Package: friendlynumber (via r-universe)

October 15, 2026

Title Translate Numbers into Number Words

Version 1.0.0

Description Converts vectors of numbers into character vectors of numerals, including cardinals (one, two, three) and ordinals (first, second, third). Supports negative numbers, fractions, and arbitrary-precision integer and high-precision floating-point vectors provided by the 'bignum' package.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

URL https://github.com/EthanSansom/friendlynumber, https://ethansansom.github.io/friendlynumber/

BugReports https://github.com/EthanSansom/friendlynumber/issues

Suggests bignum, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Repository https://ethansansom.r-universe.dev

Date/Publication 2025-04-02 10:39:33 UTC

RemoteUrl https://github.com/ethansansom/friendlynumber

RemoteRef HEAD

RemoteSha 8788718211c3fad381b5fd28381979351dd45454

Contents

bigfloat_friendly 2
biginteger_friendly 5
format_number 6
friendlynumber_default_options 8
integerish_friendly 8
nth_friendly 10

ntimes_friendly 12
number_friendly 14
numeric_friendly 18
ordinal_friendly 21
quantifier_friendly 23

Index **26**

`bigfloat_friendly` *Translate a bigfloat to a cardinal character vector*

Description

Convert a <bignum_bigfloat> to a cardinal numeral (e.g. one tenth, one, two).

A `bignum::bigfloat()` can store numbers with up to 50 decimal digits of precision, which is useful for manipulating numbers which can't be accurately represented in a <numeric> vector.

`bigfloat_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `bigfloat_friendly()` does not perform input validation to maximize its speed.

Usage

```
bigfloat_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL
)

bigfloat_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
```

```
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL
)
```

Arguments

numbers	[bignum_bigfloat] A bignum::bigfloat() vector to translate.
zero	[character(1)] What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)] What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)] What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)] What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)] A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
decimal	[character(1)] A word inserted between the whole and fractional part of translated numbers. decimal is the string " and " by default.
and	[TRUE / FALSE] Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.
and_fractional	[TRUE / FALSE] Whether to insert an " and " before the smallest fractional tens place of translated numbers (e.g. "one hundred one thousandths" vs. "one hundred and one thousandths"). and_fractional is equal to and by default.
hyphenate_fractional	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 in the fractional part of translated numbers (e.g. "twenty-one hundredths" or "twenty one hundredths"). This also determines the hyphenation of the fractional units (e.g. "one ten-millionth" vs. "one ten millionth"). hyphenate_fractional is equal to hyphenate by default.
english_fractions	[character] A named character vector used as a dictionary for the translation of the fractional part of numbers. The names (i.e. keys) are the decimal digits of a fractional number and the values are the corresponding translations.

For example `english_fractions = c("5" = "a half")` matches the number `0.5` (translated as `"a half"`) and `2.5` (translated as `"two and a half"`).

By default `english_fractions` is a named character vector with translations for fractions `x / y` for `x = 1, 2, ..., 8` and `y = 1, 2, ..., 9`. For example, `2 / 3` is translated as `"two thirds"` and `1 / 2` is translated as `"one half"`.

Provide an empty character to `english_fractions` to opt out of any such translations. In this case `1 / 2` is translated as `"five tenths"` instead of `"one half"`.

Value

A non-NA character vector of the same length as `numbers`.

