

String

Properties

constructor

length

prototype

Methods & returned type

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

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

String.raw()

str.charAt(index) *new string*

str.charCodeAtAt(index) *number*

str.codePointAt(position) *number*

str.concat(string2[, string3, ..., stringN]) *new string*

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

str.indexOf(searchValue, fromIndex) *index*

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

str.search(regex) *string*

str.match(regex) *array*

str.matchAll(regex) *array*

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

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

str.repeat(count) *string*

str.trim() *string*

str.trimStart() *string*

str.trimEnd() *string*

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

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

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

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

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

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

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

str.toLowerCase() *string*

str.toUpperCase() *string*

str.toLocaleLowerCase() *string*

str.toLocaleUpperCase() *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, deleteCount, 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(searchElement, fromIndex)</code>	<i>index</i>
<code>lastIndexOf(searchElement, fromIndex)</code>	<i>index</i>
<code>copyWithin(target, start, end)</code>	<i>array</i>
<code>fill(value, 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>forEach(callback, thisArg)</code>	<i>string</i>
<code>map(callback[, thisArg])</code>	<i>new array</i>
<code>filter(callback, thisArg);</code>	<i>new array</i>
<code>reduce(callbackFn, initialValue)</code>	<i>value</i>
<code>reduceRight(callbackFn, initialValue)</code>	<i>value</i>
<code>some(callback[, objectThis])</code>	<i>boolean</i>
<code>every(callback[, thisArg])</code>	<i>boolean</i>
<code>sort(compareFunction)</code>	<i>object</i>
<code>flatMap()</code>	<i>new array</i>
<code>flatMap(function)</code>	<i>new array</i>
<code>toString()</code>	<i>string</i>

Opérateur

Opérateurs logiques

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

Opérateurs de comparaison

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

Opérateurs arithmétiques

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

Opérateurs d'affectation

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

Incrément/ Décrément

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

Opérateur ternaire

`condition ? val1(true) : val2(false);`

Opérateurs unaires

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

Opérateurs relationnels

`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 le valeur absolue
Math.sqrt()	Renvoie le racine carrée
Math.pow(x, y)	Renvoie le puissance
Math.log	Renvoie le logarithme
Math.random	Renvoie un nombre aléatoire

Regex

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 (condition){
    //code to run
    final- expression
};
```

do ... while

```
do {
    //code to run
    final- expression
} while (exit- condition);
```

for

```
for ([Initiale]; [condition]; [Incrément]) {
    //code to run
};
```

label

```
label :
    instruction
```

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
};
```

