15.7.4.2 Number.prototype.toFixed(fractionDigits)

Return a string containing the number represented in fixed-point notation with *fractionDigits* digits after the decimal point. Specifically, perform the following steps:

1. Let f be ToInteger(*fractionDigits*).
2. If $f < 0$ or $f > 20$, return an implementation-defined result.
3. Let x be this number value.
4. If x is **NaN**, return the string **`"NaN"`**.
5. Let s be the empty string.
6. If x 0, go to step 9.
7. Let s be **`"-"`**.
8. Let $x = -x$.
9. If x 10^{21}, let m = ToString(x) and go to step 20.
10. Let n be an integer for which the exact mathematical value of $n/10^f - x$ is as close to zero as possible. If there are two such n, pick the larger n.
11. If $n = 0$, let m be the string **`"0"`**. Otherwise, let m be the string consisting of the digits of the decimal representation of n (in order, with no leading zeroes).
12. If $f = 0$, go to step 20.
13. Let k be the number of characters in m.
14. If $k > f$, go to step 18.
15. Let z be the string consisting of $f+1-k$ occurrences of the character '**`0`**'.
16. Let m be the concatenation of strings z and m.
17. Let $k = f+1$.
18. Let a be the first $k-f$ characters of m, and let b be the remaining f characters of m.
19. Let m be the concatenation of the three strings a, **`"."`**, and b.
20. Return the concatenation of the strings s and m.

If the **`toFixed`** method is called with more than one argument, the behavior is implementation-defined.

Note that the output of **`toFixed`** may be more precise than **`toString`** for some values because **`toString`** only prints enough significant digits to distinguish the number from adjacent Number values. For example, calling **`toString()`** on 1000000000000000128 returns **`"1000000000000000100"`**, while calling **`toFixed(0)`** on it returns **`"1000000000000000128"`**.

15.7.4.3 Number.prototype.toExponential(precision)

Return a string containing the number represented in exponential notation with *precision* digits after the mantissa's decimal point. If *precision* is missing or **undefined**, include as many mantissa digits as necessary to uniquely specify the number (just like in ToString except that in this case the number is always output in exponential notation). Specifically, perform the following steps:

1. Let x be this number value.
2. Let p be ToInteger(*precision*).
3. If x is **NaN**, return the string **`"NaN"`**.
4. Let s be the empty string.
5. If x 0, go to step 8.
6. Let s be **`"-"`**.
7. Let $x = -x$.
8. If $x = +$, let m = **`"Infinity"`** and go to step 30.
9. If *precision* is missing or **undefined**, go to step 14.
10. If $p < 0$ or $p > 20$, return an implementation-defined result.
11. If $x = 0$, go to step 16.

12. Let *e* and *n* be integers such that 10^p $n < 10^{p+1}$ and for which the exact mathematical value of $n \cdot 10^{e-p} - x$ is as close to zero as possible. If there are two such sets of *e* and *n*, pick the *e* and *n* for which $n \cdot 10^{e-p}$ is larger.
13. Go to step 20.
14. If *x*   0, go to step 19.
15. Let $p = 0$.
16. Let *m* be the string consisting of *p*+1 occurrences of the character ‘**`0`**’.
17. Let $e = 0$.
18. Go to step 21.
19. Let *e*, *n*, and *p* be integers such that *p*   0, 10^p $n < 10^{p+1}$, the number value for $n \cdot 10^{e-p}$ is *x*, and *p* is as small as possible. Note that the decimal representation of *n* has *p*+1 digits, *n* is not divisible by 10, and the least significant digit of *n* is not necessarily uniquely determined by these criteria.
20. Let *m* be the string consisting of the digits of the decimal representation of *n* (in order, with no leading zeroes).
21. If $p = 0$, go to step 24.
22. Let *a* be the first character of *m*, and let *b* be the remaining *p* characters of *m*.
23. Let *m* be the concatenation of the three strings *a*, **`"."`**, and *b*.
24. If e = 0, let *c* = **`"+"`** and *d* = **`"0"`** and go to step 29.
25. If e > 0, let *c* = **`"+"`** and go to step 28.
26. Let *c* = **`"-"`**.
27. Let $e = -e$.
28. Let *d* be the string consisting of the digits of the decimal representation of *e* (in order, with no leading zeroes).
29. Let *m* be the concatenation of the four strings *m*, **`"e"`**, *c*, and *d*.
30. Return the concatenation of the strings *s* and *m*.

