Package ‘SwimmeR’

March 24, 2023

Title Data Import, Cleaning, and Conversions for Swimming Results
Version 0.14.2
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 7.2.2
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 2023-03-24 13:20:02 UTC
R topics documented:

add_event_dummy_row .................................................. 4
add_row_numbers ............................................................. 4
age_format ................................................................. 5
age_format_helper .......................................................... 6
clean_events ................................................................. 6
coalesce_many .............................................................. 7
coalesce_many_helper ..................................................... 7
collect Relay swimmers .................................................. 8
collect Relay swimmers_old ............................................. 8
collect Relay swimmers_omega ......................................... 9
collect Relay swimmers_splash ......................................... 10
correct_split_distance ................................................... 10
correct_split_distance_helper ......................................... 11
course_convert ............................................................. 12
course_convert_DF .......................................................... 13
course_convert_helper .................................................... 14
discard_errors .............................................................. 15
dive_place ................................................................. 16
draw_bracket ............................................................... 17
event_parse ................................................................. 18
event_parse_ISL ............................................................. 19
fill_down ................................................................. 19
fill_left .......................................................... 20
fold .............................................................. 20
format_results ............................................................. 21
generate_row_to_add .................................................... 21
get_mode ................................................................. 22
heat_parse_omega .......................................................... 23
hy3_parse ................................................................. 23
hy3_places ............................................................... 24
hy3_times ................................................................. 25
hytek_clean_strings ..................................................... 25
hytek_length_3_DQ_sort ................................................ 26
hytek_length_3_sort ....................................................... 26
hytek_length_4_DQ_sort ................................................ 26
hytek_length_4_sort ....................................................... 27
hytek_length_5_sort ....................................................... 28
hytek_length_6_sort ....................................................... 28
hytek_length_7_sort ....................................................... 29
hytek_length_8_sort ....................................................... 30
hytek_length_9_sort ....................................................... 30
interleave_results ........................................................ 31
is_link_broken ............................................................ 32
King200Breast ............................................................... 32
lines_sort ................................................................. 33
list_breaker ............................................................... 33
R topics documented:

- list_to_list_names ................................................................. 34
- list_transform ................................................................. 34
- make_lineup ................................................................. 35
- make_lineup_helper ................................................................. 36
- make_lineup_helper_2 ................................................................. 37
- mmss_format ................................................................. 38
- name_reorder ................................................................. 39
- na_pad ................................................................. 40
- place ................................................................. 40
- reaction_times_parse ................................................................. 42
- read_hy3 ................................................................. 43
- read_pdf ................................................................. 43
- Read_Results ................................................................. 44
- read_results_flag ................................................................. 45
- replacement_entries ................................................................. 45
- results_score ................................................................. 46
- sec_format ................................................................. 47
- sec_format_helper ................................................................. 48
- splash_clean_strings ................................................................. 49
- splash_collect_splits ................................................................. 50
- splash_determine.indent_length ................................................................. 50
- splash_length_10_sort ................................................................. 51
- splash_length_11_sort ................................................................. 51
- splash_length_12_sort ................................................................. 52
- splash_length_4_sort ................................................................. 52
- splash_length_5_sort ................................................................. 53
- splash_length_6_sort ................................................................. 54
- splash_length_7_sort ................................................................. 54
- splash_length_8_sort ................................................................. 55
- splash_length_9_sort ................................................................. 56
- splits_parse ................................................................. 56
- splits_parse_ISL ................................................................. 57
- splits_parse_omega_relays ................................................................. 58
- splits_parse_splash ................................................................. 58
- splits_parse_splash_helper_1 ................................................................. 59
- splits_parse_splash_helper_2 ................................................................. 59
- splits_parse_splash_relays ................................................................. 60
- splits_reform ................................................................. 61
- splits_rename_omega ................................................................. 61
- splits_to_cumulative ................................................................. 62
- splits_to_cumulative_helper_recalc ................................................................. 63
- splits_to_lap ................................................................. 64
- splits_to_lap_helper_recalc ................................................................. 65
- SwimmeR-defunct ................................................................. 66
- SwimmeR-deprecated ................................................................. 66
- Swim_Parse ................................................................. 66
- swim_parse_hytek ................................................................. 68

...
add_row_numbers

add_event_dummy_row  Add dummy entry rows

Description
If a team does not have a full compliment, defined by max_entries, of athletes in a given event then dummy rows containing blank entries need to be added to that event

Usage
add_event_dummy_row(x)

Arguments
x  a list of data frames containing event results that need dummy entries added

Value
returns a list of data frames each with a dummy entry row added

add_row_numbers  Add row numbers to raw results

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

Usage
add_row_numbers(text)

Arguments
text  output from read_results
age_format

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`

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

```r
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**

```r
age_format_helper(x)
```

**Arguments**

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

---

clean_events

*Regularizes event names*

---

**Description**

XXX

**Usage**

```r
clean_events(x)
```

**Arguments**

- `x` a character vector of event names

**Value**

a character vector of event names with naming conventions enforced to regularize event names
**coalesce_many**

*Combined paired sets of columns following a join operation*

**Description**

Combined paired sets of columns following a join operation

**Usage**

`coalesce_many(df)`

**Arguments**

- `df`: a data frame following a join and thereby containing paired columns of the form `Col_1.x, Col_1.y`

**Value**

returns a data frame with all sets of paired columns combined into single columns and named as, for example, `Col_1, Col_2` etc.

**See Also**

`coalesce_many` runs inside `swim_parse_splash`

---

**coalesce_many_helper**

*Combined paired sets of columns following a join operation*

**Description**

This function is intended to be mapped over a sequence `i` inside the function `coalesce_many`

**Usage**

`coalesce_many_helper(df, new_split_names, i)`

**Arguments**

- `df`: a data frame following a join and thereby containing paired columns of the form `Col_1.x, Col_1.y`
- `new_split_names`: a list of desired column names, e.g. `Col_1, Col_2`
- `i`: a number between 1 and the length of `new_split_names`
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
Depreciated version associated with depreciated version of swim_parse_old

Usage
collect_relay_swimmers_old(x, typo_2 = typo, replacement_2 = replacement)
**collect_relay_swimmers_omega**

**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`
**collect_relay_swimmers_splash**

*Collects relay swimmers as a data frame within swim_parse_splash*

**Description**

Collects relay swimmers as a data frame within swim_parse_splash

**Usage**

