

# Package ‘SwimmeR’

August 14, 2021

**Title** Data Import, Cleaning, and Conversions for Swimming Results

**Version** 0.12.0

**Description** The goal of the 'SwimmeR' package is to provide means of acquiring, and then analyzing, data from swimming (and diving) competitions. To that end 'SwimmeR' allows results to be read in from .html sources, like 'Hy-Tek' real time results pages, '.pdf' files, 'ISL' results, 'Omega' results, and (on a development basis) '.hy3' files. Once read in, 'SwimmeR' can convert swimming times (performances) between the computationally useful format of seconds reported to the '100ths' place (e.g. 95.37), and the conventional reporting format (1:35.37) used in the swimming community. 'SwimmeR' can also score meets in a variety of formats with user defined point values, convert times between courses ('LCM', 'SCM', 'SCY') and draw single elimination brackets, as well as providing a suite of tools for working cleaning swimming data. This is a developmental package, not yet mature.

**License** MIT + file LICENSE

**Imports** purrr, dplyr, stringr, utils, rvest, pdftools, magrittr, xml2, readr

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 7.1.1

**Suggests** testthat (>= 2.1.0), knitr, rmarkdown

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Greg Pilgrim [aut, cre] (<<https://orcid.org/0000-0001-7831-442X>>), Caitlin Baldwin [ctb]

**Maintainer** Greg Pilgrim <gpilgrim2670@gmail.com>

**Depends** R (>= 3.5.0)

**Repository** CRAN

**Date/Publication** 2021-08-14 20:00:02 UTC

**R topics documented:**

add_row_numbers . . . . .	3
age_format . . . . .	4
age_format_helper . . . . .	4
collect_relay_swimmers . . . . .	5
collect_relay_swimmers_old . . . . .	5
collect_relay_swimmers_omega . . . . .	6
correct_split_distance . . . . .	6
correct_split_distance_helper . . . . .	7
course_convert . . . . .	8
course_convert_DF . . . . .	9
discard_errors . . . . .	10
dive_place . . . . .	11
draw_bracket . . . . .	11
event_parse . . . . .	13
event_parse_ISL . . . . .	13
fill_down . . . . .	14
fill_left . . . . .	14
fold . . . . .	15
format_results . . . . .	15
get_mode . . . . .	16
heat_parse_omega . . . . .	17
hy3_parse . . . . .	17
hy3_places . . . . .	18
hy3_times . . . . .	19
interleave_results . . . . .	20
is_link_broken . . . . .	20
King200Breast . . . . .	21
lines_sort . . . . .	21
list_transform . . . . .	22
mmss_format . . . . .	22
name_reorder . . . . .	23
reaction_times_parse . . . . .	24
Read_Results . . . . .	24
read_results_flag . . . . .	25
results_score . . . . .	26
samms_parse . . . . .	27
sec_format . . . . .	28
sec_format_helper . . . . .	29
splits_parse . . . . .	30
splits_parse_ISL . . . . .	30
splits_parse_omega_relays . . . . .	31
splits_reform . . . . .	32
splits_rename_omega . . . . .	32
splits_to_cumulative . . . . .	33
splits_to_cumulative_helper_recalc . . . . .	34
splits_to_lap . . . . .	35

<i>add_row_numbers</i>	3
<i>splits_to_lap_helper_recalc</i>	36
<i>SwimmeR-deprecated</i>	37
<i>Swim_Parse</i>	37
<i>swim_parse_ISL</i>	39
<i>swim_parse_old</i>	40
<i>swim_parse_omega</i>	41
<i>swim_place</i>	43
<i>tie_rescore</i>	43
<i>%notin%</i>	44
<b>Index</b>	<b>45</b>

---

<i>add_row_numbers</i>	<i>Add row numbers to raw results</i>
------------------------	---------------------------------------

---

## Description

Takes the output of `read_results` and adds row numbers to it

## Usage

```
add_row_numbers(text)
```

## Arguments

<code>text</code>	output from <code>read_results</code>
-------------------	---------------------------------------

## Value

returns a data frame with event names and row numbers to eventually be recombined with swimming results inside `swim_parse`

## See Also

`add_row_numbers` is a helper function inside [swim\\_parse](#)

---

age\_format                      *Formatting yyy-mm ages as years*

---

### Description

Takes a character string (or list) representing an age as years-months (e.g. 13-06 for 13 years, 6 months) and converts it to a character value (13.5) or a list of values representing ages in years.

### Usage

```
age_format(x)
```

### Arguments

x                      A character vector of ages in yyy-mm format (e.g. 93-03) to be converted to years (93.25)

### Value

returns the value of the string x which represents an age in yyy-mm format (93-03) and converts it to years (93.25)

### See Also

[age\\_format\\_helper](#) age\_format uses age\_format\_helper

### Examples

```
age_format("13-06")
age_format(c("13-06", "25-03", NA))
```

---

age\_format\_helper              *Helper function for formatting yyy-mm ages as years, enables vectorization of age\_format*

---

### Description

Helper function for formatting yyy-mm ages as years, enables vectorization of age\_format

### Usage

```
age_format_helper(x)
```

### Arguments

x                      A character vector of age(s) in yyyy-mm format (e.g. 13-06) to be converted to years (13.5)

---

`collect_relay_swimmers`*Collects relay swimmers as a data frame within swim\_parse*

---

**Description**

Collects relay swimmers as a data frame within swim\_parse

**Usage**

```
collect_relay_swimmers(x)
```

**Arguments**

x                    output from read\_results followed by add\_row\_numbers

**Value**

returns a data frame of relay swimmers and the associated performance row number

**See Also**

collect\_relay\_swimmers\_data runs inside of swim\_parse

---

`collect_relay_swimmers_old`*Collects relay swimmers as a data frame within swim\_parse\_old*

---

**Description**

Deprecated version associated with deprecated version of swim\_parse\_old

**Usage**

```
collect_relay_swimmers_old(x, typo_2 = typo, replacement_2 = replacement)
```

**Arguments**

x                    output from read\_results followed by add\_row\_numbers

typo\_2              list of typos from swim\_parse

replacement\_2      list of replacements for typos from swim\_parse

**Value**

returns a data frame of relay swimmers and the associated performance row number

**See Also**

collect\_relay\_swimmers runs inside of swim\_parse

---

collect\_relay\_swimmers\_omega

*Collects relay swimmers as a data frame within swim\_parse\_omega*

---

**Description**

Collects relay swimmers as a data frame within swim\_parse\_omega

**Usage**

```
collect_relay_swimmers_omega(x)
```

**Arguments**

x                      output from read\_results followed by add\_row\_numbers

**Value**

returns a data frame of relay swimmers and the associated performance row number

**See Also**

collect\_relay\_swimmers\_data runs inside of swim\_parse\_omega

---

correct\_split\_distance

*Changes lengths associated with splits to new values*

---

**Description**

Useful for dealing with meets where some events are split by 50 and others by 25.

