Package 'plu'

August 12, 2022

Type Package

Title Dynamically Pluralize Phrases

Version 0.2.3

Description Converts English phrases to singular or plural form based on the length of an associated vector. Contains helper functions to create natural language lists from vectors and to include the length of a vector in natural language.

License MIT + file LICENSE

URL https://plu.rossellhayes.com, https://github.com/rossellhayes/plu

BugReports https://github.com/rossellhayes/plu/issues

Depends R (>= 2.10)

Imports lifecycle

Suggests covr,
crayon,
fracture,
nombre,
testthat (>= 3.0.0),
withr

RdMacros lifecycle

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

R topics documented:

capitalize 2
get_fun 3
plu_more 3
plu_ral 5
plu_ralize 8
plu_stick 10

capitalize *Capitalization*

Description

capitalize() returns a character vector x with the first alphabetic character replaced with a capital form (if one exists).

Usage

```
capitalize(x)

plu_capitalize(x)

is_capital(x, strict = FALSE)

is_capitalized(x, strict = FALSE)
```

Arguments

x	A character vector.
strict	If strict is TRUE, is_capital() and is_capitalized() return FALSE instead of NA when characters are neither capital nor lowercase. Defualts to FALSE.

Details

is_capital() returns TRUE if all characters are capital, FALSE if all characters are lowercase, and NA if characters are mixed case or any characters are caseless (e.g. numbers, punctuation marks, characters from a unicase language like Arabic, Chinese or Hindi).

is_capitalized() returns TRUE if the first alphabetic character in a string is capital, FALSE if the first alphabetic character is lowercase, and NA if there are no alphabetic characters.

Value

capitalize() returns a character vector of the same length as x.

is_capital() and is_capitalized() return a logical vector of the same length as x.

Examples

```
capitalize(c("word", "a whole phrase"))
capitalize("preserving MIXED Case")
capitalize("... word")

is_capital(c("a", "A", "!"))
is_capital(c("aa", "AA", "!!"))
is_capital("Aa")

is_capitalized(c("a word", "A word", "a Word"))
is_capitalized("... A word")
is_capitalized("...")
```

get_fun *Find a function*

Description

Find a function

Usage

```
get_fun(fn, default = identity)
```

Arguments

fn	A function name, either a character string or an unquoted function name, with or without colons.
default	If fn is NULL, the default function is returned. Defaults to identity().

Value

A function

Examples

```
get_fun(plu_ral)
get_fun(plu::ral)
get_fun("plu_ral")
get_fun("plu::ral")

get_fun(NULL)
get_fun(NULL, default = plu_ral)
```

plu_more *Informatively display a maximum number of elements*

Description

Informatively display a maximum number of elements

Usage

```
plu_more(x, max = 5, type = TRUE, fn = NULL, ..., det = "more")

more(x, max = 5, type = TRUE, fn = NULL, ..., det = "more")
```

Arguments

x	A vector or list.
max	The maximum number of items to list. Additional arguments are replaced with "n more". Defaults to 5. If max if Inf, NULL, FALSE, or NA, all elements are preserved.
type	A logical or character. • If a character, type is passed to ral() and pasted after the number of elements. • If TRUE, the default, the first class of x is used as the type. – If x is a list with different classes of element, "element" is used in place of a class name. • If FALSE or NA, nothing is pasted after the number of elements.
fn	A function to apply to the number of additional elements. Default to NULL, which applies no function.
...	Additional arguments to fn.
det	A determiner to place before the number of additional elements. Defaults to "more".

Value

If x is a vector, a character vector with a length of max + 1 or less. If x is a list, a list with max + 1 or fewer elements.

