_Format_for_Japanese Extended_UNIX_Code_Fixed_Width_for_Japanese BS_4730 SEN_850200 C IT ES
DIN_66003 NS_4551-1 NF_Z_62-010 ISO-10646-UTF-1 ISO_646.basic:1983 INVARIANT ISO_646.irv:1983 NATS-SEFI NATS-SEFI-ADD
NATS-DANO NATS-DANO-ADD SEN_850200 KS_C_560 2-KR EUC_2022-JP ISO-10646-2 JIS_C6220-1969-jp
JIS_C6220-1969-ro PT greek7-old ISO_639-2-62-010 greek7 ISO_639-2-62-010 S_viewdata INIS INIS-8
INIS-cyrillic ISO_5427:1981 ISO_5427:1981 ISO_5427:1981 ISO_5427:1981 ISO_5427:1981 ISO_5427:1981 ISO_5427:1981 ISO_5427:1981
ASMO_449 iso-ir-90 JIS_C6229-1 JIS_C6229-1984 JIS_C6229-1984 JIS_C6229-1984 JIS_C6229-1984-hand JIS_C6229-1984-hand-add
JIS_C6229-1984-kana ISO_2033-1 ANSI_X3.4-1964 51-7bit T.61-8bit A-cyrillic ISO_10646-1 CSA_Z243.4-1985-2
CSA_Z243.4-1985-gr ISO_8859-6 ISO_8859-8 ISO_8859-8-E ISO_8859-8-I ISO_69103 JUS_I.B1.003 IEC_P27-1
JUS_I.B1.003-serb JUS_I.B1.003-mac greek7 ISO_6937-2 ISO_6937-2-74 ISO_8859-13 ISO_10367-box latin-lap
JIS_X0212-1990 DS_2089 us-dk dk-us KSC_5601 UNICOD_10-2022 CN-EXT UTF-8 ISO-8859-13 ISO-8859-14
ISO-8859-15 ISO-8859-16 GBK GB18030 EBCDIC_15 U_15 EBCDIC_15 SD EBCDIC_15 ISO-11548-1 KZ-1048
ISO-10646-UCS-2 ISO-10646-UCS-4 ISO-10646-UCS-B ISO-10646-UCS-E ISO-10646-J-1 ISO-10646-J-1 ISO-10646-J-1
ISO-Unicode-IBM-1268 ISO-Unicode-IBM-1268 ISO-Unicode-IBM-1268 ISO-Unicode-IBM-1268 UNICOD_10-2022 SCSU UTF-7 UTF-16BE UTF-16LE
UTF-16 CESU-8 UTF-32 UTF-32BE UTF-32LE BOCU-1 UTF-7-IMAP ISO-8859-1 Windows-3.1 Windows-3.1-Latin-1
ISO-8859-2-Windows-Latin-2 ISO-8859-9-Windows-Latin-1 ISO-8859-1-Windows-3.1-Latin-1
IBM850 PC8-Danish-Norwegian IBM850 PC8-Turkish IBM850 IBM850 IBM850 IBM850 IBM850 IBM850 IBM850 IBM850
Ventura-Math Microsoft-Publishing IBM850
IBM281 IBM284 IBM285 IBM290 IBM437 IBM437 IBM852 IBM852 IBM852 IBM852 IBM852 IBM852 IBM852 IBM852
IBM864 IBM865 IBM868 IBM869
EBCDIC-CA-FR EBCDIC-DK-NO EBCDIC-DK-NO-A EBCDIC-FI-SE EBCDIC-FI-SE-A EBCDIC-FR EBCDIC-IT EBCDIC-PT EBCDIC-ES EBCDIC-ES-A
EBCDIC-ES-S EBCDIC-UK EBCDIC-US UNKNOWN-8BIT MNEMONIC MNEM VISCII VIQR KOI8-R HZ-GB-2312 IBM866 IBM775 KOI8-U IBM00858
IBM00924 IBM01140 IBM01141 IBM01142 IBM01143 IBM01144 IBM01145 IBM01146 IBM01147 IBM01148 IBM01149 Big5-HKSCS IBM1047
PTCP154 Amiga-1251 KOI7-switched BRF TSCII CP51932 windows-874 windows-1250 windows-1251 windows-1252 windows-1253
windows-1254 windows-1255 windows-1256 windows-1257 windows-1258 TIS-620 CP50220

258!