```r
collect_relay_swimmers_splash(x, relay_indent = Indent_Length)
```

**Arguments**

- `x`: output from `read_results` followed by `add_row_numbers`
- `relay_indent`: the number of spaces relay swimmer lines are indented compared to regular swimmer lines

**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_splash

---

**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**

```r
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")
)

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
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

• Time
• Course
• Course_To
• Event
• Time_Converted_sec
• Time_Converted_mmss

. 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.
course_convert_DF

References
Uses the USA swimming age group method described here: https://support.gomotionapp.com/en/articles/6457476-how-to-perform-course-conversion-factoring-of-times

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 - defunct

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
course_convert_helper

Note

Relays are not presently supported.

References

Uses the USA swimming age group method described here https://support.gomotionapp.com/en/articles/6457476-how-to-perform-course-conversion-factoring-of-times

---

**course_convert_helper  Swimming Course Convertor Helper**

Description

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

Usage

course_convert_helper(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

• Time
• Course
• Course_To
• Event
• Time_Converted_sec
• Time_Converted_mmss

. 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
discard_errors

- Event
- Time_Converted_sec
- Time_Converted_mmss

dependent on the value of verbose

See Also

course_convert_helper is a helper function inside course_convert

discard_errors Discards elements of list that have an error value from purrr::safely.

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)
**dive_place**

*Adds places to diving results*

**Description**

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

**Usage**

```r
dive_place(
    df,
    score_col = Finals,
    max_place = NULL,
    keep_nonscoring = TRUE,
    verbose = TRUE
)
```

**Arguments**

- `df`: a data frame with results from `swim_parse`, including only diving results (not swimming)
- `score_col`: the name of a column in `df` containing scores on which to place (order) performances
- `max_place`: highest place value that scores are placed
- `keep_nonscoring`: are athletes in places greater than `max_place` be retained in the data frame. Either TRUE or FALSE
- `verbose`: should warning messages be posted. Default is TRUE and should rarely be changed.

**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

Creates a bracket for tournaments involving 5 to 64 teams, single elimination

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

teamsa list of teams, ordered by desired seed, to place in bracket. Must be between 5 and 64 inclusive. Teams must have unique names

titlebracket title

text_sizenumber passed to cex in plotting

round_twoa list of teams advancing to the second round (need not be in order)

round_threea list of teams advancing to the third round (need not be in order)

round_foura list of teams advancing to the fourth round (need not be in order)

round_fivea list of teams advancing to the fifth round (need not be in order)

round_sixa list of teams advancing to the fifth round (need not be in order)

championthe 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

```r
## 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  
*Pulls out event labels from text*

### 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

```r
event_parse(text)
```

### arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>output from <code>read_results</code> followed by <code>add_row_numbers</code></td>
</tr>
</tbody>
</table>

### 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

Pulls out event labels from text

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**

```r
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

---

fold  

*Fold a vector onto itself*

**Description**

Fold a vector onto itself.

**Usage**

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

**Arguments**

- `x` a vector
- `block.size` 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.
**format_results**

Formats data for analysis within `swim_parse`

**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, by moving over Prelims. 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`

---

**generate_row_to_add**

Create a one-line data frame containing an entry to be appended to an in-progress data frame of all entries

**Description**

Create a one-line data frame containing an entry to be appended to an in-progress data frame of all entries

**Usage**

`generate_row_to_add(df_helper_2, e_rank_helper_2, k, e_helper)`

**Arguments**

- `df_helper_2` : a master data frame of athlete ranks by event
- `e_rank_helper_2` : a data frame of candidate athlete entries to add to a given event
- `k` : an integer denoting which element of `e_rank_helper` is under evaluation for addition. Should be 1, 2, 3 or 4 depending on the minimum number of entries
- `e_helper` : the event for which entries are being evaluated
get_mode

Value

a one row data frame containing an improved entry

---

**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

```r
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

```r
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)
numbs <- c(1, 1, 2, 2, 2, 3, NA)
get_mode(numbs, 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 `read_results`
- **avoid**: a list of strings. Rows in x 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**: 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

