Tu connais ce type ?

PRÉAMBULE

Après des années à arpenter les voies de sa stack favorite, on se surprend toujours à découvrir un comportement abscons au détour d'une ligne de code obscure. Plusieurs jours peuvent passer avant de réaliser qu'on ne comprenait pas vraiment des types de données manipulés en toute bonne foi.

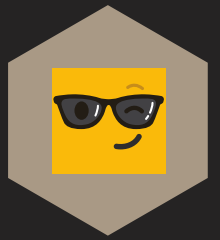
Les meilleures techniques de développement ne suffisent pas à vous mettre à l'abri d'un manque de connaissances sur un type de données.

Bienvenue dans l'enfer du code, aucun langage n'est à l'abri !

Un langage de programmation est censé être une façon conventionnelle de donner des ordres à un ordinateur.

Il n'est pas censé être obscur, bizarre et plein de pièges subtils (ça ce sont les attributs de la magie).

DAVE SMALL
ST MAGAZINE N°66 - 1992




HA!
HA!
HA!

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
  - 10, 12, 15, 18, 20, 24, 36, 48, 80, 128 bits…

Et des entiers non bornés ou presque

- GNU MP partout…
  - Python
  - Haskell
  - Maple
  - Mathematica
  - etc.
- Ou presque
  - JavaScript

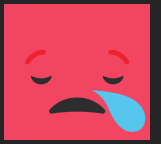
- Avantage
  - peu de limite
- Inconvénients
  - occupe plus de mémoire
  - plus lent
  - gestion mémoire délicate

Le complément à deux

- Technique la plus utilisée pour avoir des entiers signés

	non signé	*signé*
– 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

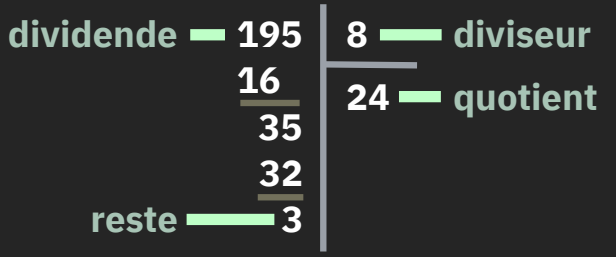
Pas d'entier natif en JavaScript

- Utilisation de nombres à virgule flottante
- Tout va bien tant que $-2^{53} \leqslant x < 2^{53}$
- Rien ne va plus quand on sort de cet intervalle…

```
for (let i = 2**53; i < 2**53 + 2; i++);
```

très amusant à lancer dans la console du navigateur !

La division euclidienne



- 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

Tu me fais tourner la tête

- Les entiers bornés fonctionnent comme des anneaux
  - entiers sur 8 bits → 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 ?

```c
#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

- Comportements spécifiques au langage
  - **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, plafonnement à **Infinity**

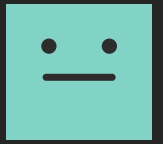
Un entier qui n’en est pas un

- 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…

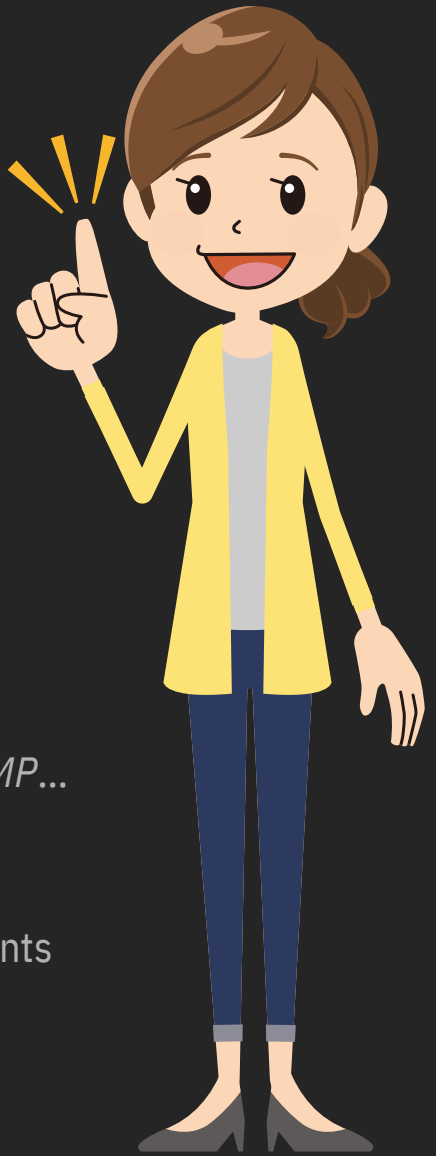
- Les opérations mathématiques ne les concernent pas

  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 !**



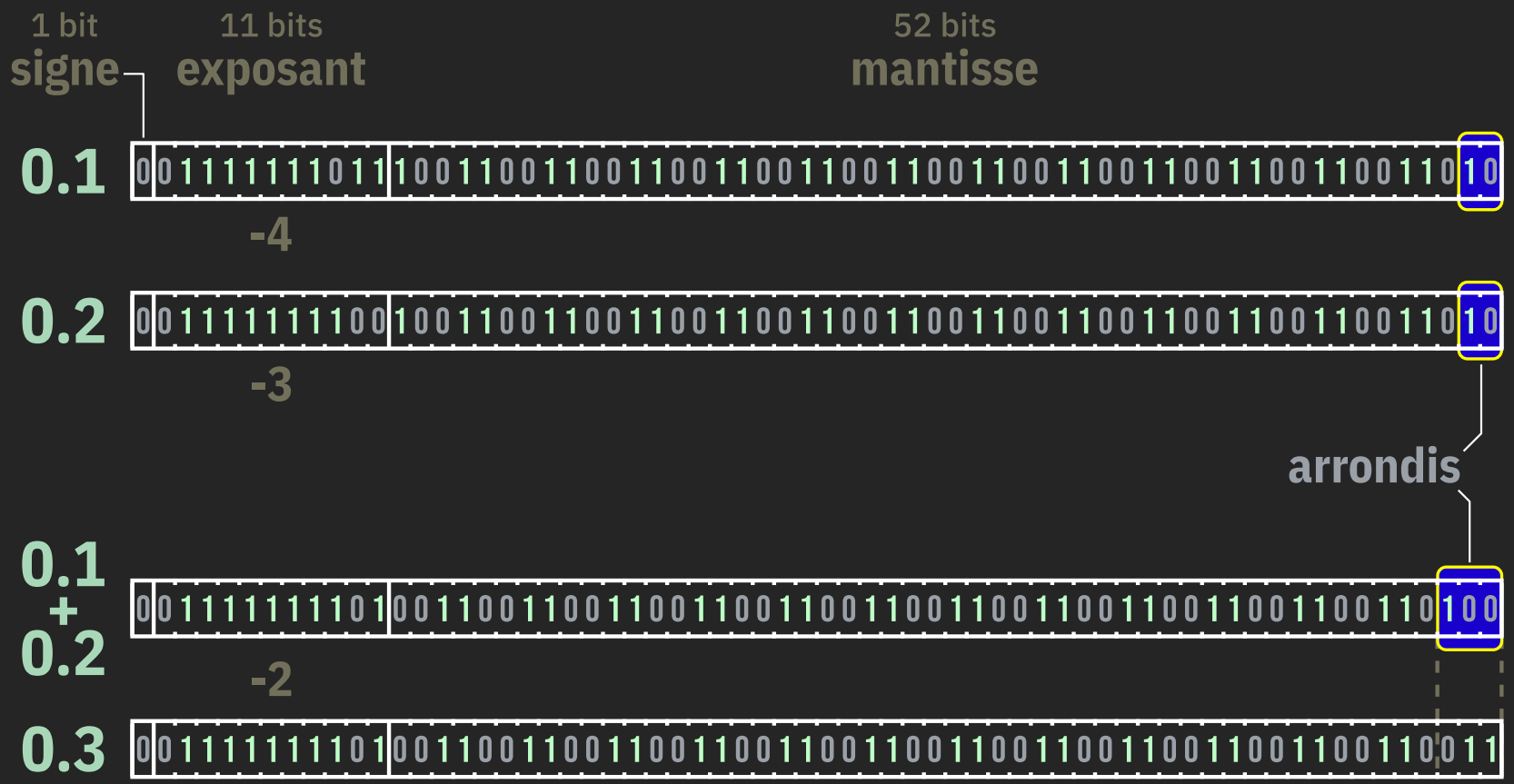
LES NOMBRES À VIRGULE FLOTTANTE OU FLOTTANTS

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



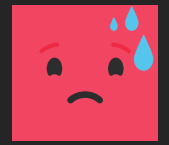
Cumul d'arrondis

Les vraies valeurs de 0.1 à 0.9

- 0.1000000000000000055511151231257827021181583404541015625
- 0.2000000000000000111022302462515654042363166809082031250
- 0.2999999999999999888977697537484345957636833190917968750
- 0.4000000000000000222044604925031308084726333618164062500
- 0.5
- 0.5999999999999999777955395074968691915273666381835937500
- 0.6999999999999999555910790149937383830547332763671875000
- 0.8000000000000000444089209850062616169452667236328125000
- 0.9000000000000000222044604925031308084726333618164062500

Les erreurs s'accumulent

- `for(let i=0, x=0; i<N; i++) x = x + 0.3;`
  - N = 100 x = 30.00000000000005
  - N = 1000 x = 300.0000000000056
  - 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) = 0.6000000000000001
```