Normes en constante évolution



- **Caractères + encodage**

- **ASCII**.....1963-1986

- Encodage 7 bits utilisé par de très nombreuses variantes

- **ISO/CEI 8859**.....1986-2001

- Extension de l'ASCII à 8 bits incluant le support international

- **ISO/CEI 10646**.....1993-2012

- Encodage d'Unicode : UCS-2, UCS-4, UTF-1, UTF-8, UTF-16, UTF-32

- **Caractères uniquement**

- **Unicode**.....1991-2024

- Liste de caractères et d'algorithmes, prise en compte de la polysémie



UNICODE



Unicode à la rescousse



- Liste de nombreux caractères
 - 154 998 caractères à la version 16.0, emojis inclus
 - 1 114 111 points de code maximum
 - Unicode ne contient pas tous les caractères possibles !
- Règles d'utilisation
 - algorithmes de manipulation de chaînes Unicode
 - bibliothèques ICU4C (C, C++), ICU4J (Java) et ICU4X



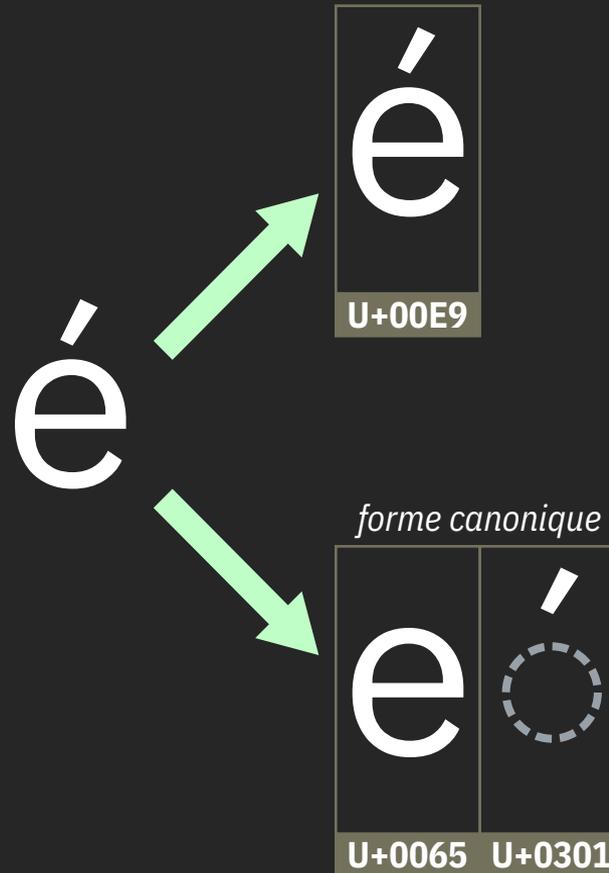
Quelques propriétés par caractère

Age, alnum, Alphabetic, Block, Case_Sensitive, Cased, Changes_When_Casemapped, Changes_When_NFKC_Casefolded, Changes_When_Titlecased, Changes_When_Uppercased, Confusable_MA, Decomposition_Type, General_Category, graph, Grapheme_Base, ID_Continue, ID_Start, Identifier_Type, Idn_Mapping, Idn_Status, idna2003, idna2008, isCased, isCasefolded, isLowercase, isNFKC, isNFKD, isNFM, ISO_Comment, isTitlecase, isUppercase, Line_Break, Lowercase, NFKC_Casefold, NFKC_Inert, NFKC_Quick_Check, NFKD_Inert, NFKD_Quick_Check, print, Script, Script_Extensions, Sentence_Break, Simple_Titlecase_Mapping, Simple_Uppercase_Mapping, subhead, Titlecase_Mapping, toIdna2003, toNFKC, toNFKD, toNFM, toTitlecase, toUppercase, toUts46n, toUts46t, uca, uca2, uca2.5, uca3, Unicode_1_Name, Uppercase_Mapping, uts46, Word_Break, XID_Continue, XID_Start, ANY, ASCII, ASCII_Hex_Digit, Basic_Emoji, Bidi_Class, Bidi_Control, Bidi_Mirrored, Bidi_Mirroring_Glyph, Bidi_Paired_Bracket, Bidi_Paired_Bracket_Type, blank, bmp, Canonical_Combining_Class, Case_Folding, Case_Ignorable, Changes_When_Casefolded, Changes_When_Lowercased, CJK_Radical, Dash, Default_Ignorable_Code_Point, Deprecated, Diacritic, East_Asian_Width, Emoji, Emoji_Component, Emoji_Flag_Sequence, Emoji_Keycap_Sequence, Emoji_Modifier, Emoji_Modifier_Base, Emoji_Modifier_Sequence, Emoji_Presentation, Emoji_Tag_Sequence, Emoji_Zwj_Sequence, Equivalent_Unified_Ideograph, Extended_Pictographic, Extender, Full_Composition_Exclusion, Grapheme_Cluster_Break, Grapheme_Extend, Grapheme_Link, Hangul_Syllable_Type, HanType, Hex_Digit, Hyphen, Identifier_Status, Ideographic, Idn_2008, idna2008c, IDS_Binary_Operator, IDS_Tertiary_Operator, Indic_Positional_Category, Indic_Syllabic_Category, isNFC, isNFD, Join_Control, Joining_Group, Joining_Type, kAccountingNumeric, kOtherNumeric, kPrimaryNumeric, kSimplifiedVariant, kTraditionalVariant, Lead_Canonical_Combining_Class, Logical_Order_Exception, Lowercase_Mapping, Math, Name_Alias, Named_Sequences, Named_Sequences_Prov, NFC_Inert, NFC_Quick_Check, NFD_Inert, NFD_Quick_Check, Noncharacter_Code_Point, Numeric_Type, Numeric_Value, Pattern_Syntax, Pattern_White_Space, Prepended_Concatenation_Mark, Quotation_Mark, Radical, Regional_Indicator, Segment_Starter, Sentence_Terminal, Simple_Case_Folding, Simple_Lowercase_Mapping, Soft_Dotted, Standardized_Variant, Terminal_Punctuation, toCasefold, toLowercase, toNFC, toNFD, Trail_Canonical_Combining_Class, Unified_Ideograph, Uppercase, Variation_Selector, Vertical_Orientation, White_Space, xdigit