Value

returns a data frame with columns Name, Place, Age, Team, Prelims, Finals, & Event. May also contain Seed_Time, USA_ID, and/or Birthdate. Note all swims will have a Finals, 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**

```r
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"
hy3_times

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

hytek_clean_strings

Cleans input strings

Description

Cleans input from read_results is passed to hytek_swim_parse to remove unneded characters and otherwise set it up for sorting. Input is in the form of character strings

Usage

hytek_clean_strings(x, time_score_string = Time_Score_String)
Arguments

x a list of character strings
time_score_string a regex string for matching results (times and scores) but not special strings like DQ

Value

returns a list of character strings that have been cleaned in preparation for parsing/sorting

#’ @seealso hytek_clean_strings runs inside of hytek_parse_splash

---

**hytek_length_3_DQ_sort**  
Sort data in DQ lists of length 3 within hytek_swim_parse

---

**Description**

Sort data in DQ lists of length 3 within hytek_swim_parse

**Usage**

hytek_length_3_DQ_sort(x)

**Arguments**

x a list of lists containing DQ results with all sub-lists having length 3 strings

**Value**

returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

---

**hytek_length_3_sort**  
Sort data in lists of length 3 within hytek_swim_parse

---

**Description**

Sort data in lists of length 3 within hytek_swim_parse

**Usage**

hytek_length_3_sort(x)

**Arguments**

x a list of lists with all sub-lists having length 3 strings
Value
returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

hytek_length_4_DQ_sort
Sort data in DQ lists of length 4 within hytek_swim_parse

Description
Sort data in DQ lists of length 4 within hytek_swim_parse

Usage
hytek_length_4_DQ_sort(x)

Arguments
x a list of lists containing DQ results with all sub-lists having length 4 strings

Value
returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

hytek_length_4_sort
Sort data in lists of length 4 within hytek_swim_parse

Description
Sort data in lists of length 4 within hytek_swim_parse

Usage
hytek_length_4_sort(x, time_score_specials_string = Time_Score_Specials_String)

Arguments
x a list of lists with all sub-lists having length 4 strings
time_score_specials_string a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

Value
returns a formatted data frame to be combined with others to make the output of hytek_swim_parse
\begin{document}

\begin{Verbatim}
hytek_length_5_sort \quad Sort data in lists of length 5 within hytek_swim_parse
\end{Verbatim}

\textbf{Description}
Sort data in lists of length 5 within hytek_swim_parse

\textbf{Usage}
\begin{verbatim}
hytek_length_5_sort(
    x,
    name_string = Name_String,
    age_string = Age_String,
    para_string = Para_String,
    time_score_specials_string = Time_Score_Specials_String
)
\end{verbatim}

\textbf{Arguments}
\begin{itemize}
    \item \textit{x} \quad a list of lists with all sub-lists having length 5 strings
    \item \textit{name_string} \quad a regex string for matching athlete names
    \item \textit{age_string} \quad a regex string for matching athlete ages
    \item \textit{para_string} \quad a regex string for matching Paralympics classification strings
    \item \textit{time_score_specials_string} \quad a regex string for matching results - i.e. times, diving scores and 'specials' like DQ
\end{itemize}

\textbf{Value}
returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

\begin{Verbatim}
hytek_length_6_sort \quad Sort data in lists of length 6 within hytek_swim_parse
\end{Verbatim}

\textbf{Description}
Sort data in lists of length 6 within hytek_swim_parse

\textbf{Usage}
\begin{verbatim}
hytek_length_6_sort(
    x,
    name_string = Name_String,
    age_string = Age_String,
    para_string = Para_String,
    time_score_specials_string = Time_Score_Specials_String
)
\end{verbatim}

\end{document}
Arguments

- **x**: a list of lists with all sub-lists having length 6 strings
- **name_string**: a regex string for matching athlete names
- **age_string**: a regex string for matching athlete ages
- **para_string**: a regex string for matching Paralympics classification strings
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

Value

returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

Description

Sort data in lists of length 7 within hytek_swim_parse

Usage

```r
hytek_length_7_sort(
  x,
  brit_id_string = Brit_ID_String,
  para_string = Para_String,
  age_string = Age_String,
  time_score_specials_string = Time_Score_Specials_String
)
```

Arguments

- **x**: a list of lists with all sub-lists having length 7
- **brit_id_string**: a regex string for matching British swimming IDs
- **para_string**: a regex string for matching Paralympics classification strings
- **age_string**: a regex string for matching athlete ages
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

Value

returns a formatted data frame to be combined with others to make the output of hytek_swim_parse
**hytek_length_8_sort**  
*Sort data in lists of length 8 within hytek_swim_parse*

**Description**

Sort data in lists of length 8 within hytek_swim_parse

**Usage**

```r
hytek_length_8_sort(
  x,
  brit_id_string = Brit_ID_String,
  para_string = Para_String,
  age_string = Age_String,
  time_score_specials_string = Time_Score_Specials_String
)
```

**Arguments**

- **x**: a list of lists with all sub-lists having length 8
- **brit_id_string**: a regex string for matching British swimming IDs
- **para_string**: a regex string for matching Paralympics classification strings
- **age_string**: a regex string for matching athlete ages
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**

returns a formatted data frame to be combined with others to make the output of hytek_swim_parse

**hytek_length_9_sort**  
*Sort data in lists of length 9 within hytek_swim_parse*

**Description**

Sort data in lists of length 9 within hytek_swim_parse

**Usage**

```r
hytek_length_9_sort(
  x,
  brit_id_string = Brit_ID_String,
  para_string = Para_String,
  age_string = Age_String,
  time_score_specials_string = Time_Score_Specials_String
)
```
interleave_results

Arguments

- **x**: a list of lists with all sub-lists having length 9
- **brit_id_string**: a regex string for matching British swimming IDs
- **para_string**: a regex string for matching Paralympics classification strings
- **age_string**: a regex string for matching athlete ages
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

Value

returns a formatted data frame to be combined with others to make the output of hytek_swim_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  \hspace{1em} \textit{Determines if a link is valid}

\section*{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

\section*{Usage}

\begin{verbatim}
is_link_broken(link_to_test, duration = 1)
\end{verbatim}

\section*{Arguments}

\begin{itemize}
  \item \texttt{link_to_test} a link
  \item \texttt{duration} the lowest row number
\end{itemize}

\section*{Value}

\begin{itemize}
  \item TRUE if the link works, FALSE if it fails
\end{itemize}

\begin{verbatim}
King200Breast  \hspace{1em} Results for Lilly King's 200 Breaststrokes
\end{verbatim}

\section*{Description}

Lilly King’s 200 Breaststroke swims from her NCAA career

\section*{Usage}

\begin{verbatim}
data(King200Breast)
\end{verbatim}

\section*{Format}

An object of class "data.frame"

\section*{Source}

NCAA Times Database
**lines_sort**

*Sorts and collects lines by performance and row number*

**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**

```r
lines_sort(x, min_row = minimum_row, to_wide = TRUE)
```

**Arguments**

- `x`: a list of character strings including performances, with tow numbers added by `add_row_numbers`
- `min_row`: the lowest row number
- `to_wide`: should the data frame `x` be converted to wide format? Default is `TRUE` as used in Hytek and Omega results. Use `FALSE` in Splash results

**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_breaker**

*Breaks out lists of lists by sub-list length*

**Description**

XXXXXX

**Usage**