Examples

```
bigfloat_friendly(bignum::bigfloat(c(0.5, 0, 0.123, NA, NaN, Inf)))

# Specify the translations of "special" numbers
bigfloat_friendly(bignum::bigfloat(NaN), nan = "NAN")

# Modify the output formatting
big <- bignum::bigfloat(1234.5678)
bigfloat_friendly(big)
bigfloat_friendly(big, decimal = " point ")
bigfloat_friendly(big, hyphenate_fractional = FALSE)
bigfloat_friendly(big, and = TRUE, and_fractional = TRUE, decimal = " . ")

# The `friendlynumber.bigfloat.digits` option specifies the number of
# `<bignum_bigfloat>` digits mentioned by `bigfloat_friendly()`
opts <- options()
options(friendlynumber.bigfloat.digits = 5)
bigfloat_friendly(bignum::bigpi)

options(friendlynumber.bigfloat.digits = 10)
bigfloat_friendly(bignum::bigpi)
options(opts)

# Set `english_fractions` to specify the translation of certain
# fractions. The names (keys) of `english_fractions` should match
# the decimal part of a fraction (e.g. `"04"` matches `0.04`).
bigfloat_friendly(
  bignum::bigfloat(c(1/2, 0.04, 1.5, 10)),
  english_fractions = c(`5` = "1/2", `04` = "4/100")
)

# Input validation
try(bigfloat_friendly_safe(bignum::bigpi, and = NA))
```

`biginteger_friendly` *Translate a biginteger to a cardinal character vector*

Description

Convert a <bignum_biginteger> to a cardinal numeral (e.g. one, two, three).

A `bignum::biginteger()` can store any integer (i.e. arbitrary precision), which is useful for manipulating numbers too large to be represented (accurately) in an <integer> or <numeric> vector.

`biginteger_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `biginteger_friendly()` does not perform input validation to maximize its speed.

Usage

```
biginteger_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)

biginteger_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)
```

Arguments

`numbers`	`[bignum_biginteger]` A `bignum::biginteger()` vector to translate.
`zero`	`[character(1)]` What to call values of `0` in `numbers` (e.g. `zero = "zero"`).
`na`	`[character(1)]` What to call values of `NA` in `numbers` (e.g. `na = "missing"`).
`nan`	`[character(1)]` What to call values of `NaN` in `numbers` (e.g. `nan = "undefined"`).

inf	[character(1)]
	What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)]
	A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
and	[TRUE / FALSE]
	Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE]
	Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.

Value

A non-NA character vector of the same length as numbers.

Examples

```
biginteger_friendly(bignum::biginteger(c(0, 1, 2, NA, 10001)))

# Specify the translations of "special" numbers
biginteger_friendly(bignum::biginteger(-10), negative = "minus ")
biginteger_friendly(bignum::biginteger(NA), na = "unknown")

# Modify the output formatting
biginteger_friendly(bignum::biginteger(9999))
biginteger_friendly(bignum::biginteger(9999), and = TRUE)
biginteger_friendly(bignum::biginteger(9999), hyphenate = FALSE)

# Translate large numbers
large <- bignum::biginteger(10L)^1001L
biginteger_friendly(large)

# Input validation
try(biginteger_friendly_safe(1L))
```

---

format_number *Format a vector of numbers*

Description

Format a vector of numbers using format().

Usage

```
format_number(x, ...)

## S3 method for class 'integer'
format_number(x, bigmark = TRUE, ...)

## S3 method for class 'bignum_biginteger'
format_number(x, bigmark = TRUE, ...)

## S3 method for class 'numeric'
format_number(x, bigmark = TRUE, ...)

## S3 method for class 'bignum_bigfloat'
format_number(x, bigmark = TRUE, ...)

## Default S3 method:
format_number(x, ...)
```

Arguments

x	A vector of numbers to format. The friendlynumber package defines methods for integer, numeric, `bignum::biginteger()`, and `bignum::bigfloat()` numbers.
...	Additional arguments passed to or from other methods.
bigmark	`[TRUE / FALSE]` Whether the thousands places of formatted numbers should be separated with a comma (e.g. `"10,000,000"` vs. `"10000000"`). `bigmark` is `TRUE` by default.

Details

The number of decimal digits shown in the output of `format_number()` is controlled the `friendlynumber.numeric.digits` option for numeric vectors and `friendlynumber.bigfloat.digits` for `bignum::bigfloat()` vectors.

These options also control the number of decimal digits translated by `numeric_friendly()` and `bigfloat_friendly()` respectively. Because of this, `format_number()` is useful for verifying that the output of these `*_friendly()` functions is correct.