If the **`toExponential`** method is called with more than one argument, the behavior is implementation-defined.

NOTE For implementations which provide more accurate conversions than required by the rules above, it is recommended that the following alternative version of step 19 be used as a guideline:

> Let *e*, *n*, and *p* be integers such that *p*   0, 10^p $n < 10^{p+1}$, the number value for $n \cdot 10^{e-p}$ is *x*, and *p* is as small as possible. If there are multiple possibilities for *n*, choose the value of *n* for which $n \cdot 10^{e-p}$ is closest in value to *x*. If there are two such possible values of *n*, choose the one that is even.

15.7.4.4 Number.prototype.toGeneral(precision)

Return a string containing the number represented either in exponential notation with *precision* digits after the mantissa's decimal point or in fixed notation with *precision* +1 significant digits. If *precision* is missing or **undefined**, use ToString (section 9.8.1) instead. Specifically, perform the following steps:

1. Let *x* be this number value.
2. If *precision* is missing or **undefined**, return ToString(*x*).
3. Let *p* be ToInteger(*precision*).
4. If *x* is **NaN**, return the string **`"NaN"`**.
5. Let *s* be the empty string.
6. If *x*   0, go to step 9.
7. Let *s* be **`"-"`**.
8. Let $x = -x$.
9. If *x* = +   , let *m* = **`"Infinity"`** and go to step 30.
10. If $p < 0$ or $p > 20$, return an implementation-defined result.
11. If *x*   0, go to step 15.

12. Let *m* be the string consisting of *p*+1 occurrences of the character ‘**`0`**’.
13. Let *e* = 0.
14. Go to step 18.
15. Let *e* and *n* be integers such that 10^p $n < 10^{p+1}$ and for which the exact mathematical value of $n \cdot 10^{e-p} - x$ is as close to zero as possible. If there are two such sets of *e* and *n*, pick the *e* and *n* for which $n \cdot 10^{e-p}$ is larger.
16. Let *m* be the string consisting of the digits of the decimal representation of *n* (in order, with no leading zeroes).
17. If *e* < -6 or *e* > 20, go to step 22.
18. If *e*   *p*, let *m* be the concatenation of *m* and *e*-*p* occurrences of the character ‘**`0`**’ and go to step 30.
19. If *e*   0, let *m* be the concatenation of the first *e*+1 characters of *m*, the character ‘**`.`**’, and the remaining *p*-*e* characters of *m* and go to step 30.
20. Let *m* be the concatenation of the string **`"0."`**, –(*e*+1) occurrences of the character ‘**`0`**’, and the string *m*.
21. Go to step 30.
22. Let *a* be the first character of *m*, and let *b* be the remaining *p* characters of *m*.
23. Let *m* be the concatenation of the three strings *a*, **`"."`**, and *b*.
24. If e = 0, let *c* = **`"+"`** and *d* = **`"0"`** and go to step 29.
25. If e > 0, let *c* = **`"+"`** and go to step 28.
26. Let *c* = **`"-"`**.
27. Let *e* = -*e*.
28. Let *d* be the string consisting of the digits of the decimal representation of *e* (in order, with no leading zeroes).
29. Let *m* be the concatenation of the four strings *m*, **`"e"`**, *c*, and *d*.
30. Return the concatenation of the strings *s* and *m*.

If the **`toGeneral`** method is called with more than one argument, the behavior is implementation-defined.