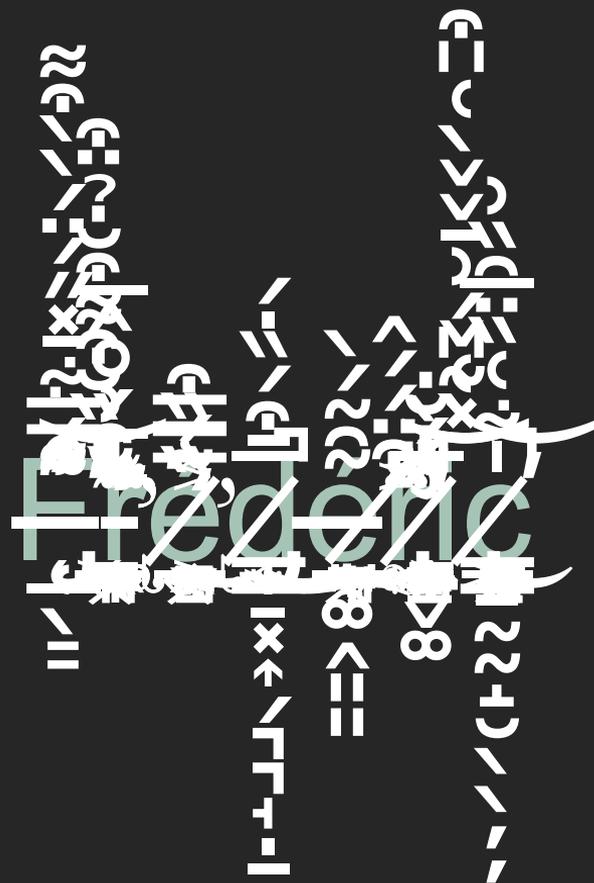
Plusieurs façons d'écrire un caractère

- Des marques
 - 318 combineurs
 - 211 modificateurs
- Un seul accent aigu?
 - le caractère U+00B4
 - le combineur U+0301
 - le modificateur U+02CA



Abus de marques: Zalgo text

- Exemple sur « Frédéric »
 - 8 lettres
 - 609 octets en UTF-8
- Espace réservé?
 - consommation mémoire
 - en base de données
- Générateur de Zalgo text
<https://lingojam.com/ZalgoText>

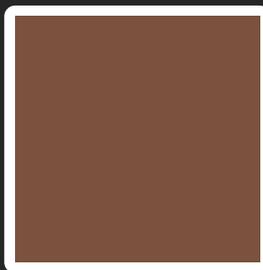




=



=



F0
9F
A7
91
F0
9F
8F
BF

1 caractère

2 points de code

8 octets (UTF8)

Émoticônes et variantes



À l'assaut de la polysémie



- **Unicode distingue les caractères en fonction de leur sens** quitte à empiéter sur le domaine des polices de caractères (graisse, empattement, chasse...)
- **Certains caractères sont alors confusants (*et dangereux*)**
<https://util.unicode.org/UnicodeJsps/confusables.jsp>

Lettre minuscule latine E	e	e	Minuscule mathématique grasse E de ronde
Lettre minuscule cyrillique ié	е	е	Minuscule mathématique gothique E
Lettre minuscule cyrillique tché abkhaze	ѣ	ѣ	Minuscule mathématique ajourée E
Symbole estimé	Ǝ	Ǝ	Minuscule mathématique gothique grasse E
Minuscule E de ronde	ⓔ	ⓔ	Minuscule mathématique sans empattement E
Minuscule E italique ajouré	ⓔ	ⓔ	Minuscule mathématique grasse sans empattement E
Lettre minuscule latine E gothique	ⓔ	ⓔ	Minuscule mathématique italique sans empattement E
Minuscule mathématique grasse E	ⓔ	ⓔ	Minuscule mathématique italique grasse sans empattement E
Minuscule mathématique italique E	ⓔ	ⓔ	Minuscule mathématique à chasse fixe E
Minuscule mathématique italique grasse E	ⓔ	ⓔ	Lettre minuscule latine E pleine chasse

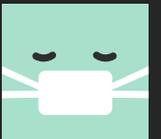
Un peu d'espace



- **ASCII**..... 1 espace
SPACE
- **ISO/CEI 8859**..... 2 espaces
SPACE / NO-BREAK-SPACE
- **Unicode**.....18 espaces
SPACE / NO-BREAK SPACE / OGHAM SPACE MARK / MONGOLIAN VOWEL SEPARATOR /
EN QUAD / EM QUAD / EN SPACE / EM SPACE / THREE-PER-EM SPACE / FOUR-PER-EM SPACE /
SIX-PER-EM SPACE / FIGURE SPACE / PUNCTUATION SPACE / THIN SPACE / HAIR SPACE /
NARROW NO-BREAK SPACE / MEDIUM MATHEMATICAL SPACE / IDEOGRAPHIC SPACE

Un point de code pour 2 (ou 3, 4...)