```r
list_breaker(x, len)
```

**Arguments**

- `x`: a list of lists, with at least some sub-lists having length `len`
- `len`: an numeric value for the length of sub-lists that `list_breaker` should break out. Must be a whole number.
Value
returns a list of lists, with all sub-lists having length 1en

---
lst_to_lst_names
Initialize a named list of lists

Description
Convert a single list to a list of lists, with the names of the lists taken from the original list, lst_of_names. The new lists will all have a single value, initialized as value.

Usage
lst_to_lst_names(lst_of_names, value = 0)

Arguments
- lst_of_names: a list of values, likely strings, to be the names of sub-lists in a new list of lists
- value: a value to initialize elements of all sub-lists to. Defaults to 0. If value has multiple elements those elements will become sub-list elements

Value
returns a list of lists with sub-list names from lst_of_names and first elements from value. Used inside determine_entries

---
lst_transform
Transform list of lists into data frame

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
lst_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
**make_lineup**

**See Also**

- list_transform is a helper function used inside of swim_parse, swim_parse_ISL, event_parse and event_parse_ISL

---

**make_lineup**

*Determine optimal entries against a given opponent lineup*

---

**Description**

Determine optimal entries against a given opponent lineup

**Usage**

```r
make_lineup(
  df,
  op_df,
  point_values,
  result_col,
  events = NULL,
  max_entries = NULL,
  max_ind_entries = NULL
)
```

**Arguments**

- **df**
  - a data frame of times for the team to be entered. Must contain column Event with the same event naming convention as op_df, a column with name matching result_col containing times or diving scores, and a column called Name containing athlete names

- **op_df**
  - a data frame containing the opponent lineup. Must contain column Event with the same event naming convention as df, a column with name matching result_col containing times or diving scores, and a column called Name containing athlete names

- **point_values**
  - either a recognized string or a list of numeric values containing the points awarded by place. Recognized strings are "hs_four_lane", "hs_six_lane", "ncaa_six_lane"

- **result_col**
  - the name of a column, present in both df and op_df that contains times and/or diving scores

- **events**
  - a list of events. If no list is entered then events will be taken from unique(op_df$Event)

- **max_entries**
  - the number of entries a team is permitted per race. usually half the number of lanes in the competition pool

- **max_ind_entries**
  - the number of individual events a given athlete may enter

**Value**

- a data frame of optimal entries based on df and op_df
**make_lineup_helper**

*Determine optimal entries against a given opponent lineup*

**Description**

Matches athletes into events. Each event is filled by the least capable (slowest) swimmer who can win or place in that event. For example if Team A has six breaststrokes at 57.00, 58.00, 59.00 and three 1:00.00s and Team B has three breaststrokes, all 1:01.00 then Team A’s entries will be the three 1:00.00s because they’re sufficient to win.

**Usage**

```r
make_lineup_helper(
  i,
  df_helper,
  op_df_helper,
  end_seq,
  max_ind_entries_helper = 2,
  result_col_helper = result_col
)
```

**Arguments**

- `i`: a sequential list of numbers incremented by 1. Used to index function.
- `df_helper`: a data frame of times for the team to be entered. Must contain column `Event` with the same event naming convention as `op_df`, a column with name matching `result_col` containing times or diving scores, and a column called `Name` containing athlete names.
- `op_df_helper`: a data frame containing the opponent lineup. Must contain column `Event` with the same event naming convention as `df`, a column with name matching `result_col` containing times or diving scores, and a column called `Name` containing athlete names.
- `end_seq`: how many events score.
- `max_ind_entries_helper`: a numeric value denoting the maximum number of individual events that may be entered by a single athlete.
- `result_col_helper`: name of column with results in it.

**Value**

a data frame containing athletes entered into events
make_lineup_helper_2

Assign overpowered entries

Description

Matches athletes into events again, this time vs. the output of make_lineup_helper. For example if Team A has six breaststrokers at 57.00, 58.00, 59.00 and three 1:00.00s and Team B has three breaststrokers, all 1:01.00 then following make_lineup_helper Team A’s entries will be the three 1:00.00s because they’re sufficient to win.

Usage

make_lineup_helper_2(
  i,
  df_helper,
  in_progress_entries_df,
  events_competed_helper = Events_Competed,
  max_entries_helper = max_entries,
  max_ind_entries_helper = max_ind_entries
)

Arguments

i a sequential list of numbers incremented by 1. Used to index function.
df_helper a data frame of all times to be entered for a given team. Must contain column Event with the same event naming convention as op_df, a column with name matching result_col containing times or diving scores, and a column called Name containing athlete names
in_progress_entries_df a data frame containing the output of make_lineup_helper, which is the minimum power set of entries
events_competed_helper a list of lists containing all the events a given athlete is competing in. Sub-lists are named with the athlete name.
max_entries_helper a numeric value denoting the maximum number of athletes a team may enter in a given event
max_ind_entries_helper a numeric value denoting the maximum number of individual events that may be entered by a single athlete

Details

Here though Team A’s three 1:00.00s will be replaced by their 57.00, 58.00 and 59.00 breaststrokers. These entries are "overpowered" but better reflect an actual set of entries. Not using make_lineup_helper_2 often results in a team’s best athletes not competing
mmss_format

Formatting seconds as mm:ss.hh

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)
mms_format(200.95)
mms_format(59.47)
mms_format(c(95.37, 200.95, 59.47, NA))
name_reorder

Orders all names as "Firstname Lastname"

Description
Names are sometimes listed as Firstname Lastname, and sometimes as Lastname, Firstname. The name_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 swim_parse containing a column called Name 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 First_Name and Last_Name extracted as best as possible from Name

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"))
## na_pad

**Pads shorter lists in a list-of-lists with NAs such that all lists are the same length**

### Description

Adds NA values to the end of each list in a list of lists such that they all become the length of the longest list. The longest list will not have any NAs added to it.

### Usage

```r
na_pad(x, y)
```

### Arguments

- **x**: a list of lists, with sub-lists having different lengths
- **y**: a list of the number of NA values to append to each sub-list

### Value

a list of lists with each sub-list the same length

## place

**Add places to results**

### Description

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

### Usage