Value

A non-NA character vector of the same length as `x`.

Examples

```
format_number(c(1/3, 0, 0.999, NA, NaN, Inf, -Inf))
format_number(c(1L, 2L, 1001L))
format_number(1001L, bigmark = FALSE)

# Set `friendlynumber.numeric.digits` to control the decimal output
```

```
opts <- options()
options(friendlynumber.numeric.digits = 2)
format_number(1234.1234)
options(opts)

if (requireNamespace("bignum", quietly = TRUE)) {
  format_number(bignum::bigfloat(1234.1234))
  format_number(bignum::biginteger(2000000))

  # Set `friendlynumber.bigfloat.digits` to control the decimal output
  opts <- options()
  options(friendlynumber.bigfloat.digits = 3)
  format_number(bignum::bigfloat(1234.1234))
  options(opts)
}
```

`friendlynumber_default_options` *Get the default options set by the friendlynumber package*

Description

Returns a list of options provided to `options()` when the friendlynumber package is loaded. Options set prior to loading the friendlynumber package are not overwritten on load.

Usage

```
friendlynumber_default_options()
```

Value

A named list of options.

Examples

```
friendlynumber_default_options()
```

`integerish_friendly` *Translate integer-ish numbers to a cardinal character vector*

Description

Convert an integer vector, or numeric vector which is coercible to an integer without loss of precision, to a cardinal numeral (e.g. one, two, three).

`integerish_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `integerish_friendly()` does not perform input validation to maximize its speed.

Usage

```
integerish_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)

integerish_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)
```

Arguments

numbers	[integer / numeric]
	An integer or integer-ish numeric vector to translate.
zero	[character(1)]
	What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)]
	What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)]
	What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)]
	What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)]
	A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
and	[TRUE / FALSE]
	Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE]
	Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.

Value

A non-NA character vector of the same length as `numbers`.

Examples

```
integerish_friendly(c(0, 1, 2, NA, NaN, Inf, -Inf))
integerish_friendly(10^10)

# Specify the translations of "special" numbers
integerish_friendly(-10, negative = "minus ")
integerish_friendly(NaN, nan = "undefined")

# Modify the output formatting
integerish_friendly(1234)
integerish_friendly(1234, and = TRUE)
integerish_friendly(1234, hyphenate = FALSE)

# Input validation
try(integerish_friendly_safe(0.5))
try(integerish_friendly_safe(1L, na = TRUE))
```

---

nth_friendly *Translate integer-ish numbers to a character vector of nths (1st, 2nd, 3rd)*

Description

Convert an integer vector, or numeric vector which is coercible to an integer without loss of precision, to an "nth" (e.g. 1st, 2nd, 3rd, 22nd, 1,000th).

`nth_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `nth_friendly()` does not perform input validation to maximize its speed.

Usage

```
nth_friendly(
  numbers,
  zero = "0th",
  na = "missingth",
  nan = "not a numberth",
  inf = "infinitieth",
  negative = "negative ",
  bigmark = TRUE
)

nth_friendly_safe(
  numbers,
  zero = "zeroth",
  na = "missingth",
```

```
  nan = "not a numberth",
  inf = "infinitieth",
  negative = "negative ",
  bigmark = TRUE
)
```

Arguments

numbers	[integer / numeric]
	An integer or integer-ish numeric vector to translate.
zero	[character(1)]
	What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)]
	What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)]
	What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)]
	What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)]
	A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
bigmark	[TRUE / FALSE]
	Whether the thousands places of formatted numbers should be separated with a comma (e.g. "10,000,000" vs. "10000000"). bigmark is TRUE by default.

Value

A non-NA character vector of the same length as numbers.