- «Vraie» ligature
 - égalité si translittération
 - œuvre ≠ oeuvre
- «Fausse» ligature
 - égalité en forme
NFKD/NFKC
 - effleurer ≈ effleurer
 - ex. : ff, fi, fl, ffi, ffl, ft, st
- Ligature visuelle
 - générée par la police
 - figure = figure



International Components for Unicode

- Parcours de chaînes
- Encodage / détection
- Conversion
- Compression
- Locales
- Normalisation
- Formatage
- Découpage
- Translittération
- Collation / tri
- Recherche
- Mise en page



Une petite collation ?



- Trier des chaînes de caractères est complexe!
 - tous les caractères ne sont pas triables : Ж, ѱ, ♪, ∞, ◇, △...
 - le point de code n'est pas un indicateur d'ordre !
 - plusieurs niveaux/ordres de comparaison (caractères, accents...)
- Dépend de : destinataire, langue, culture, personnalisation
- Points de code contrôlant la collation
 - Combining Grapheme Joiner **U+034F**
 - Bidirectional Ordering Controls **U+206[6-9] / U+202[A-E]**

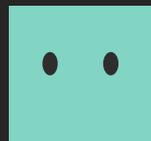
Conventions linguistiques complexes

- De langue
 - suédois z < ö
 - allemand ö < z
- D'usage
 - dico allemand of < öf
 - annuaire allemand öf < of
- De personnalisation
 - minuscules vs majuscules
 - ordre des accents
 - ex. cote, côte, coté, côté
- Des ligatures



Découper un texte

- " どこで生れたかとんと見当がつかぬ。 ".split(" ")
 - どこ / で / 生れた / か / とんと / 見当 / が / つかぬ
 - les espaces ne sont pas toujours utilisés pour séparer les mots
 - traduction : je n'ai aucune idée de l'endroit où il/elle est né/e
- La ponctuation est différente en fonction de la langue



Fonctions de formatage basiques

- Gestion des césures

- espace insécable.. **U+00A0**
- jointure de mots... **U+2060**
- espace largeur 0... **U+200B**

- Séparateurs

- de lignes.....**U+2028**
- de paragraphes.....**U+2029**

- Codes de contrôle

- retour à la ligne.....**U+000A**
- retour chariot.....**U+000D**
- tabulation..... **U+0009**
- page suivante.....**U+000C**
- ligne suivante.....**U+0085**
EBCDIC !



Alors, que faire ?

- **Considérer la complexité d'un caractère**
Polysémie, points de code, glyphe...
- **Suivre les évolutions d'Unicode**
Unicode v16.0, CLDR, nouveaux caractères, nouvelles règles...
- **Nettoyer les chaînes**
Translittération, forme NFC, remplacement/suppression des caractères indésirable
- **Utiliser des bibliothèques adaptées**
Projets ICU, ne pas se reposer sur les types et fonctions de base du langage



Pour aller plus loin

- **Unicode Security Considerations**
<https://unicode.org/reports/tr36/>
- **Unicode Common Locale Data Repository**
<http://cldr.unicode.org/>
- **International Components for Unicode (ICU)**
site project: <http://site.icu-project.org/>
ICU demonstrations: <https://icu4c-demos.unicode.org/icu-bin/icudemos>
- **Unicode Utilities**
confusables: <https://util.unicode.org/UnicodeJsps/confusables.jsp>
character properties: <https://util.unicode.org/UnicodeJsps/character.jsp>





**CHAÎNES DE CARACTÈRES
ET LANGAGES DE PROGRAMMATION**



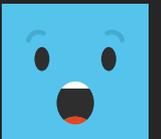
Chaînes et caractères



- **C**..... suite d'octets à zéro terminal
sans limite de taille, sans contrainte sauf l'impossibilité d'utiliser le caractère \000
- **Python 3**..... tableau de points de code Unicode
valeurs limitées à sys.maxunicode
- **Haskell**..... liste chaînée de points de code Unicode
valeurs limitées à maxBound::Char, autres types de chaînes disponibles
- **JavaScript**..... tableau de mots de 16 bits
sans contrainte
- **PHP**..... tableau d'octets
sans contrainte