```r
place(
  df,
  result_col = Finals,
  max_place = NULL,
  event_type = "ind",
  max_relays_per_team = 1,
  keep_nonscoring = TRUE,
  verbose = TRUE
)
```
Arguments

- **df**: a data frame with results from `swim_parse`, including swimming and/or diving results. `df` must contain a column called `Event`.
- **result_col**: the name of a column in `df` containing times and/or scores on which to place (order) performances. Default is `Finals`.
- **max_place**: highest place value that scores
- **event_type**: either "ind" for individual or "relay" for relays
- **max_relays_per_team**: an integer value denoting the number of relays a team may score (usually 1)
- **keep_nonscoring**: are athletes in places greater than `max_place` be retained in the data frame. Either TRUE or FALSE
- **verbose**: should warning messages be posted. Default is TRUE and should rarely be changed.

Value

a data frame modified so that places have been appended based on swimming time and/or diving score

See Also

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

Examples

def <- data.frame( Place = c(1, 1, 1, 1, 1), Name = c("Sally Swimfast", "Bonnie Bubbles", "Kylie Kicker", "Riley Ripit", "Nathan Nosplash", "Tim Tuck"), Team = c("KVAC", "UBAM", "MERC", "Upstate Diving", "Nickel City Splash", "Finger Lakes Diving"), Event = c("Women 200 Freestyle", "Boys 1 mtr Diving"), Prelims = c("2:00.00", "1:59.99", "2:01.50", "300.00", "305.00", "200.00"), Finals = c("1:58.00", "1:59.50", "2:00.50", "310.00", "307.00", "220.00"), Meet = c("Summer 2021", "Fall 2020", "Champs 2020", "Regional Champs 2021", "Other Regional Champs 2021", "City Champs 2021" ))

def %>%
  place() %>%
  dplyr::arrange(Event)

def %>%
  place(result_col = Prelims) %>%
  dplyr::arrange(Event)

def %>%
  place(result_col = "Prelims") %>%
  dplyr::arrange(Event)
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_htm  Read in html files of swimming results

Description
Read in html files of swimming results

Usage
read_htm(x, node_helper)

Arguments
x  an .html, .htm or .aspx location containing swimming results. Must be formatted in a "normal" fashion - see vignette
node_helper  receives node from read_results

Value
returns a list of results, with "read_results_flag" added as the first element of the list
**read_hy3**

*Read in hy3 files of swimming results*

**Description**

Read in hy3 files of swimming results

**Usage**

```r
read_hy3(x)
```

**Arguments**

- `x` an unzipped hy3 file containing swimming results. Must be formatted in a "normal" fashion - see vignette

**Value**

returns a list of results, with "read_results_flag" added as the first element of the list

---

**read_pdf**

*Read in pdf files of swimming results*

**Description**

Based on pdftools, this function can be temperamental

**Usage**

```r
read_pdf(x)
```

**Arguments**

- `x` a .pdf or .aspx location containing swimming results. Must be formatted in a "normal" fashion - see vignette

**Value**

returns a list of results, with "read_results_flag" added as the first element of the list
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, url or Hytek .hy3 file 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**

```r
## Not run:
link <- "http://www.nyhsswim.com/Results/Boys/2008/NYS/Single.htm", node = "pre"
read_results(link)
## End(Not run)
```
**read_results_flag**

*used to indicate that results have been read in with read_results prior to being parsed by swim_parse*

**Description**

Used to insure that read_results has been run on a data source prior to running swim_parse.

**Usage**

```r
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

---

**replacement_entries**

*Replaces superseded rows*

**Description**

Replaces superseded rows.

**Usage**

```r
replacement_entries(x, j_helper, row_to_add_replacement, e_df_replacement)
```

**Arguments**

- `x`: a data frame of entries, either df_helper_2 or Entries
- `j_helper`: an integer denoting which element of `e_df_replacement` is under test for removal. Should be 1, 2, 3 or 4 depending on the minimum number of entries
- `row_to_add_replacement`: a row containing an improved entry that should be added to `x`
- `e_df_replacement`: a data frame of entries that may be replaced

**Value**

a data frame containing entries updated to include new rows from `row_to_add_replacement` and to not contain rows from `e_df_replacement`, based on `j_helper`
results_score  

*Scores a swim meet*

**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 = NULL,  
  meet_type = c("timed_finals", "prelims_finals"),  
  lanes = c(4, 6, 8, 10),  
  scoring_heats = c(1, 2, 3),  
  point_values,  
  max_relays_per_team = 1
)
```

**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**: Either a list of point values for each scoring place or one of the following recognized strings: "hs_four_lane", "hs_six_lane", "ncaa_six_lane", "championship_8_lane_2_heat" or "championship_8_lane_3_heat"
- **max_relays_per_team**: the number of relays a team is allowed to score (usually 1)

**Value**

results with point values in a column called Points

**Examples**

```r
## Not run:
file <- system.file("extdata", "BigTen_WSWM_2018.pdf", package = "SwimmeR")
BigTenRaw <- read_results(file)
```
BigTen <- swim_parse(
    BigTenRaw,
    typo = c(
        "^\s{1,}\*",    # Remove asterisks
        "^\s{1,}(\d{1,2})\s{2,}",  # Remove numbers with spaces
        "University\s{1,}of\s{1,}\s{1,}\s{1,}\s{1,}\s{1,}\s{1,}\s{1,}University",
        "\s{1,}University",
        "SR\s{2,}",  # Remove spaces before 'SR'
        "JR\s{2,}",  # Remove spaces before 'JR'
        "SO\s{2,}",  # Remove spaces before 'SO'
        "FR\s{2,}",  # Remove spaces before 'FR'
    ),
    replacement = c(" ",  # Replace all removed strings with spaces
        " \1 ",
        "",  # Replace all removed numbers with empty strings
        "",  # Replace all removed numbers with empty strings
        "SR ",
        "JR ",
        "SO ",
        "FR "),
    avoid = c("BigTen", "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)
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", ":55.23"))

sec_format_helper

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

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)
**splash_clean_strings**  *Cleans input strings*

**Description**

Cleans input from `read_results` is passed to `splash_swim_parse` to remove unnecessary characters and otherwise set it up for sorting. Input is in the form of character strings.

**Usage**