**Usage**

```
correct_split_distance(df, new_split_length, events)
```

```
correct_split_length(df, new_split_length, events)
```

**Arguments**

df                      a data frame having some split columns (Split\_50, Split\_100 etc.)

new\_split\_length

split length to rename split columns based on

events                  list of events to correct splits for

**Value**

a data frame where all events named in the events parameter have their split column labels adjusted to reflect new\_split\_length

**Examples**

```
df <- data.frame(Name = c("Lilly King", "Caeleb Dressel"),
  Event = c("Women 100 Meter Breaststroke", "Men 50 Yard Freestyle"),
  Split_50 = c("29.80", "8.48"),
  Split_100 = c("34.33", "9.15"))

df %>% correct_split_distance(
  new_split_length = 25,
  events = c("Men 50 Yard Freestyle")
)
```

---

correct\_split\_distance\_helper

*Changes lengths associated with splits to new values*

---

**Description**

Useful for dealing with meets where some events are split by 50 and others by 25.

**Usage**

```
correct_split_distance_helper(df_helper, new_split_length_helper)
```

**Arguments**

df\_helper            a data frame having some split columns (Split\_50, Split\_100 etc.)  
new\_split\_length\_helper    split length to rename split columns based on

**Value**

a data frame where all values have been pushed left, replacing 'NA's, and all columns containing only 'NA's have been removed

**See Also**

correct\_split\_distance\_helper is a helper function inside correct\_split\_distance

---

course\_convert      *Swimming Course Convertor*

---

### Description

Used to convert times between Long Course Meters, Short Course Meters and Short Course Yards

### Usage

```
course_convert(time, event, course, course_to, verbose = FALSE)
```

### Arguments

time	A time, or vector of times to convert. Can be in either seconds (numeric, 95.97) format or swim (character, "1:35.97") format
event	The event swum as "100 Fly", "200 IM", "400 Free", "50 Back", "200 Breast" etc.
course	The course in which the time was swum as "LCM", "SCM" or "SCY"
course_to	The course to convert the time to as "LCM", "SCM" or "SCY"
verbose	If TRUE will return a data frame containing columns <ul style="list-style-type: none"> <li>• Time</li> <li>• Course</li> <li>• Course_To</li> <li>• Event</li> <li>• Time_Converted_sec</li> <li>• Time_Converted_mmss</li> </ul> . If FALSE (the default) will return only a converted time.

### Value

returns the time for a specified event and course converted to a time for the specified course\_to in swimming format OR a data frame containing columns

- Time
- Course
- Course\_To
- Event
- Time\_Converted\_sec
- Time\_Converted\_mmss

depending on the value of verbose

### Note

Relays are not presently supported.



## References

Uses the USA swimming age group method described here: <https://support.teamunify.com/en/articles/260>

## Examples

```
course_convert(time = "1:35.93", event = "200 Free", course = "SCY", course_to = "LCM")
course_convert(time = 95.93, event = "200 Free", course = "scy", course_to = "lcm")
course_convert(time = 53.89, event = "100 Fly", course = "scm", course_to = "scy")
```

---

course\_convert\_DF      *Course converter, returns data frame*

---

## Description

Used to convert times between Long Course Meters, Short Course Meters and Short Course Yards, returns data frame

## Usage

```
course_convert_DF(time, event, course, course_to)
```

```
course_convert_df(time, event, course, course_to)
```

## Arguments

time	A time, or vector of times to convert. Can be in either seconds (numeric, 95.97) format or swim (character, "1:35.97") format
event	The event swum as "100 Fly", "200 IM", "400 Free", "50 Back", "200 Breast" etc.
course	The course in which the time was swum as "LCM", "SCM" or "SCY"
course_to	The course to convert the time to as "LCM", "SCM" or "SCY"

## Value

This function returns a data frame including columns:

- Time
- Course
- Course\_To
- Event
- Time\_Converted\_sec
- Time\_Converted\_mmss

**Note**

Relays are not presently supported.

**References**

Uses the USA swimming age group method described here <https://support.teamunify.com/en/articles/260>

**Examples**

```
course_convert_df(time = "1:35.93", event = "200 Free", course = "SCY", course_to = "LCM")
course_convert_df(time = 95.93, event = "200 Free", course = "scy", course_to = "lcm")
course_convert_df(time = 53.89, event = "100 Fly", course = "scm", course_to = "scy")
```

---

discard_errors	<i>Discards elements of list that have an error value from purrr::safely.</i>
----------------	---

---

**Description**

Used in scrapping, when swim\_parse is applied over a list of results using purrr::map the result is a list of two element lists. The first element is the results, the second element is an error register. This function removes all elements where the error register is not NULL, and then returns the results (first element) of the remaining lists.

**Usage**

```
discard_errors(x)
```

**Arguments**

x a list of lists from purrr::map and purrr::safely

**Value**

a list of lists where sub lists containing a non-NULL error have been discarded and error elements have been removed from all remaining sub lists

**Examples**

```
result_1 <- data.frame(result = c(1, 2, 3))
error <- NULL

list_1 <- list(result_1, error)
names(list_1) <- c("result", "error")

result_2 <- data.frame(result = c(4, 5, 6))
```

```
error <- "result is corrupt"

list_2 <- list(result_2, error)
names(list_2) <- c("result", "error")

list_of_lists <- list(list_1, list_2)
discard_errors(list_of_lists)
```

---

dive_place	<i>Adds places to diving results</i>
------------	--------------------------------------

---

### Description

Places are awarded on the basis of score, with highest score winning. Ties are placed as ties (both athletes get 2nd etc.)

### Usage

```
dive_place(df, max_place)
```

### Arguments

df	a data frame with results from swim_parse, including only diving results (not swimming)
max_place	highest place value that scores

### Value

data frame modified so that places have been appended based on diving score

### See Also

dive\_place is a helper function used inside of results\_score

---

draw_bracket	<i>Creates a bracket for tournaments involving 5 to 64 teams, single elimination</i>
--------------	--

---

### Description

Will draw a single elimination bracket for the appropriate number of teams, inserting first round byes for higher seeds as needed

**Usage**

```
draw_bracket(
  teams,
  title = "Championship Bracket",
  text_size = 0.7,
  round_two = NULL,
  round_three = NULL,
  round_four = NULL,
  round_five = NULL,
  round_six = NULL,
  champion = NULL
)
```

**Arguments**

teams	a list of teams, ordered by desired seed, to place in bracket. Must be between 5 and 64 inclusive. Teams must have unique names
title	bracket title
text_size	number passed to cex in plotting
round_two	a list of teams advancing to the second round (need not be in order)
round_three	a list of teams advancing to the third round (need not be in order)
round_four	a list of teams advancing to the fourth round (need not be in order)
round_five	a list of teams advancing to the fifth round (need not be in order)
round_six	a list of teams advancing to the fifth round (need not be in order)
champion	the name of the overall champion team (there can be only one)

**Value**

a plot of a bracket for the teams, with results and titles as specified

**References**

based on draw.bracket from the seemingly now defunct mRchmadness package by Eli Shayer and Saber Powers and used per the terms of that package's GPL-2 license