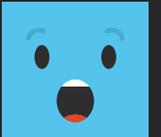
JavaScript et ses chaînes

- 3 opérations
 - stricte..... `a === b`
 - faible.....`a == b`
 - très stricte.. `Object.is(a, b)`
- Comparaison abstraite
 - `[1, 2] == "1,2"`
 - `10 == "1e1"`
mais `"1e1" != 10`
- 4 algorithmes
 - stricte (`===`)
 - abstraite (`==`)
 - SameValue
 - SameValueZero (`String.includes`)



PHP et ses chaînes

- À la recherche du nombre perdu!
 - "00000000042" == "42" / "1e1" == "10"
 - 10 == "1e1", "1e1" == 10 et 0 == "a" (pour PHP < 8)
 - md5('240610708') == md5('QNKCDZO')
"0e462097431906509019562988736854" == "0e830400451993494058024219903391"
- Attention aux clés!
 - array(42=>24)[**42**] === array(42=>24)["**42**"]
 - array(42=>24)[**42**] !== array(42=>24)["**042**"]

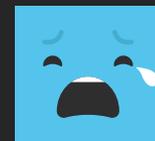


MySQL et ses chaînes

- À la recherche du nombre perdu
 - SELECT 10 = "1e1"; vrai
 - SELECT "1e1" = 10; vrai
 - SELECT 0 = "a";vrai
 - SELECT "1e1" = "10"; faux
 - SELECT md5('240610708') = md5('QNKCDZO');faux

- Pas de comparaison sur la forme canonique

Nécessité de normaliser les chaînes Unicode avant leur insertion en base de données

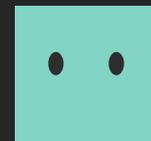


Littéralement

- Chaque langage a sa propre gestion des chaînes littérales
 - plusieurs types de chaînes littérales
 - usage généralisé trompeur de ' et "
- Une grande variété de caractères spéciaux

```
\0, \a, \b, \f, \n, \r, \t, \v, \", \&, \', \\, \NUL, \SOH, \STX, \ETX,  
\EOT, \ENQ, \ACK, \BEL, \BS, \HT, \LF, \VT, \FF, \CR, \SO, \SI, \DLE,  
\DC1, \DC2, \DC3, \DC4, \NAK, \SYN, \ETB, \CAN, \EM, \SUB, \ESC, \FS,  
\GS, \RS, \US, \SP, \DEL, \^@, \^A, \^B, \^C, \^D, \^E, \^F, \^G, \^H,  
\^I, \^J, \^K, \^L, \^M, \^N, \^O, \^P, \^Q, \^R, \^S, \^T, \^U, \^V,  
\^W, \^X, \^Y, \^Z, \^[, \^\\, \^], \^^, \^_, \99999, \o7777777, \xFFFFFF
```

Haskell

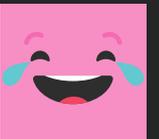


матрёшки

- Une chaîne peut contenir
 - une chaîne qui contient
 - une chaîne qui contient
 - une chaîne etc.



- Caractères spéciaux et séquence d'échappement
 - différences entre langages
 - plusieurs types de chaînes par langage
 - plusieurs types de valeurs littérales par langage



The great escape

```
php -r 'system("echo \"\\\\\\\\\\\\\\\\Hello, World\\\\\\\\\\\\\\\\!\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\"");'
```

- A) Hello, World!
- B) \Hello, World\!\
- C) \\Hello, World\!\
- D) \\\Hello, World\\!\



Alors, que faire ?

- **Considérer les problématiques**

Encodage, spécificité du langage, caractères confusants, évolution d'Unicode.

- **Ne pas utiliser le Shell**

Trop de variantes et de subtilités existent pour que cela soit fiable

- **Contrôler la génération de chaînes**

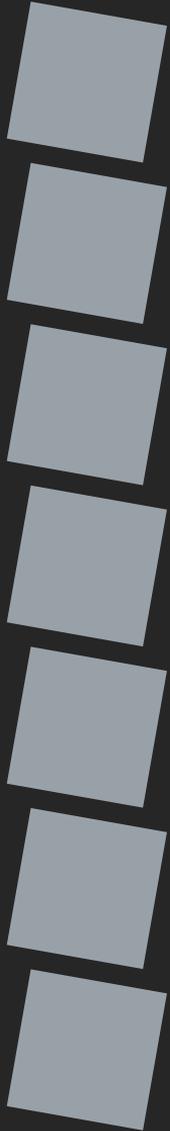
Injection SQL, charge utile XSS...

- **Normaliser les chaînes de caractères**

Les espaces en début ou en fin d'une chaîne sont-ils autorisés ?

- **Typer fortement, encapsuler**

PHP ou JavaScript ont exacerbé la versatilité de la chaîne de caractères !



