

IFT1015 Programmation 1

type texte

Marc Feeley

(avec ajouts de Aaron Courville et Pascal Vincent)

Types de données

- En plus des **nombre**s, JS offre les types suivants :
 - Textes ou chaînes de caractères : p.ex. **"allo"**
 - Booléens ou valeurs de vérité : **true** et **false**
 - Les objets, **null** et **undefined**

Type texte

Texte

- Un **texte (string)** c'est une donnée qui représente une **séquence (ou chaîne) de caractères**
- C'est utile pour stocker des informations textuelles et les communiquer à des humains :
 - Noms : p.ex. "**Jean Coutu**"
 - Adresses postales : p.ex. "**7 rue Langlois**"
 - Numéros de dossier : p.ex. "**0001296-ETU**"
 - Messages d'information : p.ex. "**Le vol AC870 pour Paris décolle à 8h30**"

Texte

- Un programme peut contenir des **constantes littérales textuelles**
- Il y a deux syntaxes pour les littéraux textuels : avec délimiteurs **guillemets** ou **apostrophes**
- Syntaxe : **"***<caractères>***"** ou **'***<caractères>***'**
- *<caractères>* est une séquence de caractères

```
> "L'été est trop court."  
"L'été est trop court."
```


Texte

- Un littéral textuel ne doit pas contenir le caractère délimiteur (" ou ') ou être brisé sur plusieurs lignes
- Pour inclure un " ou ' , ou avoir un texte de plus d'une ligne on se sert d'un **échappement spécial**
 - \ " pour inclure un "
 - \ ' pour inclure un '
 - \\ pour inclure un \
 - \n pour inclure une «fin de ligne»

Unicode

- **Unicode** est un standard d'encodage des caractères qui contient les lettres de **toutes les langues du monde**, des symboles, accents, ...
- Tout caractère Unicode peut être mis dans un texte avec l'échappement **\uXXXX** où le *xxxx* est le code du caractère en notation hexadécimale

```
> print("\u03c0 est un nombre irrationnel")
  π est un nombre irrationnel
> print("Le \u265e a pris le \u2659")
  Le ♘ a pris le ♙
```


Unicode

	000	001	002	003	004	005	006	007
0	NUL	DLE	SP	0	@	P	`	p
1	STX	DC1	!	1	A	Q	a	q
2	SOT	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	END	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(8	H	X	h	x
9	HT	EM)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[k	{
C	FF	FS	,	<	L	\	l	
D	CR	GS	-	=	M]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

ASCII

	037	038	039	03A	03B	03C	03D	03E	03F
0			ı	Π	û	π	δ	ə	κ
1			Α	Ρ	α	ρ	θ	ə	Q
2			B		β	ς	Υ	ω	c
3			Γ	Σ	γ	σ	Υ	ω	j
4	'	'	Δ	T	δ	τ	ÿ	q	Θ
5	,	“	E	Y	ε	υ	φ	q	€
6			Α	Z	Φ	ζ	φ	ω	h
7			H	X	η	χ	χ	ς	Đ
8			Ε	Θ	Ψ	θ	ψ	Q	2
9			Η	I	Ω	ι	ω	Q	2
A	ı		I	K	İ	κ	ï	Ç	X
B	ı		Λ	ÿ	λ	ü	ς	x	ı
C	ı		Ο	M	ά	μ	ó	F	δ
D	ı		N	έ	ν	ύ	F	δ	ı
E	;	Y	Ξ	ή	ξ	ώ	ı	†	€
F		Ω	O	ı	o		ı	†	ı

	260	261	262	263	264	265	266	267	268	269	26A	26B	26C
0	☀	☐	☼	☼	♀	♂	♠	♣	♠	♠	♠	♠	♠
1	☂	☑	☑	☑	♂	♂	♥	♣	♠	♠	♠	♠	♠
2	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
3	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
4	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
5	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
6	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
7	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
8	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
9	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
A	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
B	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
C	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
D	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
E	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠
F	☂	☑	☑	☑	♂	♂	♠	♣	♠	♠	♠	♠	♠

Opérations sur les string

Nombreuses fonctions

- Soit `s` une expression de type `string`
- `s.length` est la longueur (nb de caractères)
Ex: `"Hello World!".length`
vaut 12
- Possibilité de comparer deux `string`
(voir opérateurs de comparaison plus bas)

Extraction de sous-chaînes

- Soit `s` une expression de type `string`
- `s.charAt(i)` retourne le caractère à la position `i`
les caractères sont numérotés de 0 à longueur-1

Ex: `"Hello World!".charAt(0)` vaut `"H"`

`"Hello World!".charAt(11)` vaut `"!"`

- `s.substring(pos_debut, pos_fin_exclue)`
retourne une sous-chaîne

Ex: `"Bonjour".substring(3,6)` vaut `"jou"`

↑ ↑
3 6

Textes numériques

Textes numériques

- Un texte qui contient une séquence de caractères correspondant à la syntaxe d'un nombre, un **texte numérique**, c'est **différent d'un nombre**

"123" \neq 123

- Cependant, sous certaines conditions, certains opérateurs font automatiquement la **conversion** des textes numériques en nombres ou vice-versa

"123" \rightarrow 123
conversion

123 \rightarrow "123"
conversion

Textes numériques

- À l'exception de l'opérateur **+** binaire, les opérateurs numériques (**+** unaire, **-**, *****, **/**, **%**, ...) font automatiquement la conversion des opérandes qui sont des **textes numériques**
- Un texte numérique peut être en notation scientifique ou hexadécimale, peut contenir un signe, et peut contenir des espaces blancs avant et après le nombre, p.ex. **"-1.5e2"**
- Par convention l'opérateur **+** unaire est utilisé pour faire la conversion des textes numériques

Textes numériques

```
> var a = "10"  
> var b = "-20"  
> a-1  
9  
> 2*a  
20  
> b  
"-20"  
> +b  
-20  
> a*b  
-200  
> (+a) * (+b)  
-200  
> +"allo"  
NaN
```


Opérateur + binaire

- L'opérateur + binaire est **surchargé** car différentes combinaisons de types de données provoquent des opérations différentes
 - Si les deux opérandes sont des nombres : **addition** numérique des opérandes
 - Si les deux opérandes sont des textes : **concaténation** des opérandes (raboutage des textes)
 - Si une des opérandes est un texte et l'autre est un nombre, le nombre est **converti** en texte numérique avant de faire la **concaténation**

Opérateur +

> "bon"

"bon"

> 11+22

33

> "bon"+"jour"

"bonjour"

> 11+22+"bon"

"33bon"

> "bon" + (11+22)

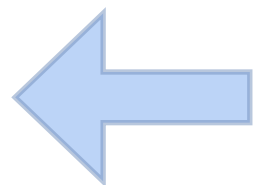
"bon33"

> "bon"+11+22

"bon1122"

> ""+33

"33"



concaténation et addition sont
sur le même niveau de préséance