Véritable valeur : `0.60000000000000008881784197001252323389053344726562 5`

```
0.1 +(0.2 + 0.1 + 0.2) = 0.6
```

Véritable valeur : `0.59999999999999977795539507496869191527366638183593 75`

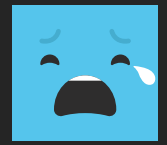
La multiplication n'est pas associative

```
(0.1 * 0.2)* 0.3  = 0.006000000000000001
```

Véritable valeur : 0.006000000000000009922618282587336580036208033561706542968750

```
 0.1 *(0.2 * 0.3) = 0.006
```

Véritable valeur : 0.006000000000000001249000902703301107976585626602172851562500



Parfois x × 1÷x ≠ 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

NaN : pas toujours un nombre

- Addition, soustraction : ∞ - ∞
- Multiplication, division : ∞ × 0, 0 ÷ ∞, 0 ÷ 0, ∞ ÷ 0
- Tout calcul avec **NaN** donne **NaN**

+	$-\infty$	$-n$	-0	$+0$	$+n$	$+\infty$	NaN
$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	NaN	NaN
$-n$	$-\infty$	$-2n$ $-\infty$	$-n$	$-n$	$+0$	$+\infty$	NaN
-0	$-\infty$	$-n$	-0	$+0$	n	$+\infty$	NaN
$+0$	$-\infty$	$-n$	$+0$	$+0$	n	$+\infty$	NaN
$+n$	$-\infty$	$+0$	n	n	$2n$ $+\infty$	$+\infty$	NaN
$+\infty$	NaN	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

−	$-\infty$	$-n$	-0	$+0$	$+n$	$+\infty$	NaN
$-\infty$	NaN	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	NaN
$-n$	$+\infty$	$+0$	$-n$	$-n$	$-2n$ $-\infty$	$-\infty$	NaN
-0	$+\infty$	n	$+0$	-0	$-n$	$-\infty$	NaN
$+0$	$+\infty$	n	$+0$	$+0$	$-n$	$-\infty$	NaN
$+n$	$+\infty$	$2n$ $+\infty$	n	n	$+0$	$-\infty$	NaN
$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

×	$-\infty$	$-n$	-0	$+0$	$+n$	$+\infty$	NaN
$-\infty$	$+\infty$	$+\infty$	NaN	NaN	$-\infty$	$-\infty$	NaN
$-n$	$+\infty$	n^2 $+\infty$	$+0$	-0	$-n^2$ $-\infty$	$-\infty$	NaN
-0	NaN	$+0$	$+0$	-0	-0	NaN	NaN
$+0$	NaN	-0	-0	$+0$	$+0$	NaN	NaN
$+n$	$-\infty$	$-n^2$ $-\infty$	-0	$+0$	n^2 $+\infty$	$+\infty$	NaN
$+\infty$	$-\infty$	$-\infty$	NaN	NaN	$+\infty$	$+\infty$	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

÷	$-\infty$	$-n$	-0	$+0$	$+n$	$+\infty$	NaN
$-\infty$	NaN	$+\infty$	$+\infty$	$-\infty$	$-\infty$	NaN	NaN
$-n$	$+0$	$+1$	$+\infty$	$-\infty$	-1	-0	NaN
-0	NaN	$+0$	NaN	NaN	-0	NaN	NaN
$+0$	NaN	-0	NaN	NaN	$+0$	NaN	NaN
$+n$	$-\infty$	-1	$-\infty$	$+\infty$	$+1$	$+\infty$	NaN
$+\infty$	NaN	$-\infty$	$-\infty$	$+\infty$	$+\infty$	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Mélange des genres : multiplication

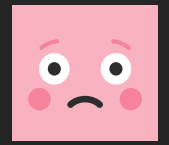
- `1.0 * "foo"`
  - **PHP < 8** `0`
  - **PHP ⩾ 8** *Fatal error: Uncaught DivisionByZeroError*
  - **Python** *Can't multiply sequence by non-int of type 'float'*
  - **Haskell** *No instance for (Fractional [Char])*
  - **C** *error: invalid operands to binary **
  - **JavaScript** NaN
- `1.0 * "1.0"`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

Mélange des genres : division

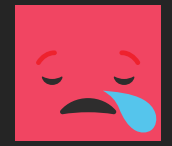
- `1.0 / "foo"`
  - **PHP < 8** INF *PHP Warning: Division by zero*
  - **PHP ⩾ 8** *Fatal error: Uncaught DivisionByZeroError*
  - **Python** *Unsupported operand type(s) for /: 'float' and 'str'*
  - **Haskell** *No instance for (Fractional [Char])*
  - **C** *error: invalid operands to binary /*
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

Petite subtilité : -0 et +0

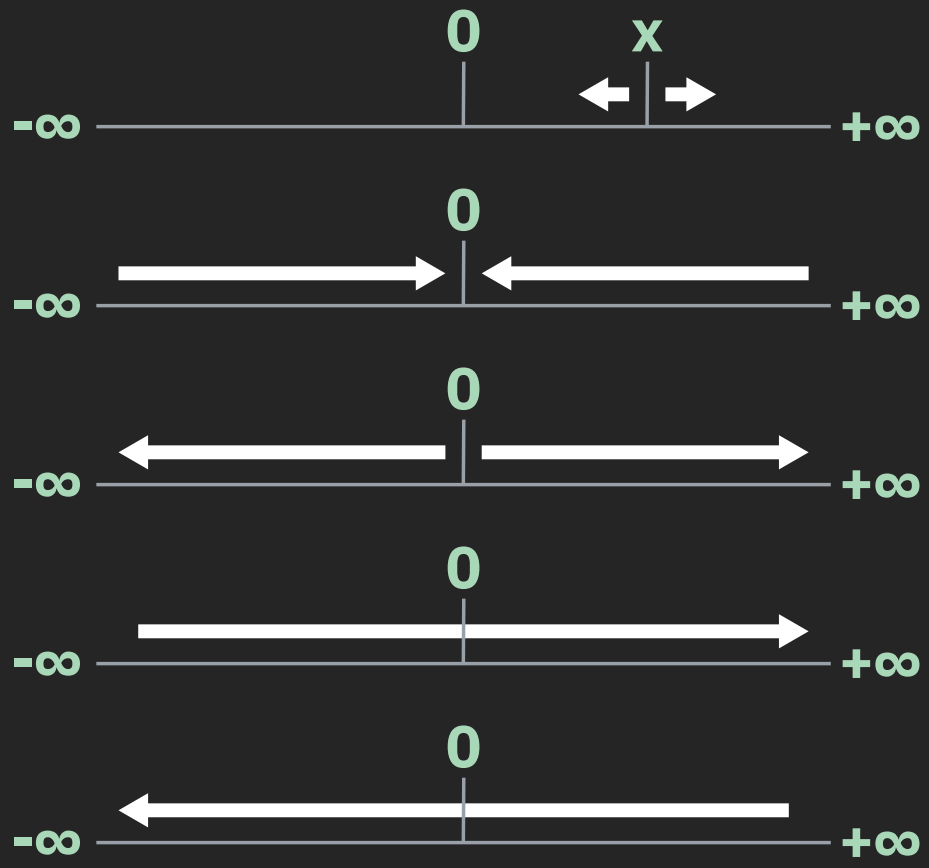
- Division
  - `1 / -0 == -Infinity`
  - `1 /  0 ==  Infinity`
  - `Infinity / -0 == -Infinity`
- Addition
  - `-0 + -0 == -0`
  - `0 + -0 == 0`
- Égalité
  - `-0 == +0`



Arrondir les angles



- Plusieurs types d’arrondis
  - Arrondi au plus proche
  - Vers 0
  - Vers ±∞
  - Vers +∞
  - Vers -∞

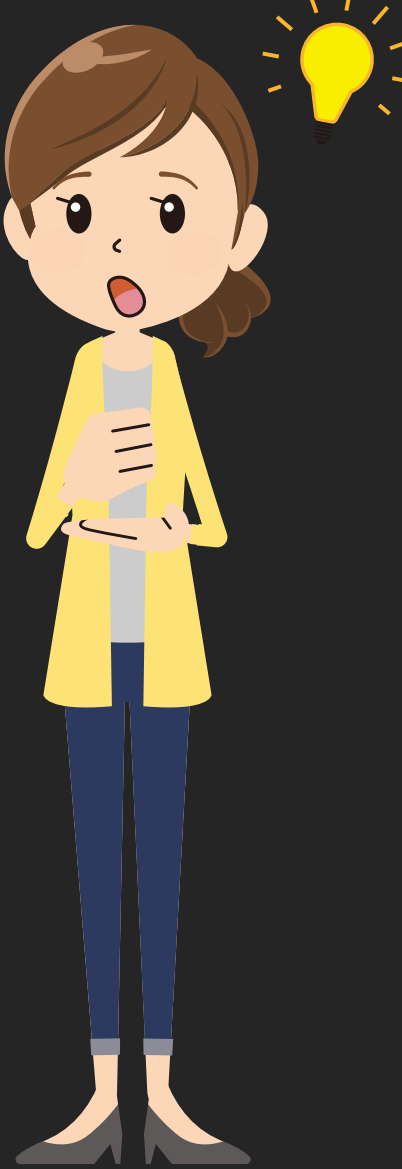


Alors, que faire ?