Pour aller plus loin

- **Falsehoods about text**

<https://wiesmann.codiferes.net/wordpress/?p=30296>

- **UTF-8 decoder capability and stress test**

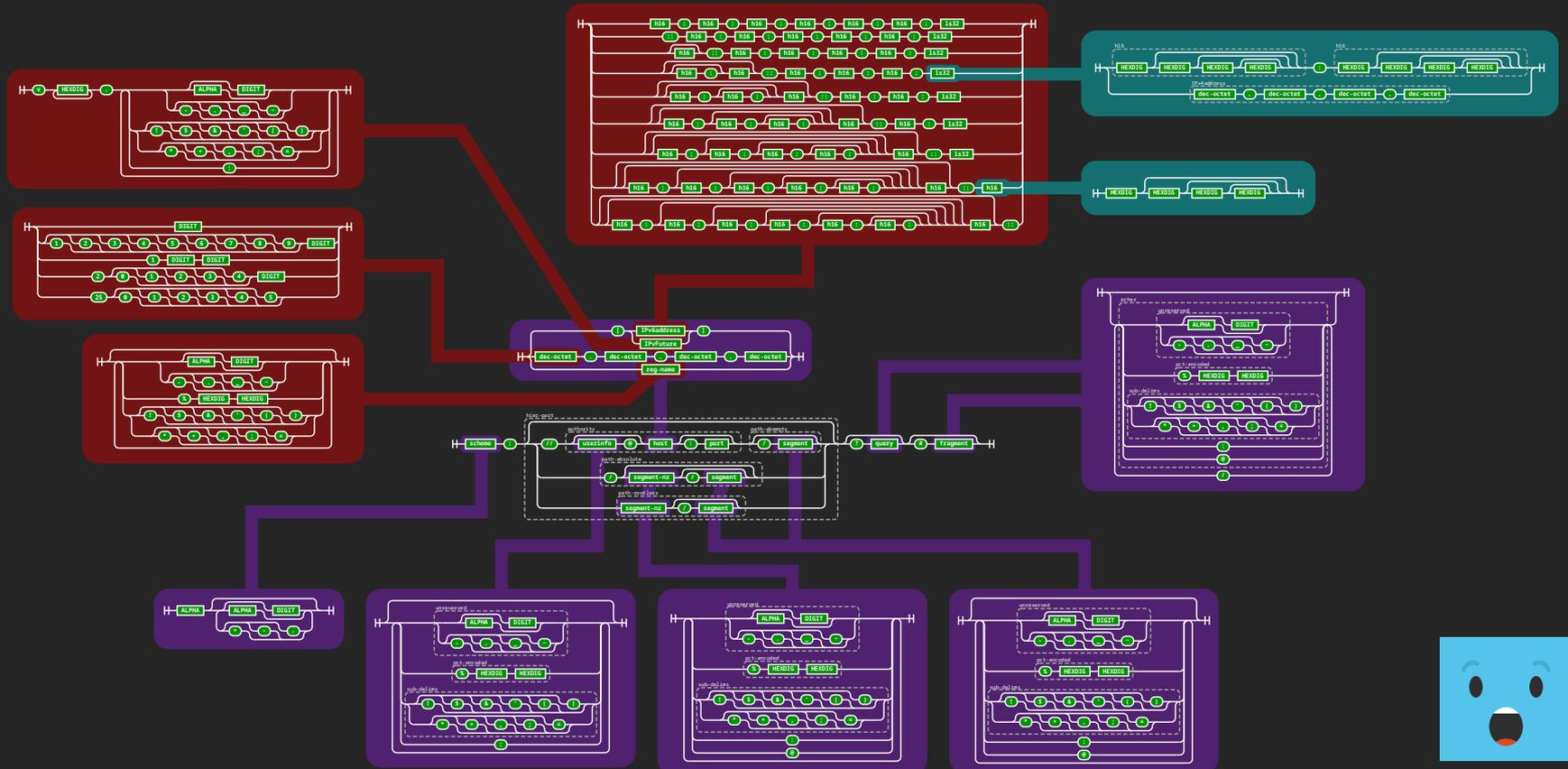
<https://www.cl.cam.ac.uk/~mgk25/ucs/examples/UTF-8-test.txt>



Les URLs



Grammaire d'une URI, RFC 3986



Un type sans type

- Représenté par / manipulé avec des chaînes de caractères
- Mélange des genres
 - URI → RFC 3986
 - IRI → RFC 3987
 - Systèmes de fichiers
 - Microsoft (MS-DOS, Windows, FAT, NTFS...)
 - Apple (MacOS, MacOSX, HFS...)
 - Les autres (Unix...)

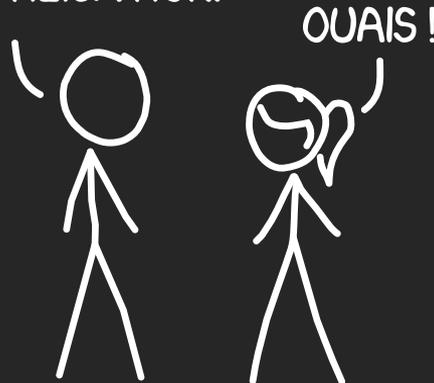


COMMENT LES STANDARDS PROLIFÈRENT

(EX : CHARGEURS, ENCODAGE DES CARACTÈRES, MESSAGERIE INSTANTANÉE, ETC)

SITUATION :
IL EXISTE
14 STANDARDS
CONCURRENTS.

