

ES6 Library Extensions

Contents

Math Extensions	1
Number extensions.....	3
String extensions	4

Math Extensions

15.8.2.18 log10 (x)

Returns an implementation-dependent approximation to the base 10 logarithm of x .

- If x is NaN, the result is NaN.
- If x is less than 0, the result is NaN.
- If x is +0 or -0, the result is $-\infty$.
- If x is 1, the result is +0.
- If x is $+\infty$, the result is $+\infty$.

15.8.2.19 log2 (x)

Returns an implementation-dependent approximation to the base 2 logarithm of x .

- If x is NaN, the result is NaN.
- If x is less than 0, the result is NaN.
- If x is +0 or -0, the result is $-\infty$.
- If x is 1, the result is +0.
- If x is $+\infty$, the result is $+\infty$.

15.8.2.20 log1p (x)

Returns an implementation-dependent approximation to the natural logarithm of $1 + x$. The result is computed in a way that is accurate even when the value of x is close to zero.

- If x is NaN, the result is NaN.
- If x is less than -1, the result is NaN.
- If x is -1, the result is $-\infty$.
- If x is +0 or -0, the result is x .
- If x is $+\infty$, the result is $+\infty$.

15.8.2.21 expm1 (x)

Returns an implementation-dependent approximation to subtracting 1 from the exponential function of x (e raised to the power of x , where e is the base of the natural logarithms). The result is computed in a way that is accurate even when the value of x is close 0.

- If x is NaN, the result is NaN.
- If x is +0, the result is +0.
- If x is -0, the result is -0.
- If x is $+\infty$, the result is $+\infty$.
- If x is $-\infty$, the result is +1.

15.8.2.22 cosh(x)

Returns an implementation-dependent approximation to the hyperbolic cosine of x .

- If x is NaN, the result is NaN.
- If x is $+0$ or -0 , the result is 1.
- If x is $+\infty$ or $-\infty$, the result is $+\infty$.

Note: The value of $\cosh(x)$ is the same as $(\exp(x) + \exp(-x))/2$.

15.8.2.23 sinh(x)

Returns an implementation-dependent approximation to the hyperbolic sine of x .

- If x is NaN, the result is NaN.
- If x is $+0$ or -0 , the result is x .
- If x is $+\infty$, the result is $+\infty$.
- If x is $-\infty$, the result is $-\infty$.

Note: The value of $\cosh(x)$ is the same as $(\exp(x) - \exp(-x))/2$.

15.8.2.24 tanh(x)

Returns an implementation-dependent approximation to the hyperbolic tangent of x .

- If x is NaN, the result is NaN.
- If x is $+0$ or -0 , the result is x .
- If x is $+\infty$, the result is $+1$.
- If x is $-\infty$, the result is -1 .

Note: The value of $\tanh(x)$ is the same as $(\exp(x) - \exp(-x))/(\exp(x) + \exp(-x))$.

15.8.2.25 acosh(x)

Returns an implementation-dependent approximation to the inverse hyperbolic cosine of x .

- If x is NaN, the result is NaN.
- If x is less than 1, the result is NaN.
- If x is 1, the result is $+0$.
- If x is $+\infty$, the result is $+\infty$.

15.8.2.26 asinh(x)

Returns an implementation-dependent approximation to the inverse hyperbolic sine of x .

- If x is NaN, the result is NaN.
- If x is $+0$ or -0 , the result is x .
- If x is $+\infty$, the result is $+\infty$.
- If x is $-\infty$, the result is $-\infty$.

15.8.2.27 atanh(x)

Returns an implementation-dependent approximation to the inverse hyperbolic tangent of x .

- If x is NaN, the result is NaN.
- If x is less than -1 , the result is NaN.
- If x is greater than 1, the result is NaN.
- If x is -1 , the result is $-\infty$.
- If x is $+1$, the result is $+\infty$.
- If x is $+0$ or -0 , the result is x .

15.8.2.28 hypot(x, y)

Returns an implementation-dependent approximation to the hypotenuse of a right triangle with sides of length x and y .

