

String

Properties

constructor

length

prototype

Methods & returned type

String.fromCharCode(num1 [, ...[, numN]])

String.fromCharCodePoint(num1 [, ...[, numN]])

String.raw()

String.charAt(index) *new string*

String.charCodeAt(index) *number*

String.codePointAt(position) *number*

String.concat(string1, string2, ..., stringN) *new string*

String.includes(searchTerm, position); *boolean*

String.indexOf(searchValue, fromIndex) *index*

String.lastIndexOf(searchValue[, fromIndex]) *index*

String.search(regex) *string*

String.split(regex) *array*

String.splitAll(regex) *array*

String.padStart(targetLength[, padString]) *string*

String.padEnd(targetLength[, padString]) *string*

String.repeat(count) *string*

String.trim() *string*

String.trimStart() *string*

String.trimEnd() *string*

String.replace(searchFor, replaceWith) *new string*

String.replaceAll(searchFor, replaceWith) *new string*

String.slice(beginIndex[, endIndex]) *new string*

String.substring(indexStart[, indexEnd]) *string*

String.split(separator[, limit]) *array*

String.startsWith(searchStr[, position]); *boolean*

String.endsWith(searchStr[, position]); *boolean*

String.toLowerCase() *string*

String.toUpperCase() *string*

String.localeLowerCase() *string*

String.localeUpperCase() *string*



String (cont)

<code>str.valueOf()</code>	<i>primitive</i>
<code>str.localeCompare(strToCompare[, locales[, options]])</code>	<i>1 0</i>
<code>str.normalize(form);</code>	<i>string</i>
<code>toString</code>	<i>conversion</i>

7 booleans {false}

<code>boolean(0);</code>	<i>false</i>
<code>Boolean(-0)</code>	<i>false</i>
<code>(Boolean(null))</code>	<i>false</i>
<code>Boolean(false)</code>	<i>false</i>
<code>Boolean(NaN)</code>	<i>false</i>
<code>Boolean(undefined)</code>	<i>false</i>
<code>Boolean("")</code>	<i>false</i>

Array

Properties

`length`
`constructor`
`prototype`

Methods & returned value

<code>Array.from(arrayLike[, functionMap[, thisArg]])</code>	<i>new array</i>
<code>Array.isArray(value)</code>	<i>boolean</i>
<code>Array.of(item0[, item1[, ...[, itemN]])</code>	<i>new array</i>
<code>concat(value0, value1, ... , valueN)</code>	<i>new array</i>
<code>push(item0, item1, / ... ,/ itemN)</code>	<i>number</i>
<code>pop()</code>	<i>string</i>
<code>shift()</code>	<i>number</i>
<code>unshift(item0, item1, / ... ,/ itemN)</code>	<i>number</i>
<code>reverse()</code>	<i>array</i>
<code>slice(start, end)</code>	<i>array</i>
<code>splice(start, delete Count, item1, item2, itemN)</code>	<i>array</i>
<code>join(separator)</code>	<i>string</i>
<code>includes(searchElement, fromIndex)</code>	<i>boolean</i>
<code>find((element, index, array) => { ... })</code>	<i>index</i>
<code>findIndex((item, index, array) => { ... })</code>	<i>index</i>



Array (cont)

<code>indexOf(s ear chE lement, fromIndex)</code>	<i>index</i>
<code>lastIn dex Of(sea rch Ele ment, fromIndex)</code>	<i>index</i>
<code>copyWi thi n(t arget, start, end)</code>	<i>array</i>
<code>fill(v alue, start, end)</code>	<i>array</i>
<code>entries()</code>	<i>new array</i>
<code>keys()</code>	<i>new array</i>
<code>values()</code>	<i>new array</i>
<code>forEac h(c all back, thisArg)</code>	<i>string</i>
<code>map(ca llback [, thisArg])</code>	<i>new array</i>
<code>filter (ca llback, thisArg);</code>	<i>new array</i>
<code>reduce (ca llb ackFn, initia lValue)</code>	<i>value</i>
<code>reduce Rig ht(cal lba ckFn, initia lValue)</code>	<i>value</i>
<code>some(c all back[, objetT his])</code>	<i>boolean</i>
<code>every(cal lback[, thisArg])</code>	<i>boolean</i>
<code>sort(f onc tio nCo mpa raison)</code>	<i>object</i>
<code>flat(d epth)</code>	<i>new array</i>
<code>flatMa p(f unc tion)</code>	<i>new array</i>
<code>toString()</code>	<i>string</i>

