String

Properties	
constr uctor	
lenght	
prototype	
Methods & returned type	
String.fr omC har Cod e(num1 [, ...[,n umN]])	
String.fr omC ode Poi nt(num1 [, ...[,n umN]])	
String.raw()	
str.ch arA t(i ndex)	*new string*
str.ch arC ode At(index)	*number*
str.co deP oin tAt (po sition)	*number*
str.co nca t(s tri ng2[, string3, ..., stringN])	*new string*
str.in clu des (se arc hTerm, position);	*boolean*
str.in dex Of(sea rch Value, fromIndex)	*index*
str.la stI nde xOf (se arc hVa lue[, fromIn dex])	*index*
str.se arc h(r egexp)	*string*
str.ma tch (re gexp)	*array*
str.ma tch All (re gexp)	*array*
str.pa dSt art (ta rge tLenght [, padStr ing])	*string*
str.pa dEn d(t arg etLengh t[, padStr ing])	*string*
str.re pea t(c ount)	*string*
str.trim()	*string*
str.tr imS tart()	*string*
str.tr imEnd()	*string*
str.re pla ce(sea rchFor, replac eWith)	*new string*
str.re pla ceA ll(sea rchFor, replac eWith)	*new string*
str.sl ice (be gin Index[, endIndex])	*new string*
str.su bst rin g(i nde xSt art[, indexEnd])	*string*
str.sp lit ([s epa rator[, limit]])	*array*
str.st art sWi th(sea rchStr [, positi on]);	*boolean*
str.en dsW ith (se arc hStr[, positi on]);	*boolean*
str.to Low erC ase()	*string*
str.to Upp erC ase()	*string*
str.to Loc ale Low erC ase()	*string*
str.to Loc ale Upp erC ase()	*string*

String (cont)

str.va lueOf()	*primitive*
str.lo cal eCo mpa re(str ToC omp are [,l oca les[, options]])	*1 || 0*
str.no rma liz e([form]);	*string*
toString	convertion

7 booleans {false}

boolea n(0);	*false*
Boolea n(-0)	*false*
(Boole an(null)	*false*
Boolea n(f alse)	*false*
Boolea n(NaN)	*false*
Boolea n(u nde fined)	*false*
Boolea n("")	*false*

Array

Properties	
lenght	
constr uctor	
prototype	
Methods & returned value	
Array.from(arrayLike [, foncti onMap[, thisArg]])	*new array*
Array.i sA rra y(v alue)	*boolean*
Array.o f(item0[, item1[, ...[, itemN]]])	*new array*
concat (va lue0, value1, ... , valueN)	*new array*
push(i tem0, item1, / ... ,/ itemN)	*number*
pop()	*string*
shift()	*number*
unshif t(i tem0, item1, / ... ,/ itemN)	*number*
reverse()	*array*
slice(strat, end)	*array*
splice (start, delete Count, item1, item2, itemN)	*array*
join(s epa rator)	*string*
includ es(sea rch Ele ment, fromIndex)	*boolean*
find((ele ment, index, array) => { ... })	*index*
findIn dex ((item, index, array) => { ... })	*index*

Array (cont)

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

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.MI N_VALUE	Number.MA X_VALUE
Number.MI N_S AFE _IN TERGER	Number.MA X_S AFE _IN TERGER
Number.NaN	EPSILON
NEGATI VE_ INF INITY	POSITI VE_ INF INITY
Methods & returned type	
Number.is NaN()	*boolean*
Number.is Fin ite()	*boolean*
Number.is Int eger()	*boolean*
Number.is Saf eIn teger()	*boolean*
Number.pa rse Flo at(string)	*number*
Number.pa rse Int (st ring, [base])	*number*
toExpo nen tia l(f rac tio nDi gits)	*string*
toFixe d(d igits)	*number*
toPrec isi on(pre cision)	*string*
toStri ng([ra dix])	*string*
valueOf()	*number*

Math object

Math.r ound()	Arrondi à l'entier le plus proche
Math.f loor()	Arrondi à l'entier inférieur
Math.c eil()	Arrondi à l'entier supérieur
Math.min()	Renvoie le plus petit nombre
Math.max()	Renvoie le plus grand nombre
Math.s ign()	Renvoie le signe d'un nombre
Math.abs()	Renvoie le valeur absolue
Math.s qrt()	Renvoie le racine carrée
Math.p ow(x, y)	Renvoie le puissance
Math.log	Renvoie le logarithme
Math.r andom	Renvoie un nombre aléatoire

Regexp

Pattern Modifiers	
i *insensible à la case*	g *match global*
m *multiligne matching*	
Brackets	

Regexp (cont)

[abc] ou [a-c]	*Correspond à n'importe quels caractères entre les crochets*
[^abc] ou [^a-c]	*Correspond à tout ce qui n'est pas indiqué entre les crochets.*
[a-ZA- Z-0-9_]	*Tout caractères qui n'est pas une lettre ou un chiffre*
[0-9]	
[A-z]	
(a|b|c)	
x|y	*Correspond à 'x' ou 'y'*
Character classes	
\w *word*	\W *NOT word*
\d *digit*	\D *NOT digit*
\s *whitespace*	\S *NOT whitespace*
\t *tabs*	\n *line breaks*
Quanti fiers	
z? *0 ou 1 occurrence*	z 0 ou multiple occurr ences*
z+ *1 ou multiple occurr ences*	z{n} *n occurr ences*
z{min,max} min/max occurr ences	
Anchors	
^hello *start of the strings*	
hello$ *end of the string*	
\b *limite de mot*	\B *pas limite de mot*
x(?=y) *'x' s'il est suivi de 'y'*	
x(?|y|) *'x' s'il n'est pas suivi de 'y'*	
(?<=y)x *'x' s'il est précédé par 'y'*	
(?<!y)x *x' s'il n'est pas précédé par 'y'*	
String regexp function	
match() search() replace() split()	
regexp methods	
test() exec()	

Cheatography

Javascript Base Keyboard Shortcuts

by Exalys (Exalys) via cheatography.com/143213/cs/30776/

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
 };
```