**Examples**

```
## Not run:
teams <- c("red", "orange", "yellow", "green", "blue", "indigo", "violet")
round_two <- c("red", "yellow", "blue", "indigo")
round_three <- c("red", "blue")
champion <- "red"
draw_bracket(teams = teams,
             round_two = round_two,
             round_three = round_three,
             champion = champion)
```

```
## End(Not run)
```

---

event_parse	<i>Pulls out event labels from text</i>
-------------	---

---

### Description

Locates event labels in text of results output from `read_results` and their associated row numbers. The resulting data frame is joined back into results to include event names

### Usage

```
event_parse(text)
```

### Arguments

text            output from `read_results` followed by `add_row_numbers`

### Value

returns a data frame with event names and row numbers to eventually be recombined with swimming results inside `swim_parse`

### See Also

`event_parse` is a helper function inside [swim\\_parse](#)

---

event_parse_ISL	<i>Pulls out event labels from text</i>
-----------------	---

---

### Description

Locates event labels in text of 'ISL' results output from `read_results` and their associated row numbers. The resulting data frame is joined back into results to include event names

### Usage

```
event_parse_ISL(text)
```

### Arguments

text            output from `read_results` followed by `add_row_numbers`

**Value**

returns a data frame with event names and row numbers to eventually be recombined with swimming results inside swim\_parse\_ISL

**See Also**

event\_parse\_ISL is a helper function inside [swim\\_parse\\_ISL](#)

---

fill\_down

*Fills NA values with previous non-NA value*

---

**Description**

This is a base approximation of `tidyr::fill()`

**Usage**

```
fill_down(x)
```

**Arguments**

x a list having some number of non-NA values

**Value**

a list where NA values have been replaced with the closest previous non-NA value

**See Also**

fill\_down is a helper function inside `lines_sort`

---

fill\_left

*Shifts non-NA values to left in data frame*

---

**Description**

Moves non-NA data left into NA spaces, then removes all columns that contain only NA values

**Usage**

```
fill_left(df)
```

**Arguments**

df a data frame having some 'NA' values

**Value**

a data frame where all values have been pushed left, replacing 'NA's, and all columns containing only 'NA's have been removed

**See Also**

`fill_left` is a helper function inside `lines_sort` and `splits_parse`

---

<code>fold</code>	<i>Fold a vector onto itself</i>
-------------------	----------------------------------

---

**Description**

Fold a vector onto itself

**Usage**

```
fold(x, block.size = 1)
```

**Arguments**

<code>x</code>	a vector
<code>block.size</code>	the size of groups in which to block the data

**Value**

a new vector in the following order: first block, last block, second block, second-to-last block, ...

**References**

from the seemingly now defunct `mRchmadness` package by Eli Shayer and Saber Powers and used per the terms of that package's GPL-2 license

---

<code>format_results</code>	<i>Formats data for analysis within <code>swim_parse</code></i>
-----------------------------	---

---

**Description**

Takes the output of `read_results` and, inside of `swim_parse`, removes "special" strings like DQ and SCR from results, replacing them with NA. Also ensures that all athletes have a `Finals_Time`, by moving over `Prelims_Time`. This makes later analysis much easier.

**Usage**

```
format_results(df)
```

**Arguments**

df                    a data frame of results at the end of swim\_parse

**Value**

returns a formatted data frame

**See Also**

splits\_parse runs inside [swim\\_parse](#) on the output of [read\\_results](#) with row numbers from [add\\_row\\_numbers](#)

---

get\_mode

*Find the mode (most commonly occurring) element of a list*

---

**Description**

Determines which element of list appears most frequently. Based on `base::which.max()`, so if multiple values appear with the same frequency will return the first one. Ignores NA values. In the context of swimming data is often used to clean team names, as in the Lilly King example below.

**Usage**

```
get_mode(x, type = "first")
```

**Arguments**

x                    A list. NA elements will be ignored.

type                a character string of either "first" or "all" which determines behavior for ties. Setting `type = "first"` (the default) will return the element that appears most often and appears first in list x. Setting `type = "all"` will return all elements that appear most frequently.

**Value**

the element of x which appears most frequently. Ties go to the lowest index, so the element which appears first.

**Examples**

```
a <- c("a", "a", "b", "c")
get_mode(a)
ab <- c("a", "a", "b", "b", "c") # returns "a", not "b"
get_mode(ab)
#' ab <- c("a", "a", "b", "b", "c") # returns "a" and "b"
get_mode(ab, type = "all")
a_na <- c("a", "a", NA, NA, "c")
get_mode(a_na)
```



```
nums <- c(1, 1, 1, 2, 2, 2, 3, NA)
get_mode(nums, type = "all")

Name <- c(rep("Lilly King", 5))
Team <- c(rep("IU", 2), "Indiana", "IUWSD", "Indiana University")
df <- data.frame(Name, Team, stringsAsFactors = FALSE)
df$Team <- get_mode(df$Team)
```

---

heat\_parse\_omega      *Pulls out heat labels from text*

---

### Description

Locates heat labels in text of results output from `read_results` and their associated row numbers. The resulting data frame is joined back into results to include heat numbers

### Usage

```
heat_parse_omega(text)
```

### Arguments

text                  output from `read_results` followed by `add_row_numbers`

### Value

returns a data frame with heat names and row numbers to eventually be recombined with swimming results inside `swim_parse_omega`

### See Also

heat\_parse\_omega is a helper function inside [swim\\_parse\\_omega](#)

---

hy3\_parse              *Parses Hy-Tek .hy3 files*

---

### Description

Helper function used inside ‘`swim_parse`’ for dealing with Hy-Tek `.hy3` files. Can have more columns than other ‘`swim_parse`’ outputs, because `.hy3` files can contain more data

**Usage**

```

hy3_parse(
  file,
  avoid = avoid_minimal,
  typo = typo_default,
  replacement = replacement_default
)

```

**Arguments**

file	output from <code>read_results</code>
avoid	a list of strings. Rows in <code>x</code> containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to <code>avoid</code> . The default is <code>avoid_default</code> , which contains many strings similar to "Pool:", such as "STATE:" and "Qual:". Users can supply their own lists to <code>avoid</code> .
typo	a list of strings that are typos in the original results. <code>swim_parse</code> is particularly sensitive to accidental double spaces, so "Central High School", with two spaces between "Central" and "High" is a problem, which can be fixed. Pass "Central High School" to <code>typo</code> . Unexpected commas as also an issue, for example "Texas, University of" should be fixed using <code>typo</code> and <code>replacement</code>
replacement	a list of fixes for the strings in <code>typo</code> . Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to <code>replacement</code> fix the issues described in <code>typo</code>

**Value**

returns a data frame with columns Name, Place, Age, Team, Prelims\_Time, Finals\_Time, & Event. May also contain Seed\_Time, USA\_ID, and/or Birthdate. Note all swims will have a Finals\_Time, even if that time was actually swam in the prelims (i.e. a swimmer did not qualify for finals). This is so that final results for an event can be generated from just one column.