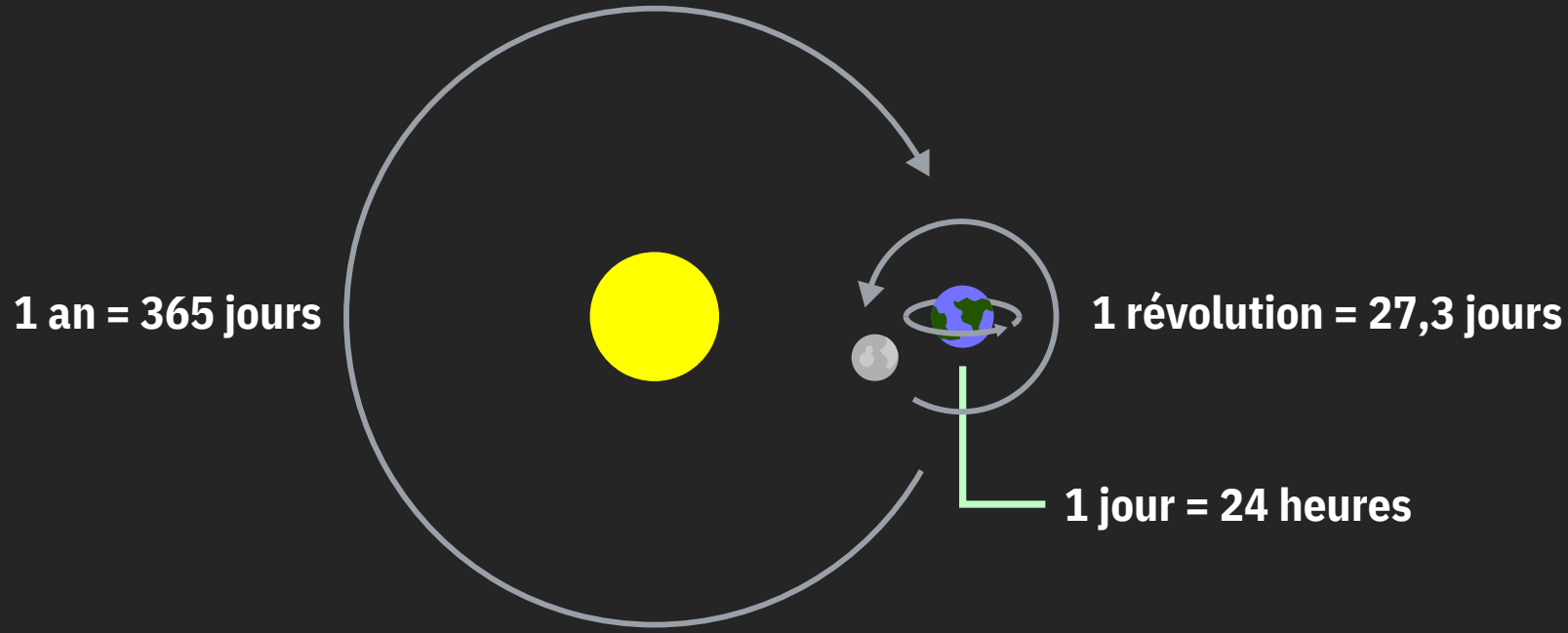
- **Connaître les flottants**
  Précision, comportements aux limites, arrondis, cas particuliers…
- **Tester les cas aux limites**
  Division par 0, signe de zéro, opérations sur l'infini…
- **Utiliser les types adéquats**
  Nécessité des nombres à virgule flottante, arrondis
- **Typer fortement, encapsuler**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



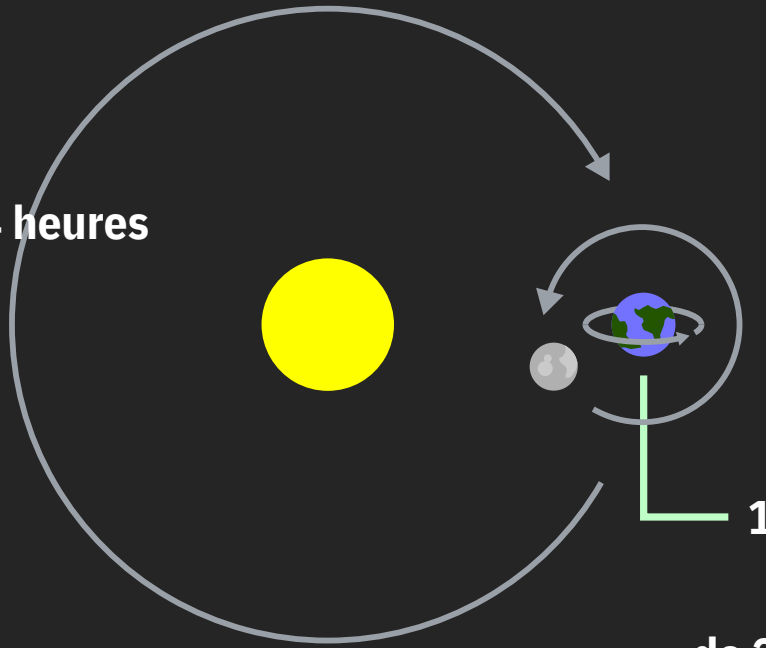
Les dates



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

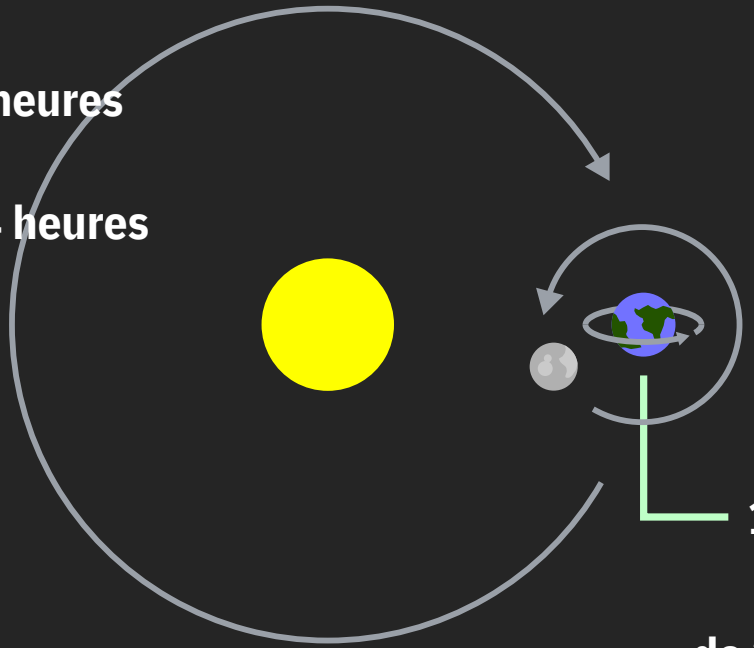
1 jour solaire
de 23 h 59 min. 39 sec.
à 24 h 0 min. 30 sec.

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
27,321582 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.