Opérateur

Opérateurs logiques

`&& AND, || OR, ! NOT`

Opérateurs de compar aison

`==, ===, !=, !==, >, >, =, <, <=`

Opérateurs arithm étiques

`+, -, , /, %, *`

Opérateurs d'affe ctation

`+=, -=, =, /=, %/, *`

Incrém ent ation Décrém ent ation

`i++, ++i, i--, --i`

Opérateur ternaire

`condition ? val1(true) : val2(f alse);`

Opérateurs unaires

`delete, typeof, void, +, -, ~, !`

Opérateurs relati onnels

`in, instanceof`



Number

Properties

Number.MIN_VALUE	Number.MAX_VALUE
Number.MIN_SAFE_INTEGER	Number.MAX_SAFE_INTEGER
Number.NaN	EPSILON
NEGATIVE_INFINITY	POSITIVE_INFINITY

Methods & returned type

Number.isNaN()	<i>boolean</i>
Number.isFinite()	<i>boolean</i>
Number.isInteger()	<i>boolean</i>
Number.isSafeInteger()	<i>boolean</i>
Number.parseFloat(string)	<i>number</i>
Number.parseInt(string, [base])	<i>number</i>
toExponential(fractionDigits)	<i>string</i>
toFixed(digits)	<i>number</i>
toPrecision(precision)	<i>string</i>
toString([radix])	<i>string</i>
valueOf()	<i>number</i>

Math object

Math.round()	Arrondi à l'entier le plus proche
Math.floor()	Arrondi à l'entier inférieur
Math.ceil()	Arrondi à l'entier supérieur
Math.min()	Renvoie le plus petit nombre
Math.max()	Renvoie le plus grand nombre
Math.sign()	Renvoie le signe d'un nombre
Math.abs()	Renvoie la valeur absolue
Math.sqrt()	Renvoie la racine carrée
Math.pow(x, y)	Renvoie la puissance
Math.log	Renvoie le logarithme
Math.random	Renvoie un nombre aléatoire

Regexp

Pattern Modifiers

<i>i</i> insensible à la case	<i>g</i> match global
<i>m</i> multiligne matching	

Brackets



Regexp (cont)

<code>[abc]</code> ou <code>[a-c]</code>	Correspond à n'importe quels caractères entre les crochets
<code>[^abc]</code> ou <code>[^a-c]</code>	Correspond à tout ce qui n'est pas indiqué entre les crochets.
<code>[a-zA-Z-0-9_]</code>	Tout caractères qui n'est pas une lettre ou un chiffre
<code>[0-9]</code>	
<code>[A-z]</code>	
<code>(a b c)</code>	
<code>x y</code>	Correspond à 'x' ou 'y'

Character classes

<code>\w</code> word	<code>\W</code> NOT word
<code>\d</code> digit	<code>\D</code> NOT digit
<code>\s</code> whitespace	<code>\S</code> NOT whitespace
<code>\t</code> tabs	<code>\n</code> line breaks

Quantifiers

<code>z?</code> 0 ou 1 occurrence	<code>z*</code> 0 ou multiple occurrences
<code>z+</code> 1 ou multiple occurrences	<code>z{n}</code> n occurrences
<code>z{min,max}</code> min/max occurrences	

anchors

<code>^hello</code> start of the strings	
<code>hello\$</code> end of the string	
<code>\b</code> limite de mot	<code>\B</code> pas limite de mot
<code>x(=y)</code> 'x' s'il est suivi de 'y'	
<code>x(? y)</code> 'x' s'il n'est pas suivi de 'y'	
<code>(?<=y)x</code> 'x' s'il est précédé par 'y'	
<code>(?<!y)x</code> 'x' s'il n'est pas précédé par 'y'	

String regexp function

`match()` `search()` `replace()` `split()`

regexp methods

`test()` `exec()`



Statement

if ... else

```
if (condi tion) {  
    statem ent_1;  
} else {  
    statem ent_2;  
}
```

switch

```
switch (expre ssion) {  
    case label_1:  
        statem ent_1  
        [break;]    case label_2:  
            statem ent_2  
            [break;]  
    ...  
    default:  
        defaul t_s tat ement  
        [break;]  
}
```

Loops

while

```
while (condi tion){  
    //code to run  
    final- exp ression  
};
```

do ... while

```
do {  
    //code to run  
    fin al- exp ression  
} while (exit- con dit ion);
```

for

```
for ([Init iale]; [condi tion]; [Incrémt]) {  
    //code to run  
};
```

label

```
label :  
    instru ction
```

break [label]

```
for (i = 0; i < a.length; i++) {  
    if (a[i] === valeur Test) {  
        break;  
    }  
}
```

continue [label];

```
while (i < 5) {  
    i++;  
    if (i === 3) {  
        continue;  
    }    n += i;  
}
```

for ... in

```
for (variable in objet) {  
    //code to run  
};
```

for ... of

```
for (variable of objet) {  
    //code to run  
};
```