**See Also**

`parse_hy3` must be run on the output of [read\\_results](#)

`parse_hy3` runs inside of [swim\\_parse](#)

---

hy3\_places

*Helper for reading prelims and finals places from Hy-Tek .hy3 files*

---

**Description**

Used to pull prelims and finals places from .hy3 files as part of parsing them.

**Usage**

```
hy3_places(  
  file,  
  type = c("prelims", "relay_prelims", "finals", "relay_finals")  
)
```

**Arguments**

file            an output of read\_results, from an .hy3 file  
type            type of times, either "prelims", "relay\_prelims", "finals" or "relay\_finals"

**Value**

a data frame where column 1 is times and column 2 is row number

**See Also**

hy3\_places is run inside of [hy3\\_parse](#)

---

hy3\_times

*Helper for reading prelims and finals times from Hy-Tek .hy3 files*

---

**Description**

Used to pull prelims and finals times from .hy3 files as part of parsing them.

**Usage**

```
hy3_times(file, type = c("prelims", "relay_prelims", "finals", "relay_finals"))
```

**Arguments**

file            an output of read\_results, from an .hy3 file  
type            type of times, either "prelims", "relay\_prelims", "finals" or "relay\_finals"

**Value**

a data frame where column 1 is times and column 2 is row number

**See Also**

hy3\_times is run inside of [hy3\\_parse](#)

---

interleave\_results      *Helper for reading interleaving prelims and finals results*

---

### Description

Interleaves times or places based on row number ranges.

### Usage

```
interleave_results(entries, results, type = c("individual", "relay"))
```

### Arguments

entries	a data frame containing columns for minimum and maximum row number (usually 'Row_Min' and 'Row_Max'). Times or places will be interleaved into this data frame.
results	a data frame containing times (or places) in column 1 (or other values to be interleaved) and row numbers in column 2 (usually 'Row_Numb').
type	either "individual" or "relay"

### Value

a modified version of 'entries' with values from 'results' interleaved on the basis of row number

### See Also

interleave\_results is a helper function used in [hy3\\_parse](#)

---

is\_link\_broken      *Determines if a link is valid*

---

### Description

Used in testing links to external data, specifically inside of internal package tests. Attempts to connect to link for the length of duration (in s). If it fails it returns FALSE

### Usage

```
is_link_broken(link_to_test, duration = 1)
```

### Arguments

link_to_test	a link
duration	the lowest row number

**Value**

TRUE if the link works, FALSE if it fails

---

King200Breast	<i>Results for Lilly King's 200 Breaststrokes</i>
---------------	---

---

**Description**

Lilly King's 200 Breaststroke swims from her NCAA career

**Usage**

```
data(King200Breast)
```

**Format**

An object of class "data.frame"

**Source**

[NCAA Times Database](#)

---

lines_sort	<i>Sorts and collects lines by performance and row number</i>
------------	---

---

**Description**

Collects all lines, (for example containing splits or relay swimmers) associated with a particular performance (a swim) into a data frame with the appropriate row number for that performance

**Usage**

```
lines_sort(x, min_row = minimum_row)
```

**Arguments**

x	a list of character strings including performances, with tow numbers added by add_row_numbers
min_row	the lowest row number

**Value**

a data frame with Row\_Numb as the first column. Other columns are performance elements, like splits or relay swimmers, both in order of occurrence left to right

**See Also**

lines\_sort is a helper function inside splits\_parse and swim\_parse\_ISL

---

list_transform	<i>Transform list of lists into data frame</i>
----------------	--

---

**Description**

Converts list of lists, with all sub-lists having the same number of elements into a data frame where each sub-list is a row and each element a column

**Usage**

```
list_transform(x)
```

**Arguments**

x a list of lists, with all sub-lists having the same length

**Value**

a data frame where each sub-list is a row and each element of that sub-list is a column

**See Also**

list\_transform is a helper function used inside of swim\_parse, swim\_parse\_ISL, event\_parse and event\_parse\_ISL

---

mmss_format	<i>Formatting seconds as mm:ss.hh</i>
-------------	---------------------------------------

---

**Description**

Takes a numeric item or list of numeric items representing seconds (e.g. 95.37) and converts to a character string or list of strings in swimming format ("1:35.37").

**Usage**

```
mmss_format(x)
```

**Arguments**

x A number of seconds to be converted to swimming format

**Value**

the number of seconds x converted to conventional swimming format mm:ss.hh

**See Also**

[sec\\_format](#) mmss\_format is the reverse of sec\_format

**Examples**

```
mmss_format(95.37)
mmss_format(200.95)
mmss_format(59.47)
mmss_format(c(95.37, 200.95, 59.47, NA))
```

---

name_reorder	<i>Orders all names as "Firstname Lastname"</i>
--------------	---

---

**Description**

Names are sometimes listed as Firstname Lastname, and sometimes as Lastname, Firstname. The `names_reorder` function converts all names to Firstname Lastname based on comma position. The reverse, going to Lastname, Firstname is not possible because some athletes have multiple first names or multiple last names and without the comma to differentiate between the two a distinction cannot be made.

**Usage**

```
name_reorder(x, verbose = FALSE)
```

**Arguments**

x	a data frame output from <code>swim_parse</code> containing a column called <code>Name</code> with some names as Lastname, Firstname
verbose	defaults to FALSE. If set to TRUE and if x is a data frame then returned data frame will include columns <code>First_Name</code> and <code>Last_Name</code> extracted as best as possible from <code>Name</code>

**Value**

a data frame with a column `Name_Reorder`, or a list, containing strings reordered as Firstname Lastname in addition to all other columns in input df. Can also contain columns `First_Name` and `Last_Name` depending on value of `verbose` argument

**Examples**

```
name_reorder(
  data.frame(
    Name = c("King, Lilly",
            "Lilly King",
            NA,
            "Richards Ross, Sanya",
```

```
    "Phelps, Michael F")),  
  verbose = TRUE  
)  
name_reorder(c("King, Lilly", "Lilly King", NA, "Richards Ross, Sanya"))
```

---

reaction\_times\_parse *Pulls out reaction times from text*

---

### Description

Locates reaction times in text of results output from `read_results` and their associated row numbers. The resulting data frame is joined back into results to include reaction times

### Usage

```
reaction_times_parse(text)
```

### Arguments

text                    output from `read_results` followed by `add_row_numbers`

### Value

returns a data frame with reaction times and row numbers to eventually be recombined with swimming results inside `swim_parse`

### See Also

`reaction_times_parse` is a helper function inside [swim\\_parse](#)

---

Read\_Results            *Reads swimming and diving results into a list of strings in preparation for parsing with `swim_parse`*

---

### Description

Outputs list of strings to be processed by `swim_parse`

### Usage

```
Read_Results(file, node = "pre")
```

```
read_results(file, node = "pre")
```



**Arguments**

file	a .pdf or .html file (could be a url) where containing swimming results. Must be formatted in a "normal" fashion - see vignette
node	a CSS node where html results are stored. Required for html results. Default is "pre", which nearly always works.