augmentation de 2 millisecondes/siècle

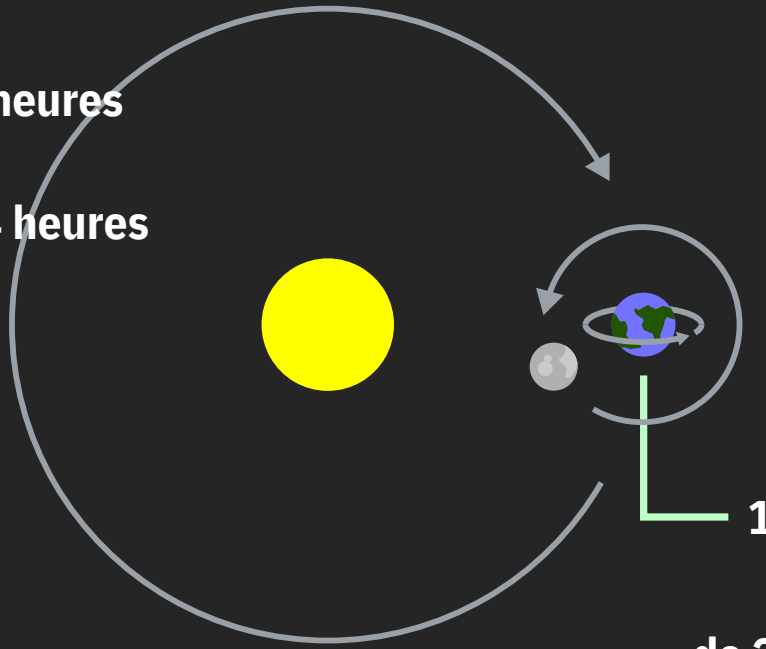
Prendre la bonne révolution

diminution de 0,5319 seconde/siècle

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
27,321582 jours

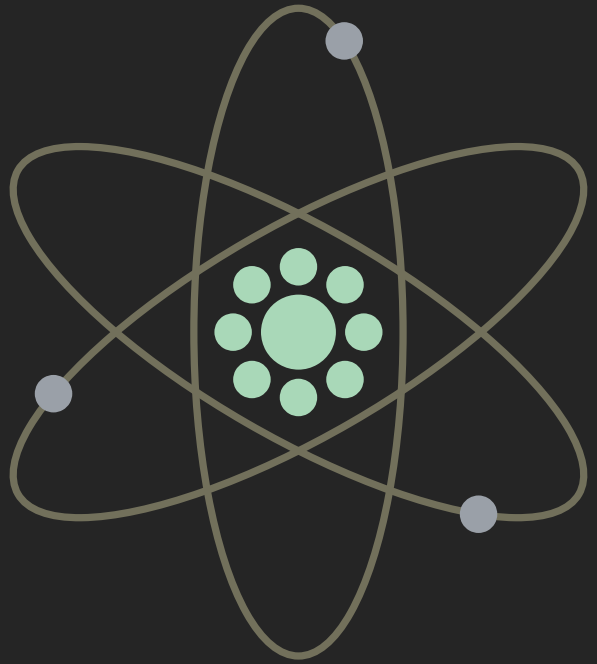
1 révolution = 27,3 jours

1 jour = 24 heures

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

augmentation de 2 millisecondes/siècle

Rien n'est figé dans le marbre



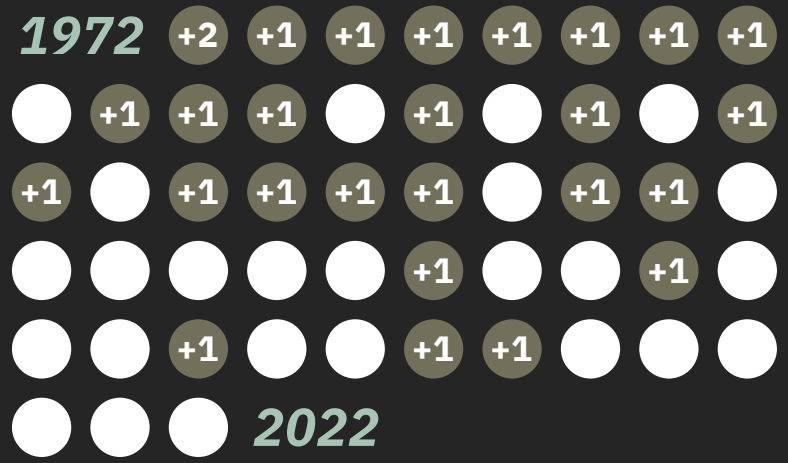
Et l'horloge atomique dans tout ça ?

Temps universel coordonné (UTC)

- Temps
  - Temps universel solaire : UT1
  - Temps atomique international : TAI
  - Temps universel coordonné : UTC
- Synchronisation et rattrapage
  - **1958** : synchronisation UT1 et TAI
  - **1972** : 10 secondes de décalage, création 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 en sursis ?

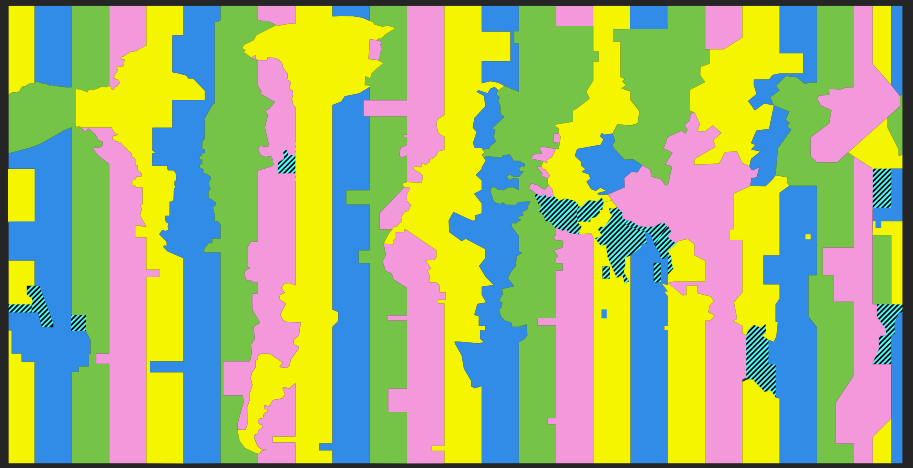


Histoire de bugs

- Calendrier Julien vs Grégorien
  Problème de calcul des années bissextiles amenant à la suppression de 11 jours
- Année zéro vs an 1
  21e siècle débutant en 2001, naissance de Jésus estimée entre -7 et -5
- 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…

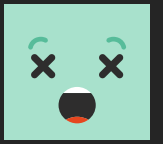
Fuseaux horaires, heure d'été…

- 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

- Langages
  - **Java 8 :** 1 ns
  - **Python :** 1 µs
  - **JavaScript**
    - API Date : 1 ms
    - API Performance : 5 µs
- Système de fichiers
  - **FAT :** 2 s
  - **EXT3 :** 1 s
  - **EXT4 :** 1 ms
  - **NTFS :** 0,1 µs
- Norme
  - **ISO 8601** : 1 µs ?



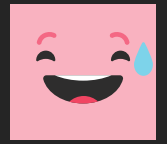
Précision réelle

- Atténuation d’attaques Meltdown/Spectre
- Atténuation de prise d’empreinte
- Instabilité de l’horloge système
  - précision du quartz
  - variations en fonction de la température
  - coordination NTP
- Comportement 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 ?

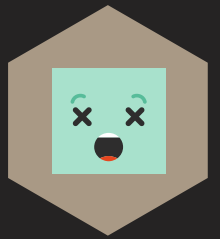
- Ça dépend **à qui** on pose la question
  - **PHP** 28, 29, 30 ou 31 jours
  - **Google** 30,4167 jours
  - **Convertlive.com** 30,4375 jours
- Ça dépend **comment** on pose la question
  - calcul naïf
  - fonctions dédiées

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 14e jour de la Lune
qui atteint cet âge le 21 mars
ou immédiatement après

CONCILE DE NICÉE, 325



ISO 8601:2004

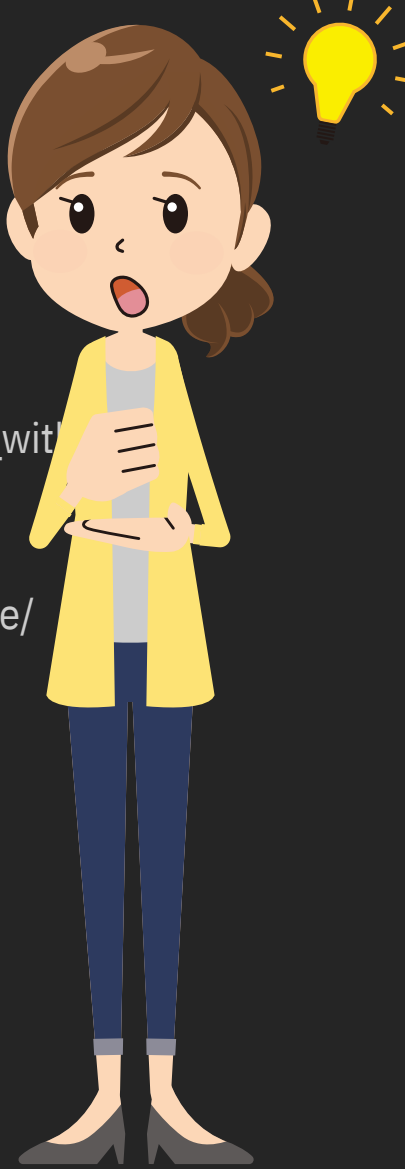
- Chaîne de caractères
  - difficulté à parser
- Calendrier grégorien proleptique
  - année zéro
  - flou avant 1582
  - seconde intercalaire
  - date ou intervalle de dates

Alors, que faire ?