```r
splash_clean_strings(
  x,
  indent_length = Indent_Length,
  time_score_string = Time_Score_String,
  record_string = Record_String,
  header_string = Header_String,
  sponsorship_string = Sponsorship_String,
  reaction_string = Reaction_String,
  rule_string = Rule_String
)
```

**Arguments**

- `x` a list of character strings
- `indent_length` a numeric value denoting the number of spaces some results are indented by. `indent_length` is determined by `splash_determine_indent_length`. Must be a whole number.
- `time_score_string` a regex string for matching results (times and scores) but not special strings like DQ
- `record_string` a regex string for matching denoted records, rather than results
- `header_string` a regex string from matching splash headers/footers included in result documents
- `sponsorship_string` a regex string for matching sponsorship text within result documents
- `reaction_string` a regex string for matching reaction times
- `rule_string` a regex string for matching rule text e.g. 'Rule 4.24' that sometimes accompanies DQs

**Value**

returns a list of character strings that have been cleaned in preparation for parsing/sorting

`#* @seealso splash_clean_strings runs inside of swim_parse_splash`
**splash_collect_splits**  
*Collects Splash format splits*

**Description**  
Collects splits and breaks them into a distance and a time, with a corresponding row number

**Usage**  
splash_collect_splits(df)

**Arguments**  
- **df**  
a data frame containing two columns, V1 is row numbers and Dummy as a string combining split distance and split time

**Value**  
a data frame with three columns, V1, Split_Distance and Split

**splash_determine_indent_length**  
*Determines indent length for data within swim_parse_splash*

**Description**  
In Splash results there are two line types that are of interest and don’t begin with either a place or a special string (DNS, DSQ etc.). These are ties and relays swimmers. Relay swimmers are indented further than ties. This function determines the number of spaces, called indent length, prior to a tie row, plus a pad of four spaces.

**Usage**  
splash_determine_indent_length(x, time_score_string)

**Arguments**  
- **x**  
output from read_results followed by add_row_numbers
- **time_score_string**  
a regular expression as a string that describes swimming times and diving scores

**Value**  
returns a number indicating the number of spaces preceding an athlete’s name in a tie row

**See Also**  
splash_determine_indent_length runs inside of swim_parse_splash
**splash_length_10_sort**

_sort data in lists of length 10 within splash_swim_parse_

---

**Description**

Sort data in lists of length 10 within splash_swim_parse

**Usage**

```r
splash_length_10_sort(
  x,
  time_score_string = Time_Score_String,
  time_score_specials_string = Time_Score_Specials_String
)
```

**Arguments**

- **x**: a list of lists with all sub-lists having length 10
- **time_score_string**: a regex string for matching results (times and scores) but not special strings like DQ
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**

returns a formatted data frame to be combined with others to make the output of splash_swim_parse

---

**splash_length_11_sort**

_sort data in lists of length 11 within splash_swim_parse_

---

**Description**

Sort data in lists of length 11 within splash_swim_parse

**Usage**

```r
splash_length_11_sort(
  x,
  time_score_specials_string = Time_Score_Specials_String
)
```
Arguments

\[ x \] a list of lists with all sub-lists having length 11

\[ \text{time_score_specials_string} \]

a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

Value

returns a formatted data frame to be combined with others to make the output of splash_swim_parse

---

**splash_length_12_sort**  Sort data in lists of length 12 within splash_swim_parse

---

**Description**

Sort data in lists of length 12 within splash_swim_parse

**Usage**

```
splash_length_12_sort(x)
```

**Arguments**

\[ x \] a list of lists with all sub-lists having length 12

**Value**

returns a formatted data frame to be combined with others to make the output of splash_swim_parse

---

**splash_length_4_sort**  Sort data in lists of length 4 within splash_swim_parse

---

**Description**

Sort data in lists of length 4 within splash_swim_parse

**Usage**

```
splash_length_4_sort(
  x,
  name_string = Name_String,
  time_score_specials_string = Time_Score_Specials_String
)
```
**splash_length_5_sort**

**Arguments**

- **x**  
  A list of lists with all sub-lists having length 4
- **name_string**  
  A regex string for matching athlete names
- **time_score_specials_string**  
  A regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**

Returns a formatted data frame to be combined with others to make the output of splash_swim_parse.

---

**splash_length_5_sort  Sort data in lists of length 5 within splash_swim_parse**

**Description**

Sort data in lists of length 5 within splash_swim_parse.

**Usage**

```r
splash_length_5_sort(
  x,
  name_string = Name_String,
  time_score_specials_string = Time_Score_Specials_String
)
```

**Arguments**

- **x**  
  A list of lists with all sub-lists having length 5
- **name_string**  
  A regex string for matching athlete names
- **time_score_specials_string**  
  A regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**

Returns a formatted data frame to be combined with others to make the output of splash_swim_parse.
**splash_length_6_sort**  
Sort data in lists of length 6 within splash_swim_parse

**Description**
Sort data in lists of length 6 within splash_swim_parse

**Usage**
splash_length_6_sort(
  x,
  time_score_specials_string = Time_Score_Specials_String
)

**Arguments**
- **x**: a list of lists with all sub-lists having length 6
- **time_score_specials_string**: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**
returns a formatted data frame to be combined with others to make the output of splash_swim_parse

---

**splash_length_7_sort**  
Sort data in lists of length 7 within splash_swim_parse

**Description**
Sort data in lists of length 7 within splash_swim_parse

**Usage**
splash_length_7_sort(
  x,
  time_score_string = Time_Score_String,
  time_score_specials_string = Time_Score_Specials_String
)

**Value**
returns a formatted data frame to be combined with others to make the output of splash_swim_parse
**splash_length_8_sort**

**Arguments**

- `x`: a list of lists with all sub-lists having length 7
- `time_score_string`: a regex string for matching results (times and scores) but not special strings like DQ
- `time_score_specials_string`: a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**

returns a formatted data frame to be combined with others to make the output of `splash_swim_parse`
**splash_length_9_sort**  
Sort data in lists of length 9 within **spash_swim_parse**

**Description**
Sort data in lists of length 9 within **spash_swim_parse**

**Usage**