Examples

```
plu::more(letters)

# Setting `max`
plu::more(letters, max = 10)
plu::more(letters, max = 27)

# If `max` is Inf or NULL, all elements will be preserved
plu::more(letters, max = Inf)

# If `max` is less than one, no elements will be preserved
plu::more(letters, max = 0)

# Setting element type
plu::more(letters, type = "letter")

# If `type` is FALSE or NULL, no type will be included
plu::more(letters, type = FALSE)

# Automatically generating type
plu::more(1:100)
plu::more(as.list(1:100))
plu::more(c(as.list(1:2), as.list(letters)))
plu::more(fracture::fracture((1:9) / (9:1)))

# Setting a determiner other than "more"
plu::more(letters, det = "other")
```

```
# Use plu::stick() to get a nicely formatted message
plu::stick(plu::more(letters))

# Applying a function to the number
plu::more(letters, fn = nombre::cardinal)
message(plu::stick(plu::more(sapply(letters, crayon::blue), fn = crayon::blue)))

# Automatic pluralization of type
fish <- c("sea bass", "crucian carp", "dace", "coelecanth")
plu::more(fish, max = 3, type = "fish")
plu::more(fish, max = 2, type = "fish")

teeth <- c("incisor", "canine", "molar", "wisdom tooth")
plu::more(teeth, max = 3, type = "tooth")
plu::more(teeth, max = 2, type = "tooth")

cacti <- c("saguaro", "prickly pear", "barrel", "star")
plu::more(cacti, max = 3, type = "cactus")
plu::more(cacti, max = 2, type = "cactus")

# Using plu_more() within a function
verbose_sqrt <- function(x) {
  if (any(x < 0)) {
    problems <- x[x < 0]
    prob_msg <- crayon::silver(encodeString(problems, quote = "`"))

    warning(
      "Square root is undefined for ",
      plu::stick(plu::more(prob_msg, fn = crayon::silver, type = "input.")),
      call. = FALSE
    )
  }

  sqrt(x)
}

ints <- round(runif(20, -10, 10))
verbose_sqrt(ints)
```

`plu_ral` *Pluralize a phrase based on the length of a vector*

Description

Pluralize a phrase based on the length of a vector

Usage

```
plu_ral(
  x,
  vector = NULL,
  n_fn = NULL,
  ...,
  n = NULL,
```

```
  pl = NULL,
  irregulars = c("moderate", "conservative", "liberal", "none"),
  replace_n = TRUE,
  open = "{",
  close = "}"
)

ral(
  x,
  vector = NULL,
  n_fn = NULL,
  ...,
  n = NULL,
  pl = NULL,
  irregulars = c("moderate", "conservative", "liberal", "none"),
  replace_n = TRUE,
  open = "{",
  close = "}"
)
```

Arguments

x	An character vector of English words or phrase to be pluralized. See details for special sequences which are handled differently.
vector	A vector whose length determines n. Defaults to NULL.
n_fn	A function to apply to the output of the special sequence "n". See examples. Defaults to identity, which returns n unchanged.
...	Additional arguments passed to the function n_fn.
n	The number which will determine the plurality of x. Defaults to length(vector). If specified, overrides vector.
pl	A logical value indicating whether to use the plural form (if TRUE) or the singular form (if FALSE) of x. Defaults to FALSE when n is 1 or -1 and TRUE for all other values. If specified, overrides n.
irregulars	What level of irregularity to use in pluralization. "moderate" uses the most common pluralization. "conservative" uses the most common irregular plural if one exists, even if a regular plural is more common. "liberal" uses a regular plural if it exists, even if an irregular plural is more common. "none" attempts to apply regular noun pluralization rules to all words. See section "Irregular plurals" for more details. Defaults to "moderate". The default can be changed by setting options(plu.irregulars). See examples in ralize() for more details.
replace_n	A logical indicating whether to use special handling for "n". See details. Defaults to TRUE.
open, close	The opening and closing delimiters for special strings. See section "Special strings". Defaults to "{" and "}".

Value

The character vector x altered to match the number of n

Irregular plurals

Many words in English have both regular and irregular plural forms. For example, the word "person" can be pluralized as "persons" or "people", and the word "formula" can be pluralized as "formulas" or "formulae". `plu` offers several options for how to handle words with multiple plurals.