Examples

```
nth_friendly(c(0, 1, 2, 3, 22, 1001, NA, NaN, Inf, -Inf))

# Specify the translations of "special" numbers
nth_friendly(c(1, 0, NA), zero = "noneth", na = "?")

# Use `bigmark` to add or remove commas
nth_friendly(1234, bigmark = TRUE)
nth_friendly(1234, bigmark = FALSE)

# Input validation
try(nth_friendly_safe(1234, bigmark = ","))
```

`ntimes_friendly` *Translate integer-ish numbers to a character vector of counts (once, twice, three times)*

Description

Convert an integer vector, or numeric vector which is coercible to an integer without loss of precision, to a count (e.g. no times, once, twice, four times).

`ntimes_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `ntimes_friendly()` does not perform input validation to maximize its speed.

Usage

```
ntimes_friendly(
  numbers,
  one = "once",
  two = "twice",
  three = "three times",
  zero = "no times",
  na = "an unknown number of times",
  nan = "an undefined number of times",
  inf = "infinite times",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)

ntimes_friendly_safe(
  numbers,
  one = "once",
  two = "twice",
  three = "three times",
  zero = "no times",
  na = "an unknown number of times",
  nan = "an undefined number of times",
  inf = "infinite times",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)
```

Arguments

`numbers` `[integer / numeric]`

An integer or integer-ish numeric vector to translate.

one	[character(1)] What to call values of 1 in numbers (e.g. one = "the").
two	[character(1)] What to call values of 2 in numbers (e.g. two = "both").
three	[character(1)] What to call values of 3 in numbers (e.g. three = "thrice").
zero	[character(1)] What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)] What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)] What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)] What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)] A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
and	[TRUE / FALSE] Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.

Value

A non-NA character vector of the same length as numbers.

Examples

```
ntimes_friendly(c(0, 1, 2, 3, 22, 1001, NA, NaN, Inf, -Inf))

# Specify the translations of "special" numbers
ntimes_friendly(c(3, NA), three = "thrice", na = "some times")

# Modify the output formatting
ntimes_friendly(5678)
ntimes_friendly(5678, and = TRUE)
ntimes_friendly(5678, hyphenate = FALSE)

# Input validation
try(ntimes_friendly_safe(1234, and = " - "))
```

number_friendly *Translate a vector of numbers to a cardinal character vector*

Description

Convert a vector of numbers to a cardinal numeral (e.g. one tenth, one, two).

`number_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `number_friendly()` does not perform input validation to maximize its speed.

Usage

```
number_friendly(numbers, ...)

## S3 method for class 'numeric'
number_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL,
  ...
)

## S3 method for class 'integer'
number_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  ...
)

## S3 method for class 'bignum_biginteger'
number_friendly(
```

```
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  ...
)

## S3 method for class 'bignum_bigfloat'
number_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL,
  ...
)

## Default S3 method:
number_friendly(numbers, ...)

number_friendly_safe(numbers, ...)

## S3 method for class 'numeric'
number_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL,
  ...
```

```
)

## S3 method for class 'integer'
number_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  ...
)

## S3 method for class 'bignum_biginteger'
number_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  ...
)

## S3 method for class 'bignum_bigfloat'
number_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL,
  ...
)

## Default S3 method:
number_friendly_safe(numbers, ...)
```

Arguments

numbers	A vector of numbers to translate. The friendlynumber package defines methods for integer, numeric, bignum::biginteger(), and bignum::bigfloat() numbers. • Integers are passed to integerish_friendly() • Numeric vectors are passed to numeric_friendly() • bignum::biginteger() vectors are passed to biginteger_friendly() • bignum::bigfloat() vectors are passed to bigfloat_friendly()
...	Additional arguments passed to or from other methods.
zero	[character(1)] What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)] What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)] What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)] What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)] A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
decimal	[character(1)] A word inserted between the whole and fractional part of translated numbers. decimal is the string " and " by default.
and	[TRUE / FALSE] Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.
and_fractional	[TRUE / FALSE] Whether to insert an " and " before the smallest fractional tens place of translated numbers (e.g. "one hundred one thousandths" vs. "one hundred and one thousandths"). and_fractional is equal to and by default.
hyphenate_fractional	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 in the fractional part of translated numbers (e.g. "twenty-one hundredths" or "twenty one hundredths"). This also determines the hyphenation of the fractional units (e.g. "one ten-millionth" vs. "one ten millionth"). hyphenate_fractional is equal to hyphenate by default.
english_fractions	[character]

A named character vector used as a dictionary for the translation of the fractional part of numbers. The names (i.e. keys) are the decimal digits of a fractional number and the values are the corresponding translations.