```r
splash_length_9_sort(
  x,
  heat_lane_string = Heat_Lane_String,
  time_score_string = Time_Score_String,
  time_score_specials_string = Time_Score_Specials_String
)
```

**Arguments**

- `x` a list of lists with all sub-lists having length 9
- `heat_lane_string` a regex string for matching heat-lane pairs
- `time_score_string` a regex string for matching results (times and scores) but not special strings like DQ
- `time_score_specials_string` a regex string for matching results - i.e. times, diving scores and 'specials' like DQ

**Value**
returns a formatted data frame to be combined with others to make the output of **splash_swim_parse**

---

**splits_parse**  
Collects splits within **swim_parse**

**Description**
Takes the output of **read_results** and, inside of **swim_parse**, extracts split times and associated row numbers

**Usage**

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

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 data frame with split times and row numbers

See Also

splits_parse runs inside swim_parse on the output of read_results with row numbers from add_row_numbers

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 data frame with split times and row numbers

See Also
splits_parse runs inside swim_parse_omega on the output of read_results with row numbers from add_row_numbers

---

splits_parse_splash

Collects splits within swim_parse_splash for Splash results

Description
Takes the output of read_results and, inside of swim_parse_splash, extracts split times and associated row numbers

Usage
splits_parse_splash(raw_results)

Arguments
raw_results output of read_results with row numbers appended by add_row_numbers

Value
returns a data frame with split times and row numbers
splits_parse_splash_helper_1

See Also
	splits_parse runs inside swim_parse_splash on the output of read_results with row numbers from add_row_numbers

---

splits_parse_splash_helper_1

*Produces data frames of splits within swim_parse_splash for Splash results*

---

Description

Converts strings of splits and row numbers into data frames with a row number column (V1) and a splits column (Split_XX)

Usage

splits_parse_splash_helper_1(data)

Arguments

data  
a list of lists containing splits and row numbers

Value

returns a data frame with split times and row numbers

See Also

splits_parse_splash_helper_1 runs inside splits_parse_splash

---

splits_parse_splash_helper_2

*Produces data frames of splits within swim_parse_splash for Splash results*

---

Description

Converts strings of splits and row numbers into data frames with a row number column (V1) and a splits column (Split_XX)

Usage

splits_parse_splash_helper_2(data, split_distances, i)
Arguments

data       a list of lists containing splits and row numbers
split_distances       a list of distances for splits, e.g. "50m", "100m"
i       a number between 1 and the length of split_distances

Value

returns a data frame with split times and row numbers

See Also

splits_parse_splash_helper_2 runs inside splits_parse_splash

splits_parse_splash_relays

Collects splits for relays within swim_parse_splash

Description

Takes the output of read_results and, inside of swim_parse_splash, extracts split times and associated row numbers

Usage

splits_parse_splash_relays(text, split_len = split_length_splash)

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 data frame with split times and row numbers

See Also

splits_parse runs inside swim_parse_splash on the output of read_results with row numbers from add_row_numbers
### splits_reform

Adds together splits and compares to listed finals time to see if they match.

**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**

```r
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.

### splits_rename_omega

Advances split names by one split_length

**Description**

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

**Usage**

```r
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 = 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
The `splits_to_cumulative()` function is a helper function for converting lap splits to cumulative splits. It takes a data frame containing splits in lap format, a list of values to iterate along, a list of columns containing splits, and a numeric value below which a split is taken to be lap. The function returns 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.

```r
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

Converts splits from cumulative to lap 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 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 = c("1:58.00", "1:59.00"), 
  Split_50 = c("28.00", "29.00"), 
  Split_100 = c("59.00", "60.00"), 
  Split_150 = c("1:31.00", "1:32.00"), 
  Split_200 = c("1:58.00", "1:59.00")

df %>%
splits_to_lap
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-defunct  Defunct functions in SwimmeR

Description

These functions have been made defunct (removed) from SwimmeR.

Details

- **course_convert_DF**: This function is defunct, and has been removed from SwimmeR. Instead please use course_convert(verbose = TRUE)

---

SwimmeR-deprecated  Deprecated functions in SwimmeR

Description

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

---

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 = NULL,
  typo = typo_default,
  replacement = replacement_default,
  format_results = TRUE,
  splits = FALSE,
  split_length = 50,
  relay_swimmers = FALSE
)
swim_parse(
    file,
    avoid = NULL,
    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 are 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 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, Finals, Points, Event & DQ. Note all swims will have a Finals, 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

```r
## 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)
```

```r
swim_parse_hytek

Formats Hytek style 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

```r
swim_parse_hytek(
  file_hytek,
  avoid_hytek = avoid,
  typo_hytek = typo,
  replacement_hytek = replacement,
  format_results = TRUE,
  splits = FALSE,
  split_length_hytek = split_length,
  relay_swimmers_hytek = relay_swimmers
)
```

Arguments

- `file_hytek` output from read_results
- `avoid_hytek` a list of strings. Rows in file_hytek containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to avoid_hytek. 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_hytek. avoid_hytek is handled before typo_hytek and replacement_hytek.
swim_parse_ISL

- **typo_hytek**
  - 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_hytek. Unexpected commas as also an issue, for example "Texas, University of" should be fixed using typo_hytek and replacement_hytek.

- **replacement_hytek**
  - A list of fixes for the strings in typo_hytek. Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to replacement_hytek to fix the issues described in typo_hytek.

- **format_results**
  - Should the results be formatted for analysis (special strings like "DQ" replaced with NA, Finals as definitive column)? Default is TRUE.

- **splits**
  - Either TRUE or the default, FALSE - should swim_parse attempt to include splits.

- **split_length_hytek**
  - 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_hytek**
  - Should names of relay swimmers be captured? Default is FALSE.

**Value**

Returns a data frame with columns Name, Place, Age, Team, Prelims, Finals, Points, Event & DQ. Note all swims will have a Finals, 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_Hytek must be run on the output of `read_results`.

---

### Description

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

### Usage

```r
swim_parse_ISL(file, splits = FALSE, relay_swimmers = FALSE)
Swim_Parse_ISL(file, splits = FALSE, relay_swimmers = FALSE)
```
Argumenst

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(
  splits = TRUE,
  relay_swimmers = TRUE)

## End(Not run)

swim_parse_old 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. 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

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.
typo a list of strings that are typos in the original results. swim_parse_old 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
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

```r
## 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)
```
swim_parse_omega  

Formats Omega style 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

```r
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` to fix the issues described in `typo_omega`
- `format_results` should the results be formatted for analysis (special strings like "DQ" replaced with NA, Finals 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, Finals, Points, Event & DQ. Note all swims will have a Finals, 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_parse_samms

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

swim_parse_samms(
    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.
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
should the data be formatted for analysis (special strings like "DQ" replaced with NA, Finals as definitive column)? Default is TRUE

returns a data frame with columns Name, Place, Age, Team, Prelims, Finals, Event & DQ. Note all swims will have a Finals, 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.