**Value**

returns a list of strings containing the information from file. Should then be parsed with swim\_parse

**See Also**

read\_results is meant to be followed by [swim\\_parse](#)

**Examples**

```
## Not run: read_results("http://www.nyhsswim.com/Results/Boys/2008/NYS/Single.htm", node = "pre")
```

---

read_results_flag	<i>used to indicate that results have been read in with read_results prior to being parsed by swim_parse</i>
-------------------	--

---

**Description**

Used to insure that read\_results has been run on a data source prior to running swim\_parse

**Usage**

```
read_results_flag(x)
```

**Arguments**

x	a list of results, line by line
---	---------------------------------

**Value**

returns list x, with "read\_results\_flag" added as the first element of the list

---

results_score	<i>Scores a swim meet</i>
---------------	---------------------------

---

### Description

Used to add a Points column with point values for each place. Can either score "timed finals" type meets where any athlete can get any place, or "prelims-finals", type meets, where placing is restricted by prelim performance.

### Usage

```
results_score(
  results,
  events,
  meet_type = c("timed_finals", "prelims_finals"),
  lanes = c(4, 6, 8, 10),
  scoring_heats = c(1, 2, 3),
  point_values
)
```

### Arguments

results	an output from swim_parse
events	list of events
meet_type	how to score based on timed_finals, where any place is possible, or prelims_finals where athletes are locked into heats for scoring purposes
lanes	number of lanes in to the pool, for purposes of heat
scoring_heats	number of heats which score (if 1 only A final scores, if 2 A and B final score etc.)
point_values	a list of point values for each scoring place

### Value

results with point values in a column called Points

### Examples

```
## Not run:
file <-
system.file("extdata", "BigTen_WSWIM_2018.pdf", package = "SwimmeR")
BigTenRaw <- read_results(file)

BigTen <- swim_parse(
  BigTenRaw,
  typo = c(
    "\\s{1,}\\s*",
    "\\s{1,}(\\d{1,2})\\s{2,}"
  )
)
```

```

    ",\\s{1,}University\\s{1,}of",
    "University\\s{1,}of\\s{1,}",
    "\\s{1,}University",
    "SR\\s{2,}",
    "JR\\s{2,}",
    "SO\\s{2,}",
    "FR\\s{2,}"
  ),
  replacement = c(" ",
                  "\\1 ",
                  "", "", "",
                  "SR ",
                  "JR ",
                  "SO ",
                  "FR "),
  avoid = c("B1G", "Pool")
)

BigTen <- BigTen %>%
  dplyr::filter(
    stringr::str_detect(Event, "Time Trial") == FALSE,
    stringr::str_detect(Event, "Swim-off") == FALSE
  ) %>%
  dplyr::mutate(Team = dplyr::case_when(Team == "Wisconsin, Madi" ~ "Wisconsin",
                                       TRUE ~ Team))

# begin results_score portion
df <- BigTen %>%
  results_score(
    events = unique(BigTen$Event),
    meet_type = "prelims_finals",
    lanes = 8,
    scoring_heats = 3,
    point_values = c(
      32, 28, 27, 26, 25, 24, 23, 22, 20, 17, 16, 15, 14, 13, 12, 11, 9, 7, 6, 5, 4, 3, 2, 1)
  )

## End(Not run)

```

---

 samms\_parse

*Formats swimming and diving data read with read\_results into a dataframe*

---

### Description

Takes the output of read\_results of S.A.M.M.S. results and cleans it, yielding a dataframe of swimming (and diving) results

**Usage**

```

samms_parse(
  file_samms,
  avoid_samms = avoid,
  typo_samms = typo,
  replacement_samms = replacement,
  format_samms = format_results
)

```

**Arguments**

file_samms	output from read_results of S.A.M.M.S. style results
avoid_samms	a list of strings. Rows in file containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to avoid. The default is avoid_default, which contains many strings similar to "Pool:", such as "STATE:" and "Qual:". Users can supply their own lists to avoid.
typo_samms	a list of strings that are typos in the original results. swim_parse is particularly sensitive to accidental double spaces, so "Central High School", with two spaces between "Central" and "High" is a problem, which can be fixed. Pass "Central High School" to typo. Unexpected commas as also an issue, for example "Texas, University of" should be fixed using typo and replacement
replacement_samms	a list of fixes for the strings in typo. Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to replacement fix the issues described in typo
format_samms	should the data be formatted for analysis (special strings like "DQ" replaced with NA, Finals_Time as definitive column)? Default is TRUE

**Value**

returns a data frame with columns Name, Place, Age, Team, Prelims\_Time, Finals\_Time, Event & DQ. Note all swims will have a Finals\_Time, even if that time was actually swam in the prelims (i.e. a swimmer did not qualify for finals). This is so that final results for an event can be generated from just one column.

**See Also**

swim\_parse must be run on the output of [read\\_results](#)

---

sec\_format

*Formatting mm:ss.tt times as seconds*

---

**Description**

Takes a character string (or list) representing time in swimming format (e.g. 1:35.37) and converts it to a numeric value (95.37) or a list of values representing seconds.

**Usage**

```
sec_format(x)
```

**Arguments**

x                    A character vector of time(s) in swimming format (e.g. 1:35.93) to be converted to seconds (95.93)

**Value**

returns the value of the string x which represents a time in swimming format (mm:ss.hh) and converts it to seconds

**See Also**

sec\_format is the reverse of [mmss\\_format](#)

**Examples**

```
sec_format("1:35.93")
sec_format("16:45.19")
sec_format("25.43")
sec_format(c("1:35.93", "16:45.19", "25.43"))
sec_format(c("1:35.93", "16:45.19", NA, "25.43"))
```

---

sec_format_helper	<i>Helper function for formatting mm:ss.hh times as seconds, used to enable vectorized operation of sec_format</i>
-------------------	--

---

**Description**

Helper function for formatting mm:ss.hh times as seconds, used to enable vectorized operation of sec\_format

**Usage**

```
sec_format_helper(x)
```

**Arguments**

x                    A character vector of time(s) in swimming format (e.g. 1:35.93) to be converted to seconds (95.93)

---

splits_parse	<i>Collects splits within swim_parse</i>
--------------	--

---

**Description**

Takes the output of read\_results and, inside of swim\_parse, extracts split times and associated row numbers

**Usage**

```
splits_parse(text, split_len = split_length)
```

**Arguments**

text	output of read_results with row numbers appended by add_row_numbers
split_len	length of pool at which splits are measured - usually 25 or 50

**Value**

returns a dataframe with split times and row numbers

**Author(s)**

Greg Pilgrim <gpilgrim2670@gmail.com>

**See Also**

splits\_parse runs inside [swim\\_parse](#) on the output of [read\\_results](#) with row numbers from [add\\_row\\_numbers](#)

---

splits_parse_ISL	<i>Collects splits within swim_parse_ISL</i>
------------------	--