- If x is NaN, the result is NaN.

- If y is NaN, the result is NaN.
- If x is $+0$ and y is -0 , the result is $+0$.
- If x is $+0$ and y is not -0 , the result is y .
- If x is -0 and y is -0 , the result is $+0$.
- If x is -0 and y is not -0 , the result is y .
- If x is $+\infty$, the result is $+\infty$.
- If x is $-\infty$, the result is $+\infty$.
- If y is $+\infty$, the result is $+\infty$.
- If y is $-\infty$, the result is $+\infty$.

15.8.2.29 trunc(x)

Returns the integral part of the number x , removing any fractional digits. If x is already an integer, the result is x .

- If x is NaN, the result is NaN.
- If x is -0 , the result is -0 .
- If x is $+0$, the result is $+0$.
- If x is $+\infty$, the result is $+\infty$.
- If x is $-\infty$, the result is $-\infty$.

15.8.2.30 sign(x)

Returns the sign of the x , indicating whether x is positive, negative or zero.

- If x is NaN, the result is NaN.
- If x is -0 , the result is -0 .
- If x is $+0$, the result is $+0$.
- If x is negative, the result is -1 .
- If x is positive, the result is $+1$.

Number extensions

15.7.3.7 isNaN (number)

Returns **true** if the argument is a Number value equal to **NaN**, and otherwise returns **false**.

1. If `Type(number)` is not Number, return **false**.
2. If *number* is **NaN**, return **true**.
3. Otherwise, return **false**.

NOTE A reliable way for ECMAScript code to test if a value x is a **NaN** is an expression of the form $x \text{ !== } x$. The result will be **true** if and only if x is a **NaN**.

15.7.3.8 isFinite (number)

Returns **false** if the argument is a Number value equal to **NaN**, $+\infty$, or $-\infty$, and otherwise returns **true**.

1. If `Type(number)` is not Number, return **false**.
2. If *number* is **NaN**, $+\infty$, or $-\infty$, return **false**.
3. Otherwise, return **true**.

15.7.3.8 isInteger (number)

Returns **true** if the argument is a Number value which is an integral value.

1. If `Type(number)` is not Number, return **false**.
2. Let *integer* be `ToInteger(number)`.
3. If *integer* is not equal to *number*, return **false**.
4. Otherwise, return **true**.

15.7.3.9 toInteger (number)

Returns the result of coercing the argument to an integer.

1. Return `ToInteger(number)`.

String extensions

15.5.4.21 `String.prototype.repeat` (count)

Returns a String consisting of the characters of this object (converted to String) repeated *count* time. The following steps are taken:

1. Call `CheckObjectCoercible` passing the **this** value as its argument.
2. Let *S* be the result of calling `ToString`, giving it the **this** value as its argument.
3. Let *n* be the result of calling `ToInteger(count)`.
4. Let *T* be a String value that is made from *n* copies of *S* appended together.
5. Return *T*.

15.5.4.22 `String.prototype.startsWith` (searchString, position)

Comment [LH1]: Questionable

Returns **true** if the sequence of characters of *searchString* converted to a String match the corresponding characters of this object (converted to a String) starting at *position*. Otherwise returns **false**. The following steps are taken:

1. Call `CheckObjectCoercible` passing the **this** value as its argument.
2. Let *S* be the result of calling `ToString`, giving it the **this** value as its argument.
3. Let *searchStr* be `ToString(searchString)`.
4. Let *pos* be `ToInteger(position)`. (If *position* is **undefined**, this step produces the value **0**).
5. Let *len* be the number of characters in *S*.
6. Let *start* be `min(max(pos, 0), len)`.
7. Let *searchLength* be the number of characters in *searchString*.
8. If *searchLength*+*start* is greater than *len*, return **false**.
9. If the first *searchLength* characters of *S* starting at *start* are equal to the first *searchLength* characters of *searchString*, return **true**.
10. Otherwise, return **false**.

The **length** property of the **startsWith** method is **1**.