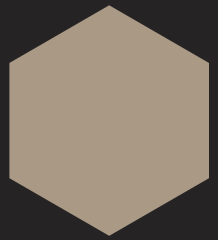
- **Considérer les problématiques**
  Années bissextiles, secondes intercalaires, référentiel, précision…
- **Faire appel aux normes et standards**
  ISO 8601, UTC…
- **Utiliser des bibliothèques**
  JavaScript : Luxon…
- **Mettre à jour régulièrement**
  Les secondes intercalaires ne peuvent pas être 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



Les chaînes de caractères

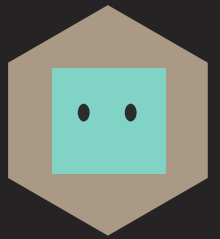


Une chaîne de caractères est à la fois

conceptuellement
une suite ordonnée de caractères

et physiquement
une suite ordonnée d'unités de code

CHAÎNE DE CARACTÈRES, WIKIPÉDIA



Concepts nés avant l'informatique

- **15e siècle** : typographie
  Caractères, signes, ligatures, espaces
- **16e siècle** : apparition des accents en français
- **19e siècle** : télégraphe
  Premières codifications et automatisations, codes de contrôle
- **1930** : téléscripteur
  Codes de contrôle, séparateurs, augmentation du nombre de caractères
- **Depuis le 19e 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…

UNICODE

U+0041	LATIN CAPITAL LETTER A
Point de code	*Désignation du caractère*

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_B KS_C_5601-1987 ISO-2022-KR EUC-KR ISO-2022-JP ISO-2022-JP-2 JIS_C6220-1969-jp JIS_C6220-1969-ro PT greek7-old latin-greek NF_Z_62-010_(1973) Latin-greek-1 ISO_5427 JIS_C6226-1978 BS_viewdata INIS INIS-8 INIS-cyrillic ISO_5427:1981 ISO_5428:1980 GB_1988-80 GB_2312-80 NS_4551-2 videotex-suppl PT2 ES2 MSZ_7795.3 JIS_C6226-1983 greek7 ASMO_449 iso-ir-90 JIS_C6229-1984-a JIS_C6229-1984-b JIS_C6229-1984-b-add JIS_C6229-1984-hand JIS_C6229-1984-hand-add JIS_C6229-1984-kana ISO_2033-1983 ANSI_X3.110-1983 T.61-7bit T.61-8bit ECMA-cyrillic CSA_Z243.4-1985-1 CSA_Z243.4-1985-2 CSA_Z243.4-1985-gr ISO_8859-6-E ISO_8859-6-I T.101-G2 ISO_8859-8-E ISO_8859-8-I CSN_369103 JUS_I.B1.002 IEC_P27-1 JUS_I.B1.003-serb JUS_I.B1.003-mac greek-ccitt NC_NC00-10:81 ISO_6937-2-25 GOST_19768-74 ISO_8859-supp ISO_10367-box latin-lap JIS_X0212-1990 DS_2089 us-dk dk-us KSC5636 UNICODE-1-1-UTF-7 ISO-2022-CN ISO-2022-CN-EXT UTF-8 ISO-8859-13 ISO-8859-14 ISO-8859-15 ISO-8859-16 GBK GB18030 OSD_EBCDIC_DF04_15 OSD_EBCDIC_DF03_IRV OSD_EBCDIC_DF04_1 ISO-11548-1 KZ-1048 ISO-10646-UCS-2 ISO-10646-UCS-4 ISO-10646-UCS-Basic ISO-10646-Unicode-Latin1 ISO-10646-J-1 ISO-Unicode-IBM-1261 ISO-Unicode-IBM-1268 ISO-Unicode-IBM-1276 ISO-Unicode-IBM-1264 ISO-Unicode-IBM-1265 UNICODE-1-1 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-3.0-Latin-1 ISO-8859-1-Windows-3.1-Latin-1 ISO-8859-2-Windows-Latin-2 ISO-8859-9-Windows-Latin-5 hp-roman8 Adobe-Standard-Encoding Ventura-US Ventura-International DEC-MCS IBM850 PC8-Danish-Norwegian IBM862 PC8-Turkish IBM-Symbols IBM-Thai HP-Legal HP-Pi-font HP-Math8 Adobe-Symbol-Encoding HP-DeskTop Ventura-Math Microsoft-Publishing Windows-31J GB2312 Big5 macintosh IBM037 IBM038 IBM273 IBM274 IBM275 IBM277 IBM278 IBM280 IBM281 IBM284 IBM285 IBM290 IBM297 IBM420 IBM423 IBM424 IBM437 IBM500 IBM851 IBM852 IBM855 IBM857 IBM860 IBM861 IBM863 IBM864 IBM865 IBM868 IBM869 IBM870 IBM871 IBM880 IBM891 IBM903 IBM904 IBM905 IBM918 IBM1026 EBCDIC-AT-DE EBCDIC-AT-DE-A 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

http://www.iana.org/assignments/character-sets/character-sets.xhtml 2021-01-04

Des 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** : 1991-2022
    Liste de caractères et d'algorithmes, prise en compte de la polysémie

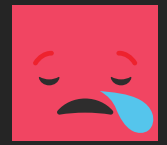
UNICODE

Unicode à la rescousse

- Liste de nombreux caractères
  - 144 697 caractères à la version 14, 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

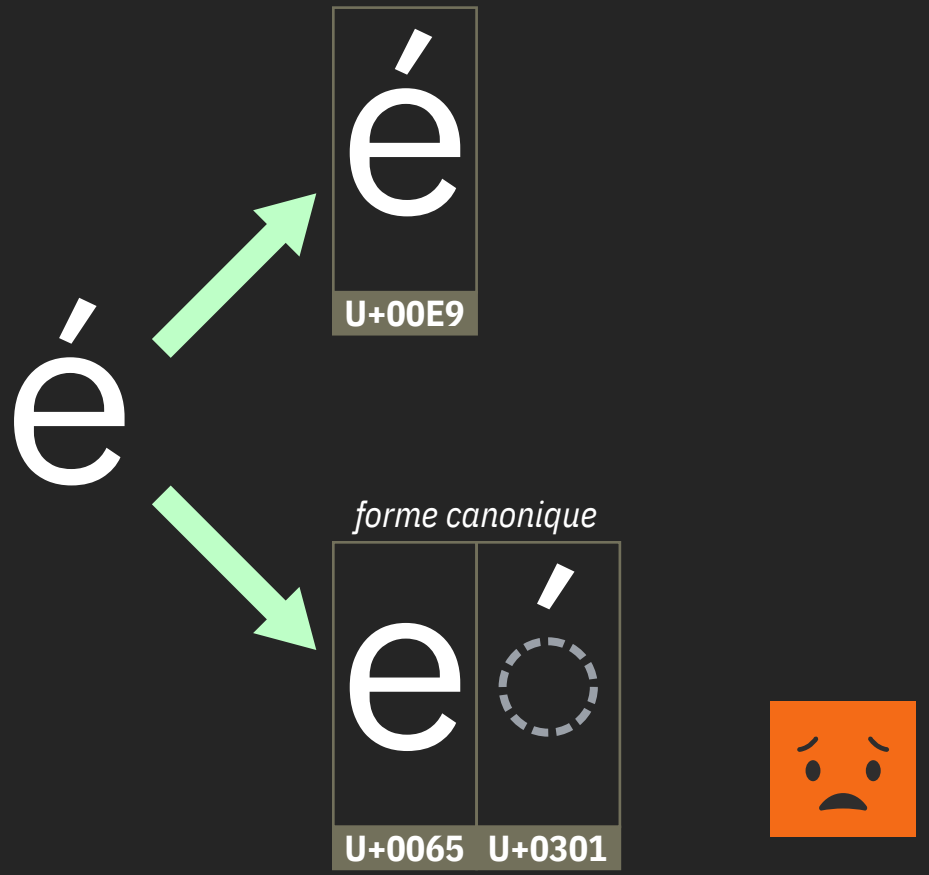
Quelques propriétés par caractère

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_Trinary_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