For example english_fractions = c("5" = "a half") matches the number 0.5 (translated as "a half") and 2.5 (translated as "two and a half").

By default english_fractions is a named character vector with translations for fractions x / y for x = 1, 2, ..., 8 and y = 1, 2, ..., 9. For example, 2 / 3 is translated as "two thirds" and 1 / 2 is translated as "one half".

Provide an empty character to english_fractions to opt out of any such translations. In this case 1 / 2 is translated as "five tenths" instead of "one half".

Value

A non-NA character vector of the same length as numbers.

See Also

integerish_friendly(), numeric_friendly(), biginteger_friendly(), bigfloat_friendly()

Examples

```
number_friendly(c(1/3, 0, 0.999, NA, NaN, Inf, -Inf))
number_friendly(c(1L, 2L, 1001L))

# Input validation
try(number_friendly_safe(1L, zero = c("a", "zero")))
```

---

numeric_friendly *Translate a numeric vector to a cardinal character vector*

Description

Convert a numeric vector to a cardinal numeral (e.g. one tenth, one, two).

numeric_friendly_safe() checks that all arguments are of the correct type and raises an informative error otherwise. numeric_friendly() does not perform input validation to maximize its speed.

Usage

```
numeric_friendly(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
```

```
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL
)

numeric_friendly_safe(
  numbers,
  zero = "zero",
  na = "missing",
  nan = "not a number",
  inf = "infinity",
  negative = "negative ",
  decimal = " and ",
  and = FALSE,
  hyphenate = TRUE,
  and_fractional = and,
  hyphenate_fractional = hyphenate,
  english_fractions = NULL
)
```

Arguments

numbers	[numeric] A numeric vector to translate.
zero	[character(1)] What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)] What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)] What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)] What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)] A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
decimal	[character(1)] A word inserted between the whole and fractional part of translated numbers. decimal is the string " and " by default.
and	[TRUE / FALSE] Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.

and_fractional [TRUE / FALSE]

Whether to insert an " and " before the smallest fractional tens place of translated numbers (e.g. "one hundred one thousandths" vs. "one hundred and one thousandths").

and_fractional is equal to and by default.

hyphenate_fractional

[TRUE / FALSE]

Whether to hyphenate numbers 21 through 99 in the fractional part of translated numbers (e.g. "twenty-one hundredths" or "twenty one hundredths"). This also determines the hyphenation of the fractional units (e.g. "one ten-millionth" vs. "one ten millionth").

hyphenate_fractional is equal to hyphenate by default.

english_fractions

[character]

A named character vector used as a dictionary for the translation of the fractional part of numbers. The names (i.e. keys) are the decimal digits of a fractional number and the values are the corresponding translations.

For example english_fractions = c("5" = "a half") matches the number 0.5 (translated as "a half") and 2.5 (translated as "two and a half").

By default english_fractions is a named character vector with translations for fractions x / y for x = 1, 2, ..., 8 and y = 1, 2, ..., 9. For example, 2 / 3 is translated as "two thirds" and 1 / 2 is translated as "one half".

Provide an empty character to english_fractions to opt out of any such translations. In this case 1 / 2 is translated as "five tenths" instead of "one half".

Value

A non-NA character vector of the same length as numbers.