14 ?! RIDICULE !
NOUS DEVONS DÉVELOPPER
UN STANDARD UNIVERSEL
COUVRANT TOUS LES CAS
D'UTILISATION.



BIENTÔT :

SITUATION :
IL EXISTE
15 STANDARDS
CONCURRENTS.

[HTTPS://XKCD.COM/927/](https://xkcd.com/927/)

Comment les standards prolifèrent



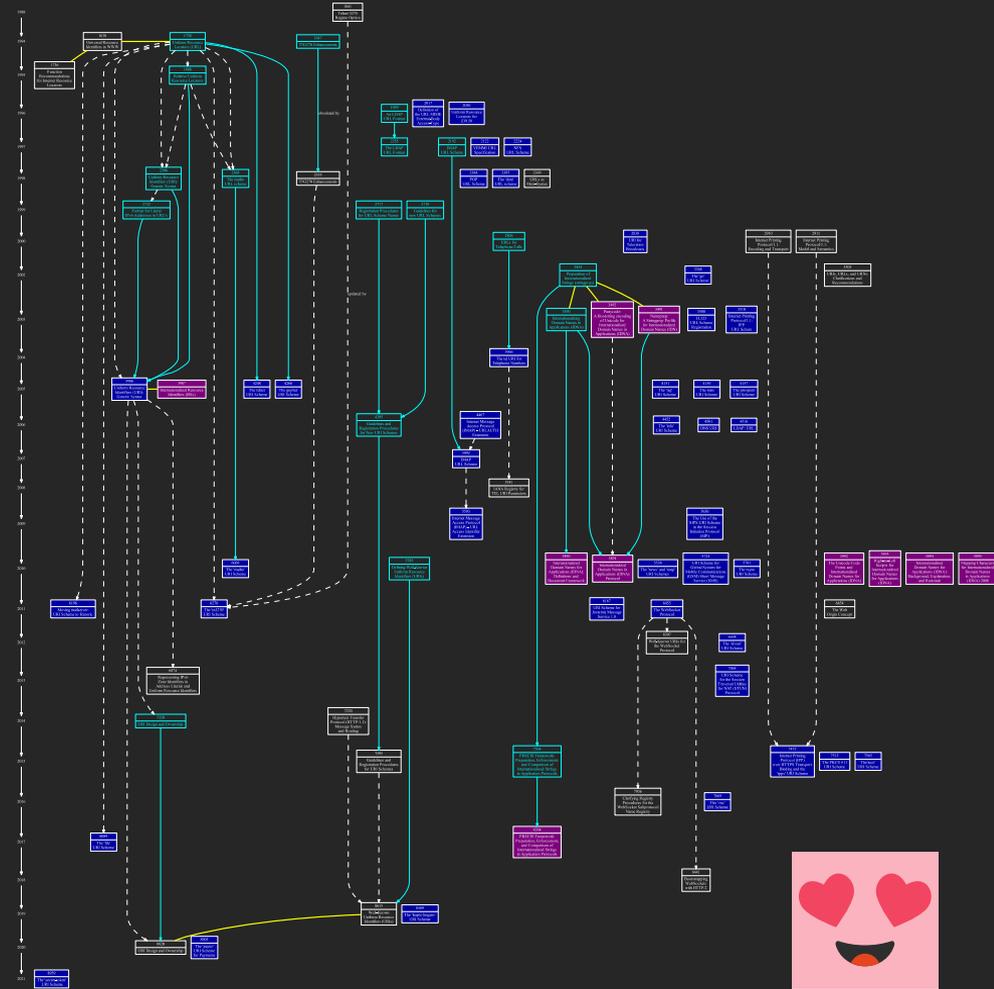
Un type bien standardisé ?

- Qui définit les URLs?
 - IETF
 - WHATWG
 - Unicode
 - W3C
- Le monde réel
 - navigateurs
 - robots (indexation...)
 - applications
 - sites web
 - etc.



IETF

- Depuis 1994
- 80+ RFC produites !
 - grammaires
 - internationalisation
 - schéma
- Stricte et généraliste



WHATWG



- Créé en 2004
 - Apple
 - Google
 - Mozilla
 - Microsoft
- Web platform test
 - <https://github.com/web-platform-tests/wpt>
- Spécifications
 - HTML Living Standard
 - URL Living Standard
- Approche
 - « vivante »
 - orientée web



Unicode



- Spécification UTS #46

- Transition

IDNA 2003 → IDNA 2008

- Limite par extension

Pour le .fr: ß à á â ã ä å æ ç è
é ê ë ì í î ï ñ ò ó ô õ ö ù
ú û ü ý ÿ æ

	IDNA 2003	UTS #46	IDNA 2008
öbb.at	✓	✓	✓
ÖBB.at	✓ A→a	✓ A→a	✗
v.com	✓	✓	✗
faß.de	✓ A→a	✓ A→a	✓
qәлп.com	~	✓	✓
Æbby.com	~	✓ A→a	✗

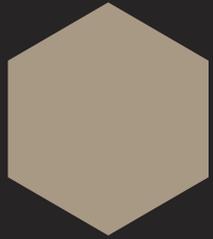
W3C

- WebPlatform.org
 - Adobe Apple Facebook Google HP
Microsoft Mozilla Nokia Opera W3C
 - Lancé en 2012, arrêté en 2015
- XHTML 2 vs HTML 5
- Recopie à peu près WHATWG...