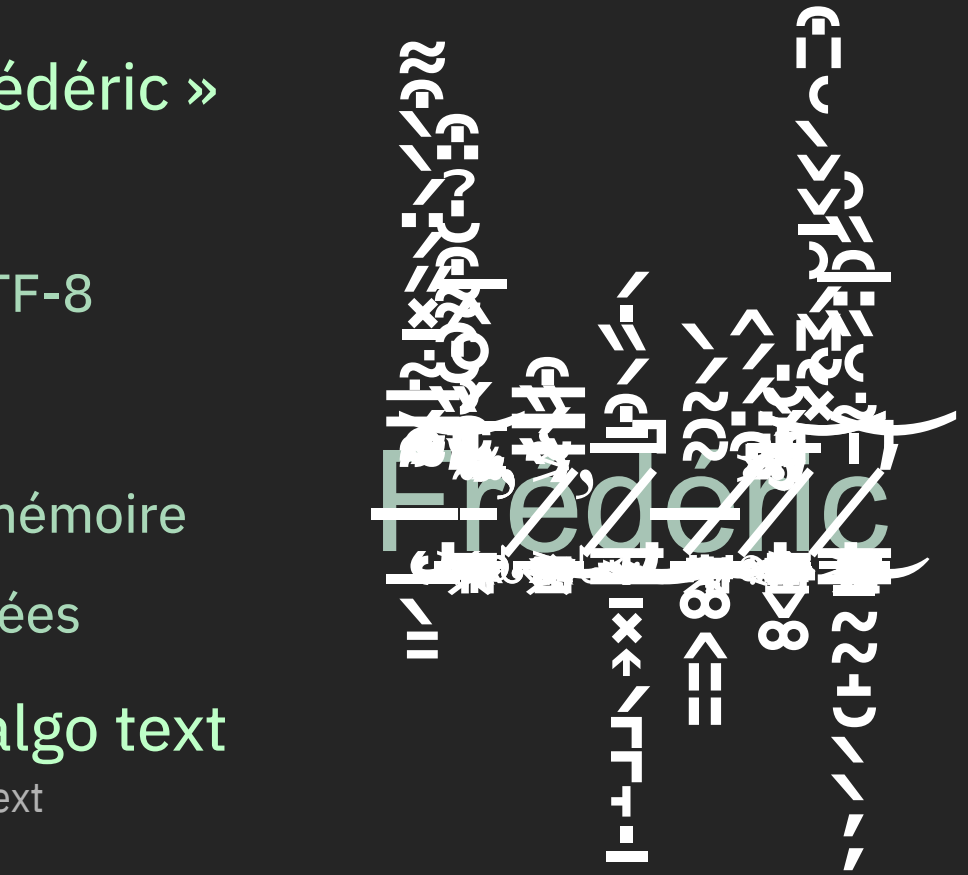
Plusieurs façons d'écrire un caractère

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



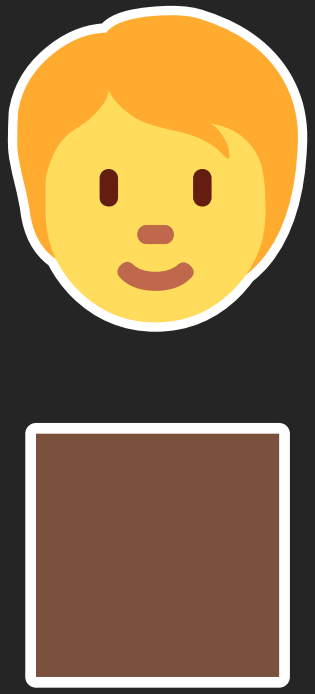
Abus de marques : Zalgo text

- 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



1 caractère

=



2 points de code

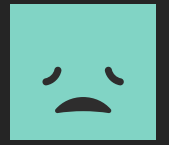
=

F0
9F
A7
91
F0
9F
8F
BF

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	𝔢	Minuscule mathématique gothique E
Lettre minuscule cyrillique tché abkhaze	ҽ	𝕖	Minuscule mathématique ajourée E
Symbole estimé	℮	𝖊	Minuscule mathématique gothique grasse E
Minuscule E de ronde	ℯ	𝖾	Minuscule mathématique sans empattement E
Minuscule E italique ajouré	𝑒	𝗲	Minuscule mathématique grasse sans empattement E
Lettre minuscule latine E gothique	𝔢	𝘦	Minuscule mathématique italique sans empattement E
Minuscule mathématique grasse E	𝐞	𝙚	Minuscule mathématique italique grasse sans empattement E
Minuscule mathématique italique E	𝑒	ⅇ	Minuscule mathématique à chasse fixe E
Minuscule mathématique italique grasse E	𝒆	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, ſt, 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, Ш, ♫, ∞, ◊, ⌂…
  - le point de code n’est pas un indicateur d’ordre !
  - plusieurs niveaux et 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 langues n’utilisent pas toutes les espaces 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**

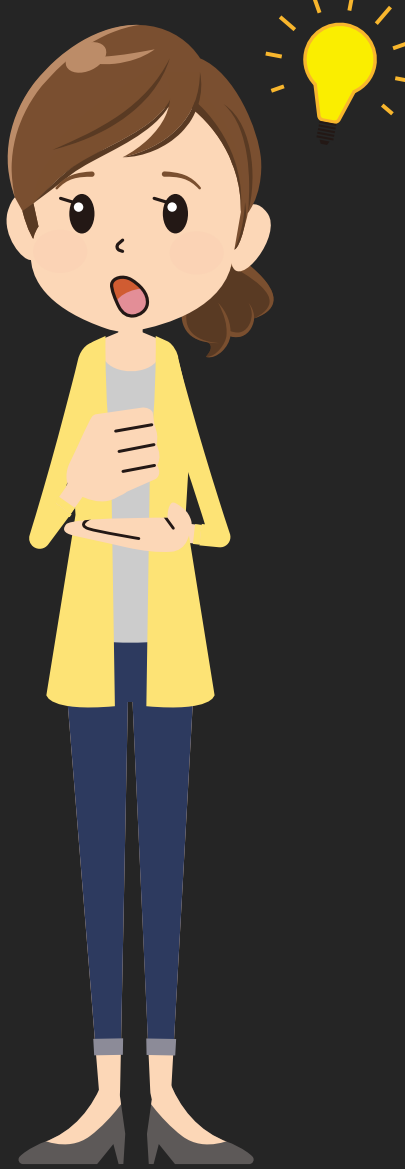
    EBCDIC !

Alors, que faire ?

- **Considérer la complexité d'un caractère**
  Polysémie, points de code, glyphe…
- **Suivre les évolutions d'Unicode**
  Unicode v14.0, CLDR, nouveaux caractères, nouvelles règles…
- **Nettoyer les chaînes**
  Translittération, forme NFC, remplacement/suppression des caractères indésirables
- **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
  aucune limite de taille, aucune contrainte hormis 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, plusieurs autres types de chaînes de caractères existent
- **JavaScript** tableau de mots de 16 bits
  aucune contrainte
- **PHP** tableau d'octets
  aucune 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)

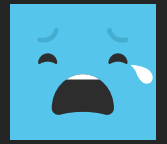
PHP et ses chaînes

- À la recherche du nombre perdu !
  - `"0000000042" == "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
  - **vrai** `SELECT 10 = "1e1";`
  - **vrai** `SELECT "1e1" = 10;`
  - **vrai** `SELECT 0 = "a";`
  - **faux** `SELECT "1e1" = "10";`
  - **faux** `SELECT md5('240610708') = md5('QNKCDZO');`
- Pas de comparaison sur la forme canonique

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



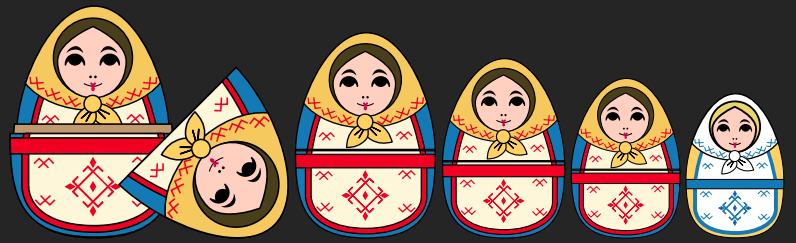
Littéralement

- 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

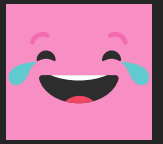
```
\0, \a, \b, \f, \n, \r, \t, \v, \", \&, \', \\, \NUL, \SOH, \STX, \ETX,
\EOT, \ENQ, \ACK, \BEL, \BS, \HT, \LF, \VT, \FF, \CR, \SO, \SI, \DLE,
\DC1, \DC2, \DC3, \DC4, \NAK, \SYN, \ETB, \CAN, \EM, \SUB, \ESC, \FS,
\GS, \RS, \US, \SP, \DEL, \^@, \^A, \^B, \^C, \^D, \^E, \^F, \^G, \^H,
\^I, \^J, \^K, \^L, \^M, \^N, \^O, \^P, \^Q, \^R, \^S, \^T, \^U, \^V,
\^W, \^X, \^Y, \^Z, \^[, \^\, \^], \^^, \^_, \99999, \o777777, \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\\\\\\!\\\\\\\\\"");'
```

8 6 9

A) Hello, World!

B) \Hello, World\!\

C) \\Hello, World\!\\

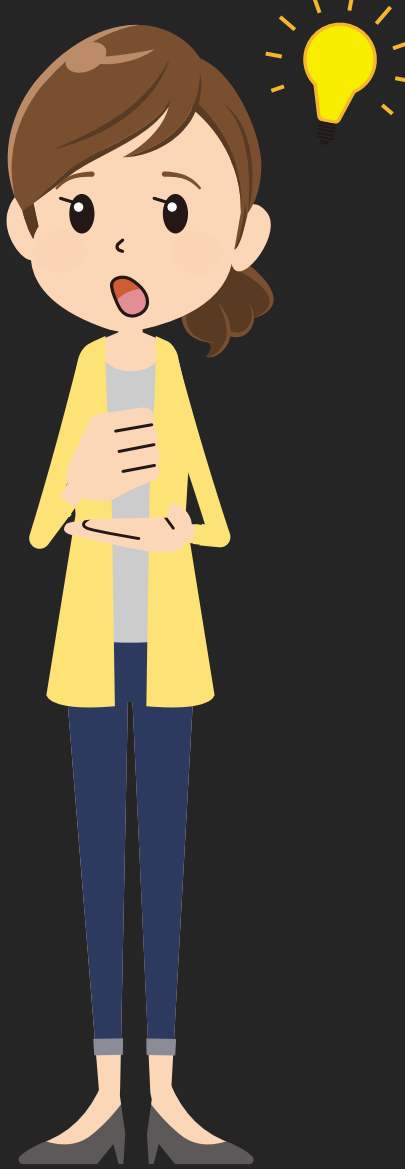
D) \\\\Hello, World\\!\\\\

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 !