Examples

```
numeric_friendly(c(1/3, 0, 0.999, NA, NaN, Inf, -Inf))

# Specify the translations of "special" numbers
numeric_friendly(c(1, 0, Inf), zero = "none", inf = "all")

# Modify the output formatting
frac <- 8765.4321
numeric_friendly(frac)
numeric_friendly(frac, decimal = " dot ")
numeric_friendly(frac, hyphenate = TRUE, hyphenate_fractional = FALSE)
numeric_friendly(frac, and = TRUE, and_fractional = TRUE, decimal = " . ")

# The `friendlynumber.numeric.digits` option specifies the number of
# numeric digits mentioned by `numeric_friendly()`
opts <- options()
options(friendlynumber.numeric.digits = 5)
numeric_friendly(0.0987654321)
```

```
options(friendlynumber.numeric.digits = 10)
numeric_friendly(0.0987654321)
options(opts)

# Set `english_fractions` to specify the translation of certain
# fractions. The names (keys) of `english_fractions` should match
# the decimal part of a fraction (e.g. `"5"` matches `0.5`).
numeric_friendly(
  c(1/2, 6/5, 12),
  english_fractions = c(`5` = "1/2", `2` = "1/5")
)

# Input validation
try(numeric_friendly_safe("A"))
```

`ordinal_friendly` *Translate integer-ish numbers to an ordinal character vector*

Description

Convert an integer vector, or numeric vector which is coercible to an integer without loss of precision, to an ordinal numeral (e.g. first, second, third).

`ordinal_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `ordinal_friendly()` does not perform input validation to maximize its speed.

Usage

```
ordinal_friendly(
  numbers,
  zero = "zeroth",
  na = "missingth",
  nan = "not a numberth",
  inf = "infinitieth",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)

ordinal_friendly_safe(
  numbers,
  zero = "zeroth",
  na = "missingth",
  nan = "not a numberth",
  inf = "infinitieth",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE
)
```

Arguments

numbers	[integer / numeric] An integer or integer-ish numeric vector to translate.
zero	[character(1)] What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)] What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)] What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)] What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)] A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
and	[TRUE / FALSE] Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE] Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.

Value

A non-NA character vector of the same length as numbers.

Examples

```
ordinal_friendly(c(0, 1, 2, 3, NA, NaN, Inf, -Inf))
ordinal_friendly(10^10)

# Specify the translations of "special" numbers
ordinal_friendly(0, zero = "noneth")

# Modify the output formatting
ordinal_friendly(1234)
ordinal_friendly(1234, and = TRUE)
ordinal_friendly(1234, hyphenate = FALSE)

# Input validation
try(ordinal_friendly_safe(0.5))
```

quantifier_friendly *Translate integer-ish numbers to a character vector of quantifiers (the, both, all three)*

Description

Convert an integer vector, or numeric vector which is coercible to an integer without loss of precision, to a quantifier (e.g. no, the, every, all five).

`quantifier_friendly_safe()` checks that all arguments are of the correct type and raises an informative error otherwise. `quantifier_friendly()` does not perform input validation to maximize its speed.

Usage

```
quantifier_friendly(
  numbers,
  one = "the",
  two = "both",
  zero = "no",
  na = "a missing",
  nan = "an undefined",
  inf = "every",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  bigmark = TRUE,
  max_friendly = 100
)

quantifier_friendly_safe(
  numbers,
  one = "the",
  two = "both",
  zero = "no",
  na = "a missing",
  nan = "an undefined",
  inf = "every",
  negative = "negative ",
  and = FALSE,
  hyphenate = TRUE,
  bigmark = TRUE,
  max_friendly = 100
)
```

Arguments

`numbers` [integer / numeric]