NOTE The **startsWith** function is intentionally generic; it does not require that its **this** value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

15.5.4.23 `String.prototype.endsWith` (searchString, endPosition)

Comment [LH2]: Questionable

Returns **true** if the sequence of characters of *searchString* converted to a String match the corresponding characters of this object (converted to a String) starting at *endPosition* – `length(this)`. Otherwise returns **false**. The following steps are taken:

1. Call `CheckObjectCoercible` passing the **this** value as its argument.
2. Let *S* be the result of calling `ToString`, giving it the **this** value as its argument.
3. Let *searchStr* be `ToString(searchString)`.
4. Let *len* be the number of characters in *S*.
5. If *endPosition* is **undefined**, let *pos* be *len*, else let *pos* be `ToInteger(endPosition)`.
6. Let *end* be `min(max(pos, 0), len)`.
7. Let *searchLength* be the number of characters in *searchString*.
8. Let *start* be *end* - *searchLength*.
9. If *start* is less than 0, return **false**.
10. If the first *searchLength* characters of *S* starting at *start* are equal to the first *searchLength* characters of *searchString*, return **true**.
11. Otherwise, return **false**.

The **length** property of the **endsWith** method is **1**.

NOTE The **endsWith** function is intentionally generic; it does not require that its **this** value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

15.5.4.24 String.prototype.contains (searchString, position)

If *searchString* appears as a substring of the result of converting this object to a String, at one or more positions that are greater than or equal to *position*, then return **true**; otherwise, returns **false**. If *position* is **undefined**, 0 is assumed, so as to search all of the String.

The **contains** method takes two arguments, *searchString* and *position*, and performs the following steps:

1. Call CheckObjectCoercible passing the **this** value as its argument.
2. Let *S* be the result of calling ToString, giving it the **this** value as its argument.
3. Let *searchStr* be ToString(*searchString*).
4. Let *pos* be ToInteger(*position*). (If *position* is **undefined**, this step produces the value 0).
5. Let *len* be the number of characters in *S*.
6. Let *start* be min(max(*pos*, 0), *len*).
7. Let *searchLen* be the number of characters in *searchStr*.
8. If there exists any integer *k* not smaller than *start* such that *k* + *searchLen* is not greater than *len*, and for all nonnegative integers *j* less than *searchLen*, the character at position *k*+*j* of *S* is the same as the character at position *j* of *searchStr*, return **true**; but if there is no such integer *k*, return **false**.

The **length** property of the **contains** method is 1.

NOTE The **contains** function is intentionally generic; it does not require that its **this** value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

15.5.4.25 String.prototype.toArray()

Returns an Array object with elements corresponding to the characters of this object (converted to a String).

The following steps are taken:

1. Call CheckObjectCoercible passing the **this** value as its argument.
2. Let *S* be the result of calling ToString, giving it the **this** value as its argument.
3. Let *array* be a new array created as if by the expression **new Array()** where **Array** is the standard built-in constructor with that name.
4. Let *len* be the number of characters in *S*.
5. Let *n* be 0
6. Repeat, while *n* < *len*:
 - a. Let *c* be the character at position *n* in *S*.
 - b. Call the [[DefineOwnProperty]] internal method of *array* with arguments ToString(*n*), the PropertyDescriptor {[[Value]]: *c*, [[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **false**}, and **false**.
 - c. Increment *n* by 1.
7. Return *array*.

The **length** property of the **toArray** method is 0.

NOTE The **toArray** function is intentionally generic; it does not require that its **this** value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

15.5.4.26 String.prototype.reverse()

Returns an Array object with elements corresponding to the characters of this object (converted to a String) in reverse order.

The following steps are taken:

8. Call CheckObjectCoercible passing the **this** value as its argument.
9. Let *S* be the result of calling ToString, giving it the **this** value as its argument.
10. Let *len* be the number of characters in *S*.
11. Return a new String value of length *len* whose character at position *i* is the value of the character at position *len*-*i* in *S*.

The **length** property of the **reverse** method is 0.

NOTE The **reverse** function is intentionally generic; it does not require that its **this** value be a String object. Therefore, it can be transferred to other kinds of objects for use as a method.