- The `moderate` list attempts to apply the most common pluralization, whether it is regular or irregular. This chooses the irregular plural "people" but the regular plural "formulas".
- The `conservative` list attempts to apply an irregular plural to every word that has one. This chooses "people" and "formulae", but still uses regular plurals for words that have no irregular plural form.
- The `liberal` list attempts to apply a regular plural to every word that has one. This chooses "persons" and "formulas", but still uses irregular plurals for words that have no common regular plural, like "women". Many words in English have invariant plurals that look exactly the same as their singular forms, like "fish" or "deer". The `liberal` list attempts to use regular plurals for these words, producing "fishes" and "deers".
- The `none` list applies regular pluralization rules to all words, even those with no common regular plural. This produces, for example, "womans" as a plural for "woman" even though this is not a common English word.

Special strings

Certain strings in `x` receive special treatment.

- By default, `"a"` and `"an"` are deleted in the plural ("a word" to "words").
- The string `"n"` will be replaced with the length of `vector` or the number in `n`.
  - This output can be modified with `n_fn`.
- Strings between `open` and `close` separated by a pipe will be treated as a custom plural (`"{a|some} word"` to "a word", "some words").
  - More than two strings separated by pipes will be treated as singular, dual, trial, ... and plural forms. For example, `"{the|both|all} word"` to "the word" (1), "both words" (2), "all words" (3+).
  - See examples for more.
- Any other string between `open` and `close` without a pipe will be treated as invariant. For example, `"attorney {general}"` to "attorneys general" (notice "general" is not pluralized).

See Also

`plu_ralize()` to convert an English word to its plural form.

Examples

```
plu::ral("apple", pl = FALSE)
plu::ral("apple", pl = TRUE)

plu::ral("apple", n = 1)
plu::ral("apple", n = 2)
plu::ral("apple", n = 0)
plu::ral("apple", n = -1)
plu::ral("apple", n = 0.5)

mon <- c("apple")
tue <- c("pear", "pear")
```

```
plu::ral("apple", mon)
plu::ral("pear", tue)

paste("Monday, the caterpillar ate", plu::ral("an apple", mon))
paste("Tuesday, the caterpillar ate", plu::ral("a pear", tue))

paste("Monday, the caterpillar visited", plu::ral("an {apple} tree", mon))
paste("Tuesday, the caterpillar visited", plu::ral("a {pear} tree", tue))

paste("Monday, the caterpillar ate", plu::ral("a {single|multiple} apple", mon))
paste("Tuesday, the caterpillar ate", plu::ral("a {single|multiple} pear", tue))

later <- c(
  rep("plum", 3), rep("strawberry", 4), rep("orange", 5),
  "chocolate cake", "ice-cream cone", "pickle", "Swiss cheese", "salami",
  "lollipop", "cherry pie", "sausage", "cupcake", "watermelon"
)

paste("The caterpillar ate", plu::ral("{the|both|all of the} apple", mon))
paste("The caterpillar ate", plu::ral("{the|both|all of the} pear", tue))
paste("The caterpillar ate", plu::ral("{the|both|all of the} delicacy", later))

paste("The caterpillar ate", plu::ral("n apple", mon))
paste("The caterpillar ate", plu::ral("n delicacy", later))

paste("The caterpillar ate", plu::ral("n apple", mon, nombre::cardinal))
paste("The caterpillar ate", plu::ral("n delicacy", later, nombre::cardinal))

# Special brace strings
plu::ral("{one|two}", n = 1)
plu::ral("{one|two}", n = 2)

plu::ral("{one|two|more}", n = 1)
plu::ral("{one|two|more}", n = 2)
plu::ral("{one|two|more}", n = 3)
plu::ral("{one|two|more}", n = 50)

plu::ral("{one|two|three|more}", n = 1)
plu::ral("{one|two|three|more}", n = 2)
plu::ral("{one|two|three|more}", n = 3)
plu::ral("{one|two|three|more}", n = 50)
plu::ral("{one|two|three|more}", n = 0)
plu::ral("{one|two|three|more}", n = 1.5)
```

`plu_ralize` *Pluralize a word*

Description

Pluralize a word

Usage

```
plu_ralize(
```

```
  x,
  irregulars = getOption("plu.irregulars", c("moderate", "conservative", "liberal",
    "none"))
)

ralize(
  x,
  irregulars = getOption("plu.irregulars", c("moderate", "conservative", "liberal",
    "none"))
)
```

Arguments

x	A character vector of English words to be pluralized
irregulars	What level of irregularity to use in pluralization. "moderate" uses the most common pluralization. "conservative" uses the most common irregular plural if one exists, even if a regular plural is more common. "liberal" uses a regular plural if it exists, even if an irregular plural is more common. "none" attempts to apply regular noun pluralization rules to all words. See section "Irregular plurals" for more details. Defaults to "moderate". The default can be changed by setting options(plu.irregulars). See examples in ralize() for more details.