---

**Description**

Takes the output of read\_results and, inside of swim\_parse\_ISL, extracts split times and associated row numbers

**Usage**

```
splits_parse_ISL(text)
```

**Arguments**

text	output of read_results with row numbers appended by add_row_numbers
------	---

**Value**

returns a data frame with split times and row numbers

**See Also**

splits\_parse\_ISL runs inside [swim\\_parse\\_ISL](#) on the output of [read\\_results](#) with row numbers from [add\\_row\\_numbers](#)

---

splits\_parse\_omega\_relays

*Collects splits for relays within swim\_parse\_omega*

---

**Description**

Takes the output of [read\\_results](#) and, inside of [swim\\_parse\\_omega](#), extracts split times and associated row numbers

**Usage**

```
splits_parse_omega_relays(text, split_len = split_length_omega)
```

**Arguments**

`text` output of [read\\_results](#) with row numbers appended by [add\\_row\\_numbers](#)  
`split_len` length of pool at which splits are measured - usually 25 or 50

**Value**

returns a dataframe with split times and row numbers

**Author(s)**

Greg Pilgrim <[gpilgrim2670@gmail.com](mailto:gpilgrim2670@gmail.com)>

**See Also**

splits\_parse runs inside [swim\\_parse\\_omega](#) on the output of [read\\_results](#) with row numbers from [add\\_row\\_numbers](#)

---

splits_reform	<i>Adds together splits and compares to listed finals time to see if they match.</i>
---------------	--

---

### Description

Used in testing the workings for `split_parse` inside `test-splits.R`. Note that even properly handled splits may not match the finals time due to issues in the source material. Sometimes splits aren't fully recorded in the source. Some relays also will not match due to the convention of reporting splits by swimmer (see vignette for more details).

### Usage

```
splits_reform(df)
```

### Arguments

`df` a data frame output from `swim_parse` created with `splits = TRUE`

### Value

a data frame with a column `not_matching` containing `TRUE` if the splits for that swim match the finals time and `FALSE` if they do not

### Author(s)

Greg Pilgrim <gpilgrim2670@gmail.com>

---

splits_rename_omega	<i>Advances split names by one split_length</i>
---------------------	---

---

### Description

Used to adjust names of splits inside `swim_parse_omega` to account for 50 split not being correctly captured

### Usage

```
splits_rename_omega(x, split_len = split_length_omega)
```

### Arguments

`x` a string to rename, from columns output by `splits_parse`  
`split_len` distance for each split



**Value**

returns string iterated up by split\_length

**See Also**

splits\_rename\_omega runs inside [swim\\_parse\\_omega](#) on the output of [splits\\_parse](#)

---

splits\_to\_cumulative *Converts splits from lap to cumulative format*

---

**Description**

Cumulative splits are when each split is the total elapsed time at a given distance. For example, if an athlete swims the first 50 of a 200 yard race in 25.00 seconds (lap and cumulative split), and the second 50 (i.e. the 100 lap split) in 30.00 seconds the cumulative 100 split is 25.00 + 30.00 = 55.00. Some swimming results are reported with lap splits (preferred), but others use cumulative splits. This function converts lap splits to cumulative splits.

**Usage**

```
splits_to_cumulative(df, threshold = Inf)
```

**Arguments**

df a data frame containing results with splits in lap format. Must be formatted in a "normal" SwimmeR fashion - see vignette

threshold a numeric value above which a split is taken to be cumulative. Default is Inf

**Value**

a data frame with all splits in lap form

**See Also**

splits\_to\_cumulative is the reverse of [splits\\_to\\_lap](#)

**Examples**

```
## Not run:
df <- data.frame(Place = rep(1, 2),
                 Name = c("Lenore Lap", "Casey Cumulative"),
                 Team = rep("KVAC", 2),
                 Event = rep("Womens 200 Freestyle", 2),
                 Finals_Time = rep("1:58.00", 2),
                 Split_50 = rep("28.00", 2),
                 Split_100 = c("31.00", "59.00"),
                 Split_150 = c("30.00", "1:29.00"),
                 Split_200 = c("29.00", "1:58.00"))
```

```

    )

    # since one entry is in lap time and the other is cumulative, need to
    # set threshold value

    # not setting threshold will produce bad results by attempting to convert
    # Casey Cumulative's splits, which are already in cumulative
    # format, into cumulative format again

    df %>%
      splits_to_cumulative()

    df %>%
      splits_to_cumulative(threshold = 20)

## End(Not run)

```

---

```
splits_to_cumulative_helper_recalc
```

*Helper function for converting lap splits to cumulative splits*

---

## Description

Helper function for converting lap splits to cumulative splits

## Usage

```
splits_to_cumulative_helper_recalc(
  df,
  i,
  split_cols = split_cols,
  threshold = threshold
)
```

## Arguments

df	a data frame containing splits in lap format
i	list of values to iterate along
split_cols	list of columns containing splits
threshold	a numeric value below which a split is taken to be lap

## Value

a list of data frames with all splits in cumulative format for a particular event, each with a single split column converted to cumulative format

---

splits_to_lap	<i>Converts splits from cumulative to lap format</i>
---------------	--

---

### Description

Cumulative splits are when each split is the total elapsed time at a given distance. For example, if an athlete swims the first 50 of a 200 yard race in 25.00 seconds (lap and cumulative split), and the second 50 (i.e. the 100 lap split) in 30.00 seconds the cumulative 100 split is 25.00 + 30.00 = 55.00. Some swimming results are reported with lap splits (preferred), but others use cumulative splits. This function converts cumulative splits to lap splits.

### Usage

```
splits_to_lap(df, threshold = -Inf)
```

### Arguments

df	a data frame containing results with splits in cumulative format. Must be formatted in a "normal" SwimmeR fashion - see vignette
threshold	a numeric value below which a split is taken to be cumulative. Default is -Inf

### Value

a data frame with all splits in lap form

### See Also

splits\_to\_lap is the reverse of [splits\\_to\\_cumulative](#)

### Examples

```
## Not run:
df <- data.frame(Place = 1,
                 Name = "Sally Swimfast",
                 Team = "KVAC",
                 Event = "Womens 200 Freestyle",
                 Finals_Time = "1:58.00",
                 Split_50 = "28.00",
                 Split_100 = "59.00",
                 Split_150 = "1:31.00",
                 Split_200 = "1:58.00")

df %>%
  splits_to_lap

df <- data.frame(Place = rep(1, 2),
                 Name = c("Lenore Lap", "Casey Cumulative"),
                 Team = rep("KVAC", 2),
                 Event = rep("Womens 200 Freestyle", 2),
```

```

    Finals_Time = rep("1:58.00", 2),
    Split_50 = rep("28.00", 2),
    Split_100 = c("31.00", "59.00"),
    Split_150 = c("30.00", "1:29.00"),
    Split_200 = c("29.00", "1:58.00")
  )

# since one entry is in lap time and the other is cumulative, need to
# set threshold value

# not setting threshold will produce bad results by attempting to convert
# Lenore Lap's splits, which are already in lap format, into lap format
# again

df %>%
  splits_to_lap()

df %>%
  splits_to_lap(threshold = 35)

## End(Not run)

```

---

```
splits_to_lap_helper_recalc
```

*Helper function for converting cumulative splits to lap splits*

---

## Description

Helper function for converting cumulative splits to lap splits

## Usage

```

splits_to_lap_helper_recalc(
  df,
  i,
  split_cols = split_cols,
  threshold = threshold
)

```

## Arguments

df	a data frame containing splits in cumulative format
i	list of values to iterate along
split_cols	list of columns containing splits
threshold	a numeric value above which a split is taken to be cumulative

**Value**

a list of data frames with all splits in lap format for a particular event, each with a single split column converted to lap format

---

SwimmeR-deprecated      *Deprecated functions in SwimmeR*

---

**Description**

These functions still work but will be removed (defunct) in upcoming versions.