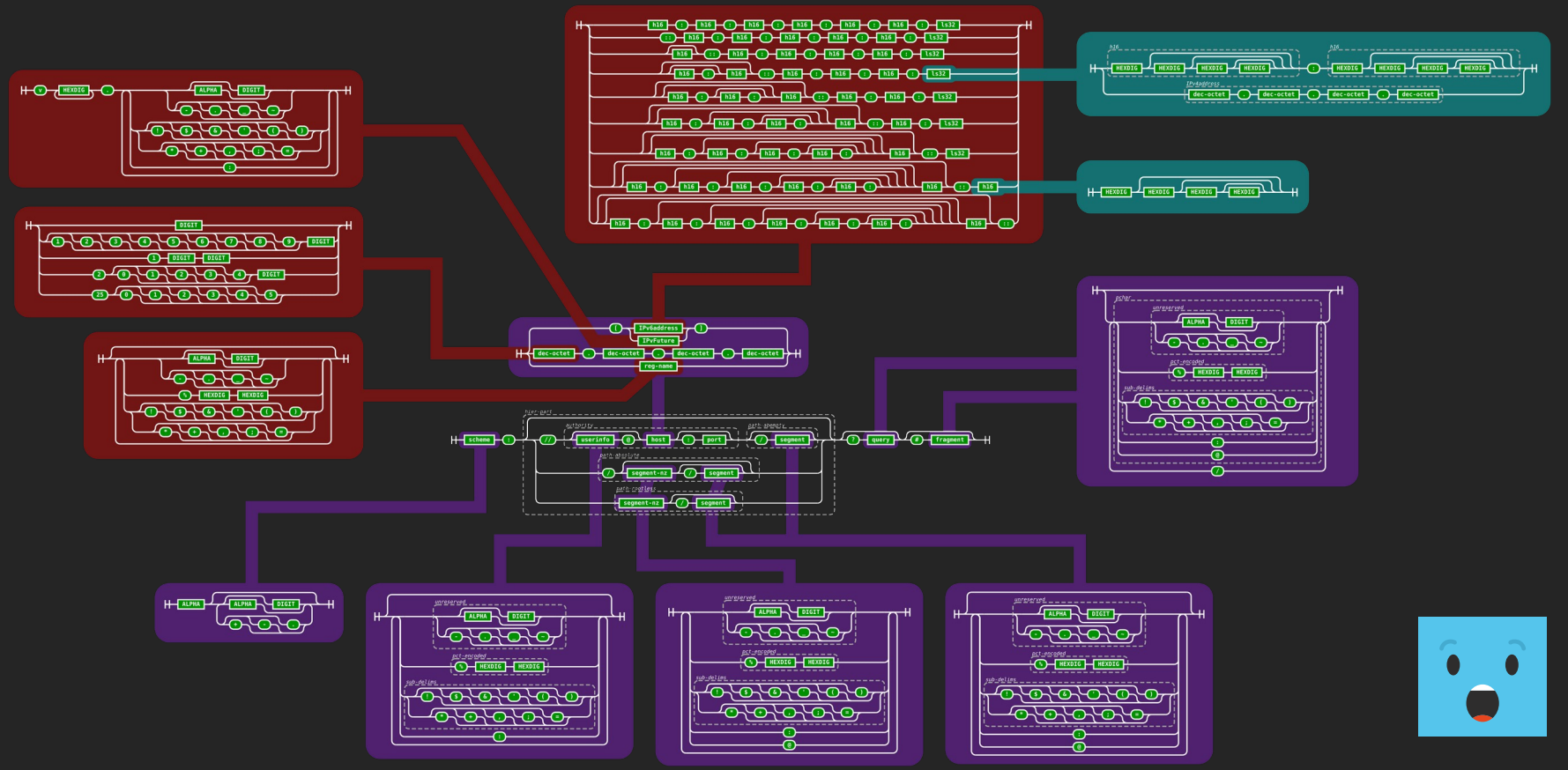
Pour aller plus loin

- 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

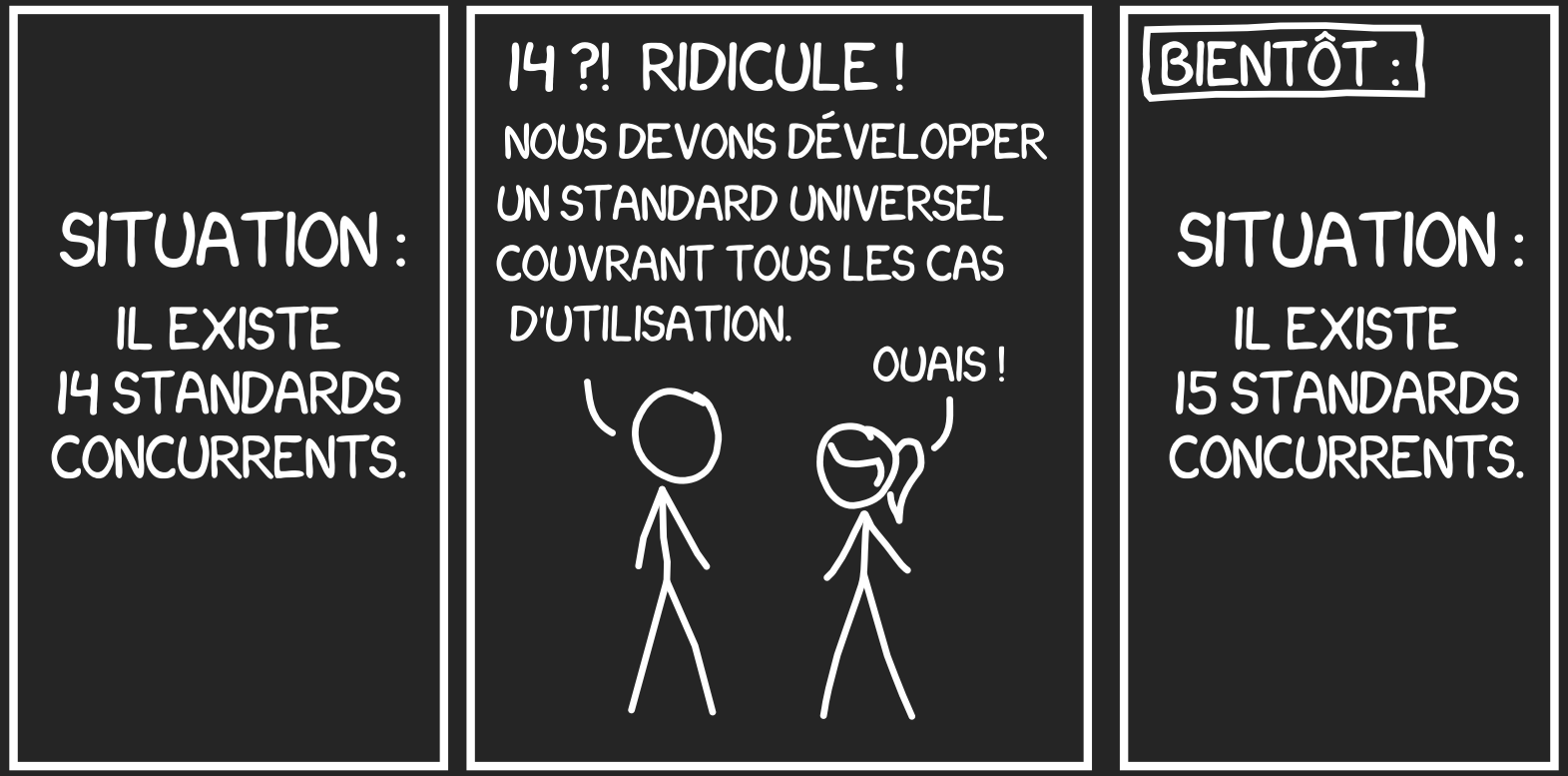


Une regex pour IRI, RFC 3987

```
/^[a-z](?:[-a-z0-9\+\.])*:(?:\/\/(?:(?:%[0-9a-f][0-9a-f]|[-a-z0-9\._~\x{A0}-\x{D7FF}\x{F900}-\x{FDCF}\x{FDF0}-\x{FFEF}\x{10000}-\x{1FFFD}\x{20000}-\x{2FFFD}\x{30000}-\
x{3FFFD}\x{40000}-\x{4FFFD}\x{50000}-\x{5FFFD}\x{60000}-\x{6FFFD}\x{70000}-\x{7FFFD}\x{80000}-\x{8FFFD}\x{90000}-\x{9FFFD}\x{A0000}-\x{AFFFD}\x{B0000}-\x{BFFFD}\x{C0000}-\
x{CFFFD}\x{D0000}-\x{DFFFD}\x{E1000}-\x{EFFFD}!\$&'\(\)\*\+,;=:])*@)?(?:\[(?:(?:(?:[0-9a-f]{1,4}:){6}(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-
9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|::(?:[0-9a-f]{1,4}:){5}(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|
25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|(?:[0-9a-f]{1,4})?::(?:[0-9a-f]{1,4}:){4}(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-
9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|(?:(?:[0-9a-f]{1,4}:){0,1}[0-9a-f]{1,4})?::(?:[0-9a-f]{1,4}:){3}(?:[0-9a-f]{1,4}:[0-9a-
f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|(?:(?:[0-9a-f]{1,4}:){0,2}[0-9a-f]{1,4})?::(?:[0-
9a-f]{1,4}:){2}(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|(?:(?:[0-9a-
f]{1,4}:){0,3}[0-9a-f]{1,4})?::[0-9a-f]{1,4}:(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4]
[0-9]|25[0-5])){3})|(?:(?:[0-9a-f]{1,4}:){0,4}[0-9a-f]{1,4})?::(?:[0-9a-f]{1,4}:[0-9a-f]{1,4}|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|
1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3})|(?:(?:[0-9a-f]{1,4}:){0,5}[0-9a-f]{1,4})?::[0-9a-f]{1,4}|(?:(?:[0-9a-f]{1,4}:){0,6}[0-9a-f]{1,4})?::)|v[0-9a-f]+\.[-a-z0-9\._~!\$&'\
(\)\*\+,;=:]+)\]|(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])(?:\.(?:[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5])){3}|(?:%[0-9a-f][0-9a-f]|[-a-z0-9\._~\x{A0}-\
x{D7FF}\x{F900}-\x{FDCF}\x{FDF0}-\x{FFEF}\x{10000}-\x{1FFFD}\x{20000}-\x{2FFFD}\x{30000}-\x{3FFFD}\x{40000}-\x{4FFFD}\x{50000}-\x{5FFFD}\x{60000}-\x{6FFFD}\x{70000}-\
x{7FFFD}\x{80000}-\x{8FFFD}\x{90000}-\x{9FFFD}\x{A0000}-\x{AFFFD}\x{B0000}-\x{BFFFD}\x{C0000}-\x{CFFFD}\x{D0000}-\x{DFFFD}\x{E1000}-\x{EFFFD}!\$&'\(\)\*\+,;=:]|[\/\?])*)?$/i
```