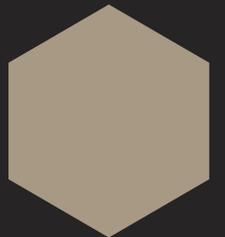
Opera



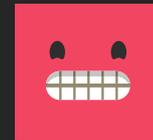


*Le W3C copie-colle parfois
notre travail sur son propre site web,
y appose son propre logo,
change le nom des rédacteurs,
etc.*

**DOMENIC DENICOLA - GOOGLE, ÉDITEUR WHATWG
10/02/2017 - REDDIT**



Pas simple d'interpréter une URL

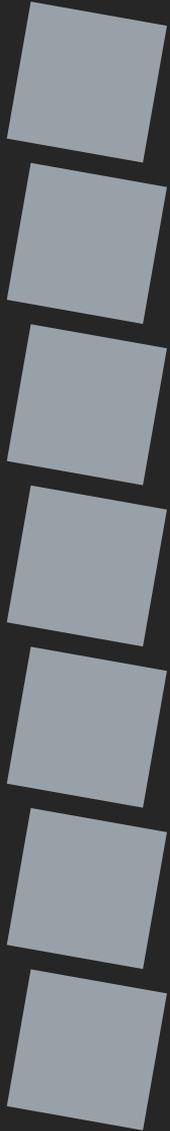
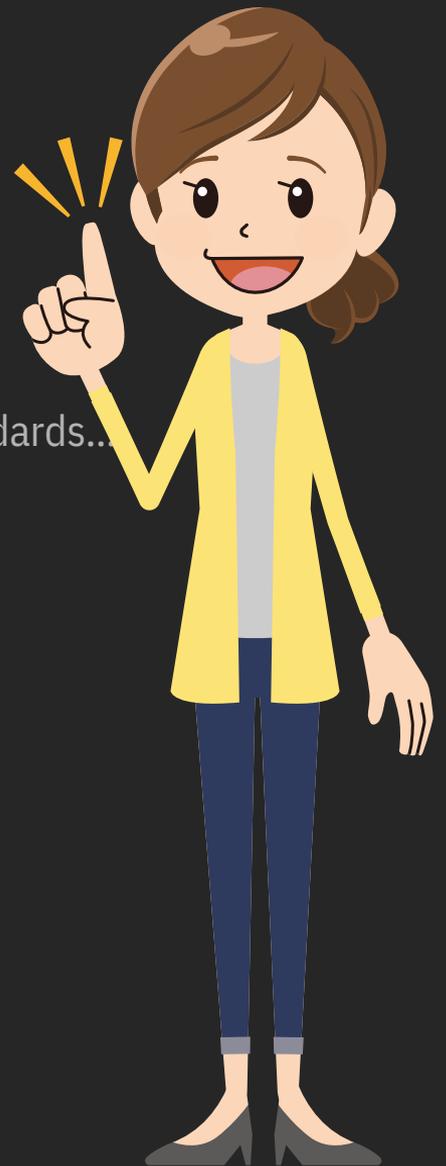


- `https://devfest.developers-group-dijon.fr/team`
Normalisation par le navigateur avant interprétation
- `http://user@example.com:81@daniel.haxx.se`

	<i>app</i>	<i>user</i>	<i>pass</i>	<i>host</i>	<i>port</i>
-	cURL	user	-	example.com	81
-	wget	user	-	example.com	<i>invalid!</i>
-	Safari	<i>invalid!</i>	-	<i>invalid!</i>	<i>invalid!</i>
-	Chrome	user@example.com	81	daniel.haxx.se	80

Alors, que faire ?

- **Il y a de la vie en dehors des navigateurs!**
Robots, bibliothèques, applications, standards, interprétation des standards...
- **Contrôler la génération d'URL**
Charge utile XSS...
- **Normaliser les URL**
N'autoriser qu'un sous-ensemble d'URL



Pour aller plus loin

- One URL standard please

<https://daniel.haxx.se/blog/2017/01/30/one-url-standard-please/>

- A new era of SSRF: exploiting URL parser in trending programming language

<https://www.blackhat.com/docs/us-17/thursday/us-17-Tsai-A-New-Era-Of-SSRF-Exploiting-URL-Parser-In-Trending-Programming-Languages.pdf>



MERCI!

- Merci à l'équipe du DevFest Dijon!
- Moi sur les internets
 - Github: <https://github.com/zigazou>
 - Twitter: [@zigazou](#)
 - Mail: zigazou@protonmail.com