**Details**

- [course\\_convert\\_DF](#): This function is deprecated, and will be removed in an upcoming version this SwimmeR. Instead please use `course_convert(verbose = TRUE)`

---

Swim\_Parse      *Formats swimming and diving data read with read\_results into a data frame*

---

**Description**

Takes the output of `read_results` and cleans it, yielding a data frame of swimming (and diving) results

**Usage**

```
Swim_Parse(
  file,
  avoid = avoid_default,
  typo = typo_default,
  replacement = replacement_default,
  format_results = TRUE,
  splits = FALSE,
  split_length = 50,
  relay_swimmers = FALSE
)
```

```
swim_parse(
  file,
  avoid = avoid_default,
  typo = typo_default,
  replacement = replacement_default,
  format_results = TRUE,
  splits = FALSE,
```

```

    split_length = 50,
    relay_swimmers = FALSE
)

```

### Arguments

file	output from read_results
avoid	a list of strings. Rows in file containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to avoid. The default is avoid_default, which contains many strings similar to "Pool:", such as "STATE:" and "Qual:". Users can supply their own lists to avoid. avoid is handled before typo and replacement.
typo	a list of strings that are typos in the original results. swim_parse is particularly sensitive to accidental double spaces, so "Central High School", with two spaces between "Central" and "High" is a problem, which can be fixed. Pass "Central High School" to typo. Unexpected commas as also an issue, for example "Texas, University of" should be fixed using typo and replacement
replacement	a list of fixes for the strings in typo. Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to replacement fix the issues described in typo
format_results	should the results be formatted for analysis (special strings like "DQ" replaced with NA, Finals_Time as definitive column)? Default is TRUE
splits	either TRUE or the default, FALSE - should swim_parse attempt to include splits.
split_length	either 25 or the default, 50, the length of pool at which splits are recorded. Not all results are internally consistent on this issue - some have races with splits by 50 and other races with splits by 25.
relay_swimmers	either TRUE or the default, FALSE - should relay swimmers be reported. Relay swimmers are reported in separate columns named Relay_Swimmer_1 etc.

### Value

returns a data frame with columns Name, Place, Age, Team, Prelims\_Time, Finals\_Time, Points, Event & DQ. Note all swims will have a Finals\_Time, even if that time was actually swam in the prelims (i.e. a swimmer did not qualify for finals). This is so that final results for an event can be generated from just one column.

### See Also

swim\_parse must be run on the output of [read\\_results](#)

### Examples

```

## Not run:
swim_parse(read_results("http://www.nyhsswim.com/Results/Boys/2008/NYS/Single.htm", node = "pre"),
  typo = c("-1NORTH ROCKL"), replacement = c("1-NORTH ROCKL"),
  splits = TRUE,
  relay_swimmers = TRUE)

```

```
## End(Not run)
## Not run:
swim_parse(read_results("inst/extdata/Texas-Florida-Indiana.pdf"),
  typo = c("Indiana University", ", University of"), replacement = c("Indiana University", ""),
  splits = TRUE,
  relay_swimmers = TRUE)

## End(Not run)
```

---

swim_parse_ISL	<i>Formats swimming results from the International Swim League ('ISL') read with read_results into a data frame</i>
----------------	---

---

## Description

Takes the output of `read_results` and cleans it, yielding a data frame of 'ISL' swimming results

## Usage

```
swim_parse_ISL(file, splits = FALSE, relay_swimmers = FALSE)
```

```
Swim_Parse_ISL(file, splits = FALSE, relay_swimmers = FALSE)
```

## Arguments

`file` output from `read_results`  
`splits` should splits be included, default is FALSE  
`relay_swimmers` should relay swimmers be included as separate columns, default is FALSE

## Value

returns a data frame of ISL results

## Author(s)

Greg Pilgrim <gpilgrim2670@gmail.com>

## See Also

`swim_parse_ISL` must be run on the output of [read\\_results](#)

**Examples**

```
## Not run:
swim_parse_ISL(
  read_results(
    "https://isl.global/wp-content/uploads/2019/11/isl_college_park_results_day_2.pdf"),
  splits = TRUE,
  relay_swimmers = TRUE)

## End(Not run)
```

---

swim_parse_old	<i>Formats swimming and diving data read with read_results into a data frame</i>
----------------	--

---

**Description**

Takes the output of `read_results` and cleans it, yielding a data frame of swimming (and diving) results. Old version, retired in dev build on Dec 21, 2020 and release version 0.7.0

**Usage**

```
swim_parse_old(
  file,
  avoid = avoid_default,
  typo = typo_default,
  replacement = replacement_default,
  splits = FALSE,
  split_length = 50,
  relay_swimmers = FALSE
)
```

**Arguments**

<code>file</code>	output from <code>read_results</code>
<code>avoid</code>	a list of strings. Rows in <code>file</code> containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to <code>avoid</code> . The default is <code>avoid_default</code> , which contains many strings similar to "Pool:", such as "STATE:" and "Qual:". Users can supply their own lists to <code>avoid</code> .
<code>typo</code>	a list of strings that are typos in the original results. <code>swim_parse_old</code> is particularly sensitive to accidental double spaces, so "Central High School", with two spaces between "Central" and "High" is a problem, which can be fixed. Pass "Central High School" to <code>typo</code> . Unexpected commas as also an issue, for example "Texas, University of" should be fixed using <code>typo</code> and <code>replacement</code>
<code>replacement</code>	a list of fixes for the strings in <code>typo</code> . Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to <code>replacement</code> fix the issues described in <code>typo</code>



splits	either TRUE or the default, FALSE - should swim_parse_old attempt to include splits.
split_length	either 25 or the default, 50, the length of pool at which splits are recorded. Not all results are internally consistent on this issue - some have races with splits by 50 and other races with splits by 25.
relay_swimmers	either TRUE or the default, FALSE - should relay swimmers be reported. Relay swimmers are reported in separate columns named Relay_Swimmer_1 etc.

**Value**

returns a data frame with columns Name, Place, Age, Team, Prelims\_Time, Finals\_Time, Points, Event & DQ. Note all swims will have a Finals\_Time, even if that time was actually swam in the prelims (i.e. a swimmer did not qualify for finals). This is so that final results for an event can be generated from just one column.