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…)



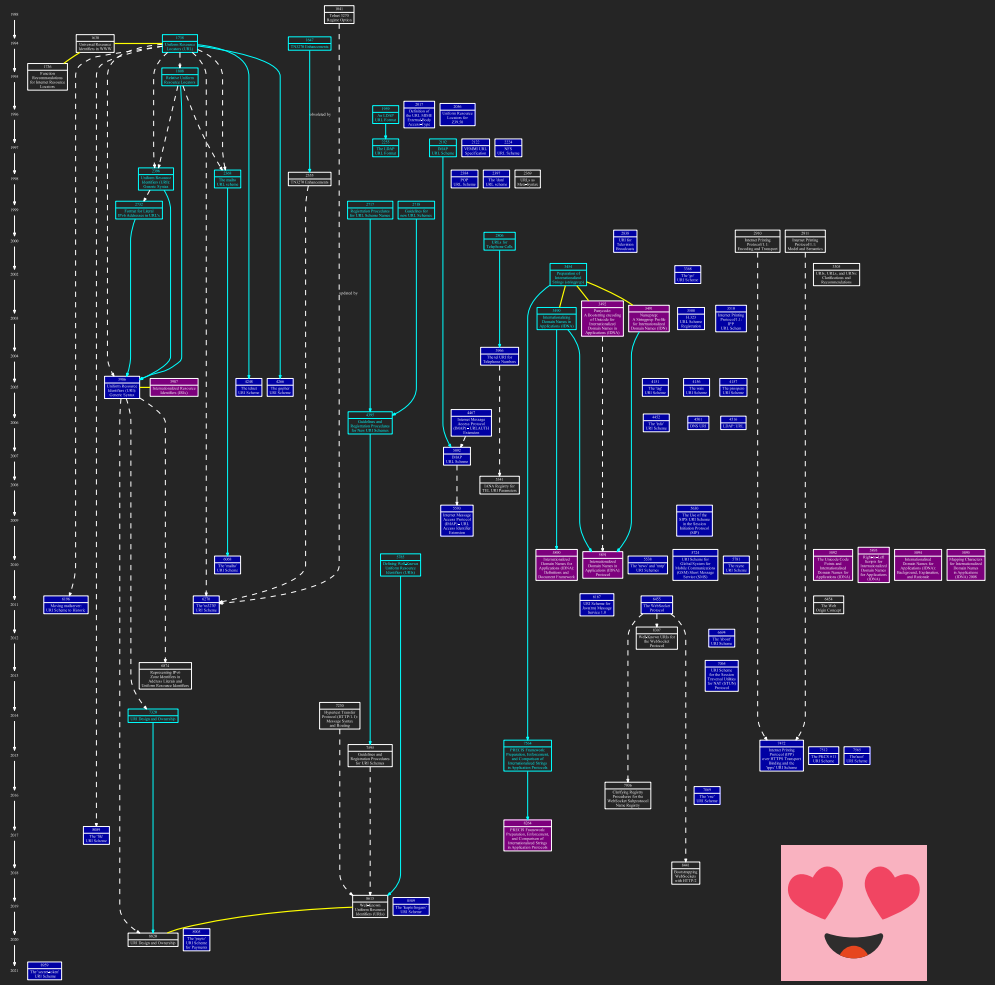
Comment les standards prolifèrent

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
- Plus de 80 RFC produites !
  - grammaires
  - internationalisation
  - schéma
- Approche 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
  - IDNA2003 → IDNA2008

	IDNA 2003	UTS #46	IDNA 2008
öbb.at	✔	✔	✔
ÖBB.at	✔ A→a	✔ A→a	✘
√.com	✔	✔	✘
ſaß.de	✔ A→a	✔ A→a	✔
qəлп.com	~	✔	✔
Ⱥbby.com	~	✔ A→a	✘

W3C

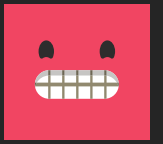
- WebPlatform.org
  - Adobe, Apple, Facebook, Google, HP, Microsoft, Mozilla, Nokia, Opera, W3C
  - Lancé en 2012, arrêté en 2015
- XHTML 2 vs HTML 5
- Recopie à peu près WHATWG…

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

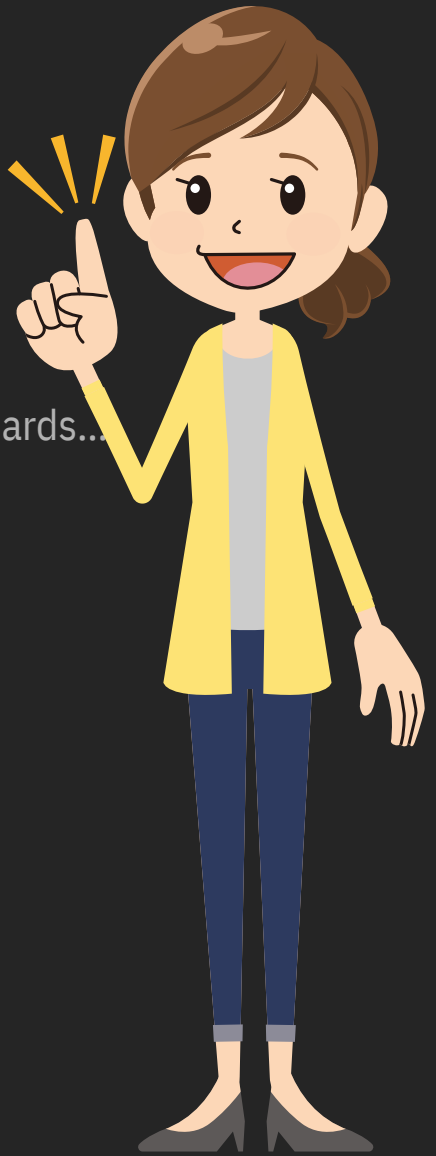
Pas simple d'interpréter une URL

- https:\/\/www.codeursenseine.com\2021
  Normalisation par le navigateur avant interprétation
- http://user@example.com:81@daniel.haxx.se
  - *app* *user* *pass* *host* *port*
  - cURL user - example.com 81
  - wget user - example.com *invalid !*
  - Safari *invalid !* - *invalid !* *invalid !*
  - Chrome user@example.com 81 daniel.haxx.se 80



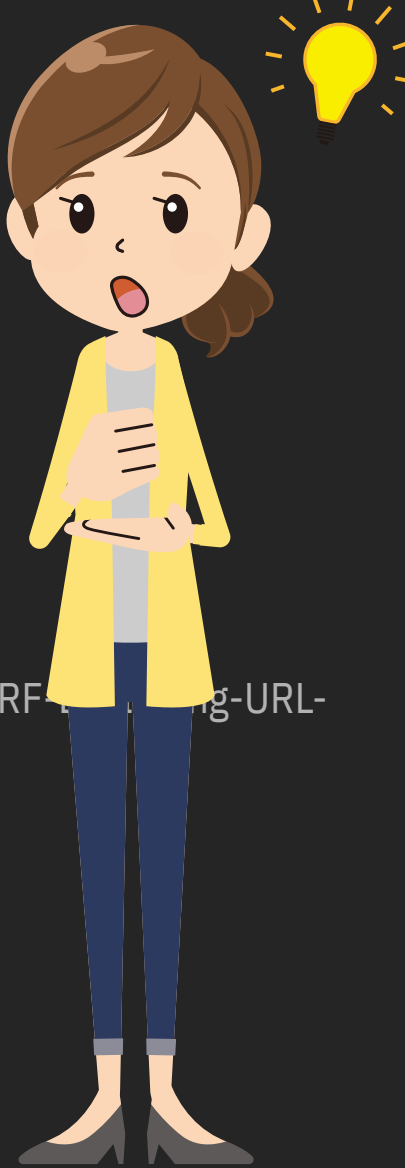
Alors, que faire ?

- 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



Informatique :

Alliance d'une science inexacte et d'une activité humaine faillible

LUC FAYARD

DICTIONNAIRE IMPERTINENT DES BRANCHÉS

MERCI !

- Merci à l'équipe de Codeurs en Seine
- Moi sur les internets
  - Github : https://github.com/zigazou
  - Twitter : @zigazou
  - Mail : zigazou@protonmail.com