	An integer or integer-ish numeric vector to translate.
one	[character(1)]
	What to call values of 1 in numbers (e.g. one = "the").
two	[character(1)]
	What to call values of 2 in numbers (e.g. two = "both").
zero	[character(1)]
	What to call values of 0 in numbers (e.g. zero = "zero").
na	[character(1)]
	What to call values of NA in numbers (e.g. na = "missing").
nan	[character(1)]
	What to call values of NaN in numbers (e.g. nan = "undefined").
inf	[character(1)]
	What to call values of Inf in numbers (e.g. inf = "infinity").
negative	[character(1)]
	A prefix added to the translation of negative elements of numbers. negative is the string "negative " by default.
and	[TRUE / FALSE]
	Whether to insert an " and " before the tens place of translated numbers. and is FALSE by default.
hyphenate	[TRUE / FALSE]
	Whether to hyphenate numbers 21 through 99 (e.g. "twenty-one" vs. "twenty one"). hyphenate is TRUE by default.
bigmark	[TRUE / FALSE]
	Whether the thousands places of formatted numbers should be separated with a comma (e.g. "10,000,000" vs. "10000000"). bigmark is TRUE by default.
max_friendly	[numeric]
	The maximum number to convert to a numeral. Elements of numbers above max_friendly are converted to formatted numbers (e.g. "all 1,000" instead of "all one thousand"). max_friendly is 100 by default.
	Use the bigmark argument to determine whether these formatted numbers are comma separated (e.g. "all 1,000" vs. "all 1000").

Value

A non-NA character vector of the same length as numbers.

Examples

```
quantifier_friendly(c(0, 1, 2, 3, NA, NaN, Inf))

# The `negative` prefix appears after the `"all"` prefix
quantifier_friendly(-4)

# `-1` and `-2` are not translated using `one` and `two`
quantifier_friendly(c(1, 2, -1, -2), one = "the", two = "both")
```

```
# Suppress the translation of large numbers
quantifier_friendly(c(99, 1234), max_friendly = -Inf)
quantifier_friendly(c(99, 1234), max_friendly = 100)
quantifier_friendly(c(99, 1234), max_friendly = 1500)

# Specify the translations of "special" numbers
quantifier_friendly(c(1, Inf), one = "a", inf = "all")

# Arguments `one`, `two`, `inf`, etc. take precedence over `max_friendly`
quantifier_friendly(1:3, one = "one", two = "two", max_friendly = -1)

# Modify the output formatting
quantifier_friendly(1021, max_friendly = Inf)
quantifier_friendly(1021, and = TRUE, max_friendly = Inf)
quantifier_friendly(1021, hyphenate = FALSE, max_friendly = Inf)
quantifier_friendly(1021, bigmark = FALSE, max_friendly = 10)
quantifier_friendly(1021, bigmark = TRUE, max_friendly = 10)

# Input validation
try(quantifier_friendly_safe(1234, max_friendly = NA))
```

Index

bigfloat_friendly, 2
bigfloat_friendly(), *7*, *17*, *18*
bigfloat_friendly_safe
(bigfloat_friendly), 2
biginteger_friendly, 5
biginteger_friendly(), *17*, *18*
biginteger_friendly_safe
(biginteger_friendly), 5
bignum::bigfloat(), *2*, *3*, *7*, *17*
bignum::biginteger(), *5*, *7*, *17*

format_number, 6
friendlynumber_default_options, 8

integerish_friendly, 8
integerish_friendly(), *17*, *18*
integerish_friendly_safe
(integerish_friendly), 8

nth_friendly, 10
nth_friendly_safe (nth_friendly), 10
ntimes_friendly, 12
ntimes_friendly_safe (ntimes_friendly),
12
number_friendly, 14
number_friendly.bignum_bigfloat
(number_friendly), 14
number_friendly.bignum_biginteger
(number_friendly), 14
number_friendly.default
(number_friendly), 14
number_friendly.integer
(number_friendly), 14
number_friendly.numeric
(number_friendly), 14
number_friendly_safe (number_friendly),
14
numeric_friendly, 18
numeric_friendly(), *7*, *17*, *18*
numeric_friendly_safe
(numeric_friendly), 18

ordinal_friendly, 21
ordinal_friendly_safe
(ordinal_friendly), 21

quantifier_friendly, 23
quantifier_friendly_safe
(quantifier_friendly), 23