**See Also**

swim\_parse\_old must be run on the output of [read\\_results](#)

**Examples**

```
## Not run:
swim_parse_old(
  read_results("http://www.nyhsswim.com/Results/Boys/2008/NYS/Single.htm", node = "pre"),
  typo = c("-1NORTH ROCKL"), replacement = c("1-NORTH ROCKL"),
  splits = TRUE,
  relay_swimmers = TRUE)

## End(Not run)
## Not run:
swim_parse_old(read_results("inst/extdata/Texas-Florida-Indiana.pdf"),
  typo = c("Indiana University", "", University of"), replacement = c("Indiana University", "")),
  splits = TRUE,
  relay_swimmers = TRUE)

## End(Not run)
```

---

swim_parse_omega	<i>Formats swimming and diving data read with read_results into a data frame</i>
------------------	--

---

**Description**

Takes the output of read\_results and cleans it, yielding a data frame of swimming (and diving) results

**Usage**

```
swim_parse_omega(
  file_omega,
  avoid_omega = avoid,
  typo_omega = typo,
  replacement_omega = replacement,
  format_results = TRUE,
  splits = FALSE,
  split_length_omega = split_length,
  relay_swimmers_omega = relay_swimmers
)
```

**Arguments**

**file\_omega** output from read\_results

**avoid\_omega** a list of strings. Rows in file\_omega containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to avoid\_omega. The default is avoid\_default, which contains many strings similar to "Pool:", such as "STATE:" and "Qual:". Users can supply their own lists to avoid\_omega. avoid\_omega is handled before typo\_omega and replacement\_omega.

**typo\_omega** a list of strings that are typos in the original results. swim\_parse is particularly sensitive to accidental double spaces, so "Central High School", with two spaces between "Central" and "High" is a problem, which can be fixed. Pass "Central High School" to typo\_omega. Unexpected commas as also an issue, for example "Texas, University of" should be fixed using typo\_omega and replacement\_omega

**replacement\_omega** a list of fixes for the strings in typo\_omega. Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to replacement\_omega fix the issues described in typo\_omega

**format\_results** should the results be formatted for analysis (special strings like "DQ" replaced with NA, Finals\_Time as definitive column)? Default is TRUE

**splits** either TRUE or the default, FALSE - should swim\_parse attempt to include splits.

**split\_length\_omega** either 25 or the default, 50, the length of pool at which splits are recorded. Not all results are internally consistent on this issue - some have races with splits by 50 and other races with splits by 25.

**relay\_swimmers\_omega** should names of relay swimmers be captured? Default is FALSE

**Value**

returns a data frame with columns Name, Place, Age, Team, Prelims\_Time, Finals\_Time, Points, Event & DQ. Note all swims will have a Finals\_Time, even if that time was actually swam in the prelims (i.e. a swimmer did not qualify for finals). This is so that final results for an event can be generated from just one column.

**See Also**

swim\_parse\_omega must be run on the output of [read\\_results](#)

---

swim_place	<i>Adds places to swimming results</i>
------------	--

---

**Description**

Places are awarded on the basis of time, with fastest (lowest) time winning. Ties are placed as ties (both athletes get 2nd etc.)

**Usage**

```
swim_place(df, max_place)
```

**Arguments**

df	a data frame with results from swim_parse, including only swimming results (not diving)
max_place	highest place value that scores

**Value**

a data frame modified so that places have been appended based on swimming time

**See Also**

swim\_place is a helper function used inside of results\_score

---

tie_rescore	<i>Rescore to account for ties</i>
-------------	------------------------------------

---

**Description**

Rescoring to average point values for ties. Ties are placed as ties (both athletes get 2nd etc.)

**Usage**

```
tie_rescore(df, point_values, lanes)
```

**Arguments**

df	a data frame with results from swim_parse, with places from swim_place and/or dive_place
point_values	a named list of point values for each scoring place
lanes	number of scoring lanes in the pool

**Value**

df modified so that places have been appended based on swimming time

**See Also**

`tie_rescore` is a helper function used inside of `results_score`

---

`%notin%`*"Not in" function*

---

**Description**

The opposite of `'FALSE'` otherwise.

**Usage**

```
x %notin% y
```

```
x %!in% y
```

**Arguments**

x                    a value

y                    a list of values

**Value**

a `'TRUE'` or `'FALSE'`

**Examples**

```
"a" %!in% c("a", "b", "c")  
"a" %notin% c("b", "c")
```

# Index

## \* datasets

- King200Breast, 21
- %!in% (%notin%), 44
- %notin%, 44
  
- add\_row\_numbers, 3, 16, 30, 31
- age\_format, 4
- age\_format\_helper, 4, 4
  
- collect\_relay\_swimmers, 5
- collect\_relay\_swimmers\_old, 5
- collect\_relay\_swimmers\_omega, 6
- correct\_split\_distance, 6
- correct\_split\_distance\_helper, 7
- correct\_split\_length
  - (correct\_split\_distance), 6
- course\_convert, 8
- course\_convert\_DF, 9, 37
- course\_convert\_df (course\_convert\_DF), 9
  
- discard\_errors, 10
- dive\_place, 11
- draw\_bracket, 11
  
- event\_parse, 13
- event\_parse\_ISL, 13
  
- fill\_down, 14
- fill\_left, 14
- fold, 15
- format\_results, 15
  
- get\_mode, 16
  
- heat\_parse\_omega, 17
- hy3\_parse, 17, 19, 20
- hy3\_places, 18
- hy3\_times, 19
  
- interleave\_results, 20
- is\_link\_broken, 20
  
- King200Breast, 21
  
- lines\_sort, 21
- list\_transform, 22
  
- mmss\_format, 22, 29
  
- name\_reorder, 23
  
- reaction\_times\_parse, 24
- Read\_Results, 24
- read\_results, 16, 18, 28, 30, 31, 38, 39, 41, 43
- read\_results (Read\_Results), 24
- read\_results\_flag, 25
- results\_score, 26
  
- samms\_parse, 27
- sec\_format, 23, 28
- sec\_format\_helper, 29
- splits\_parse, 30, 33
- splits\_parse\_ISL, 30
- splits\_parse\_omega\_relays, 31
- splits\_reform, 32
- splits\_rename\_omega, 32
- splits\_to\_cumulative, 33, 35
- splits\_to\_cumulative\_helper\_recalc, 34
- splits\_to\_lap, 33, 35
- splits\_to\_lap\_helper\_recalc, 36
- Swim\_Parse, 37
- swim\_parse, 3, 13, 16, 18, 24, 25, 30
- swim\_parse (Swim\_Parse), 37
- Swim\_Parse\_ISL (swim\_parse\_ISL), 39
- swim\_parse\_ISL, 14, 31, 39
- swim\_parse\_old, 40
- swim\_parse\_omega, 17, 31, 33, 41
- swim\_place, 43
- SwimmeR-deprecated, 37
  
- tie\_rescore, 43