Value

The character vector x pluralized

Irregular plurals

Many words in English have both regular and irregular plural forms. For example, the word "person" can be pluralized as "persons" or "people", and the word "formula" can be pluralized as "formulas" or "formulae". plu offers several options for how to handle words with multiple plurals.

- The moderate list attempts to apply the most common pluralization, whether it is regular or irregular. This chooses the irregular plural "people" but the regular plural "formulas".
- The conservative list attempts to apply an irregular plural to every word that has one. This chooses "people" and "formulae", but still uses regular plurals for words that have no irregular plural form.
- The liberal list attempts to apply a regular plural to every word that has one. This chooses "persons" and "formulas", but still uses irregular plurals for words that have no common regular plural, like "women". Many words in English have invariant plurals that look exactly the same as their singular forms, like "fish" or "deer". The liberal list attempts to use regular plurals for these words, producing "fishes" and "deers".
- The none list applies regular pluralization rules to all words, even those with no common regular plural. This produces, for example, "womans" as a plural for "woman" even though this is not a common English word.

Source

Irregular plurals list adapted from the Automatically Generated Inflection Database (AGID).

See plu-package for more details.

See Also

plu_ral() to pluralize an English phrase based on a condition

Examples

```
plu::ralize("word")
plu::ralize(c("group", "word"))

plu::ralize(c("formula", "person", "child"), irregulars = "conservative")
plu::ralize(c("formula", "person", "child"), irregulars = "moderate")
plu::ralize(c("formula", "person", "child"), irregulars = "liberal")
plu::ralize(c("formula", "person", "child"), irregulars = "none")
```

---

plu_stick *Collapse a vector into a natural language string*

Description

Collapse a vector into a natural language string

Usage

```
plu_stick(
  x,
  sep = ", ",
  conj = " and ",
  oxford = getOption("plu.oxford_comma", FALSE),
  syndeton = lifecycle::deprecated(),
  fn = lifecycle::deprecated(),
  ...
)

stick(
  x,
  sep = ", ",
  conj = " and ",
  oxford = getOption("plu.oxford_comma", FALSE),
  syndeton = lifecycle::deprecated(),
  fn = lifecycle::deprecated(),
  ...
)
```

Arguments

x	A character vector (or a vector coercible to character).
sep	A character to place between list items. Defaults to ", "
conj	A character to place between the penultimate and last list items. Defaults to " and ". If NULL, sep is used.
oxford	A logical indicating whether to place sep before conj (x, y, and z) or not (x, y and z) in lists of length three or more. Defaults to FALSE. The default can be changed by setting options(plu.oxford_comma).

syndeton	**[Deprecated]**
fn	**[Deprecated]**
...	**[Deprecated]**

Value

A character vector of length 1.

Examples

```
ingredients <- c("sugar", "spice", "everything nice")
plu::stick(ingredients)
plu::stick(ingredients, conj = " or ")

# When `conj` is `NULL`, `sep` is used between all elements
plu::stick(ingredients, sep = " and ", conj = NULL)
plu::stick(ingredients, sep = "/",     conj = NULL)

creed <- c("snow", "rain", "heat", "gloom of night")
plu::stick(creed, sep = " nor ", conj = NULL)

# Oxford commas are only added when there are three or more elements
plu::stick(letters[1:3], oxford = TRUE)
plu::stick(letters[1:2], oxford = TRUE)

# Oxford commas are optional for English, but should be FALSE for most languages
ingredientes <- c("azúcar", "flores", "muchos colores")
plu::stick(ingredientes, conj = " y ", oxford = FALSE)
```