`swim_parse` must be run on the output of `read_results`

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

`swim_parse_splash(file_splash, avoid_splash = avoid, typo_splash = typo, replacement_splash = replacement, format_results = TRUE, splits = FALSE, split_length_splash = split_length, relay_swimmers_splash = relay_swimmers)`
Arguments

file_splash output from read_results
avoid_splash a list of strings. Rows in file_splash containing these strings will not be included. For example "Pool:", often used to label pool records, could be passed to avoid_splash. 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_splash. avoid_splash is handled before typo_splash and replacement_splash.
typo_splash 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_splash. Unexpected commas as also an issue, for example "Texas, University of" should be fixed using typo_splash and replacement_splash.
replacement_splash a list of fixes for the strings in typo_splash. Here one could pass "Central High School" (one space between "Central" and "High") and "Texas" to replacement_splash to fix the issues described in typo_splash
format_results should the results be formatted for analysis (special strings like "DQ" replaced with NA, Finals as definitive column)? Default is TRUE
splits either TRUE or the default, FALSE - should swim_parse attempt to include splits.
split_length_splash 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_splash should names of relay swimmers be captured? Default is FALSE

Value

returns a data frame with columns Name, Place, Age, Team, Prelims, Finals, Points, Event & DQ. Note all swims will have a Finals, 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_splash must be run on the output of read_results

---

swim_place Add places to swimming results

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,
  time_col = Finals,
  max_place = NULL,
  event_type = "ind",
  max_relays_per_team = 1,
  keep_nonscoring = TRUE,
  verbose = TRUE
)

Arguments

df a data frame with results from swim_parse, including only swimming results (not diving)
time_col the name of a column in df containing times on which to place (order) performances. Default is Finals
max_place highest place value that scores
event_type either "ind" for individual or "relay" for relays
max_relays_per_team an integer value denoting the number of relays a team may score (usually 1)
keep_nonscoring are athletes in places greater than max_place be retained in the data frame. Either TRUE or FALSE
verbose should warning messages be posted. Default is TRUE and should rarely be changed.

Value

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

See Also

case_study is a helper function used inside of results_score

Examples

df <- data.frame(Place = c(1, 1, 1),
  Name = c("Sally Swimfast", "Bonnie Bubbles", "Kylie Kicker"),
  Team = c("KVAC", "UBAM", "MERC"),
  Event = rep("Women 200 Freestyle", 3),
  Prelims = c("2:00.00", "1:59.99", "2:01.50"),
  Finals = c("1:58.00", "1:59.50", "2:00.50"),
  Meet = c("Summer 2021", "Fall 2020", "Champs 2020"))

df %>%
  swim_place()
tie_rescore

```r
df %>%
  swim_place(time_col = Prelims)

df %>%
  swim_place(time_col = "Prelims")
```

**tie_rescore**  
*Rescore to account for ties*

**Description**

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

**Usage**

```r
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`

---

**toptimes_parse_hytek**  
*Formats Hytek style swimming and diving Top Times reports 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) top times.
Usage

toptimes_parse_hytek(
    file_hytek_toptimes,
    avoid_hytek_toptimes = avoid,
    typo_hytek_toptimes = typo,
    replacement_hytek_toptimes = replacement
)

Arguments

file_hytek_toptimes
    output from read_results

avoid_hytek_toptimes
    a list of strings. Rows in file_hytek_toptimes containing these strings will
    not be included. For example "Pool:“, often used to label pool records, could
    be passed to avoid_hytek_toptimes. 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_hytek_toptimes. avoid_hytek_toptimes
    is handled before typo_hytek_toptimes and replacement_hytek_toptimes.

typo_hytek_toptimes
    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_hytek_toptimes. Unexpected commas as also an issue,
    for example "Texas, University of“ should be fixed using typo_hytek_toptimes
    and replacement_hytek_toptimes

replacement_hytek_toptimes
    a list of fixes for the strings in typo_hytek. Here one could pass "Central High
    School“ (one space between "Central" and "High“) and "Texas“ to replacement_hytek_toptimes
    fix the issues described in typo_hytek_toptimes

Value

returns a data frame with columns Rank, Result, Name, Age, Date Meet & Event. Top Times reports
do not designate Team.

See Also

toptimes_parse_hytek must be run on the output of read_results

undo_interleave

Undoes interleaving of lists

Description

If two lists have been interleaved this function will return the lists separated and then concatenated
**update_rank_helper**

Usage

undo_interleave(x)

**Arguments**

x  
a list to be un-interleaved

**Value**

a list comprising the interleaved components of x joined into one list

**Examples**

```r
l <- c("A", "D", "B", "E", "C", "F")
undo_interleave(l)
```

---

**update_rank_helper**  
Create a one-line data frame containing an entry to be appended to an in-progress data frame of all entries

**Description**

Create a one-line data frame containing an entry to be appended to an in-progress data frame of all entries

**Usage**

```r
update_rank_helper(
  rank_helper_2,
  e_rank_helper_2,
  k,
  e_helper,
  events_remaining_helper
)
```

**Arguments**

- rank_helper_2  
a master data frame of athlete ranks by event
- e_rank_helper_2  
a data frame of candidate athlete entries to add to a given event
- k  
an integer denoting which element of e_rank_helper is under evaluation for addition. Should be 1, 2, 3 or 4 depending on the minimum number of entries
- e_helper  
the event for which entries are being evaluated
- events_remaining_helper  
a data frame with two columns, Name and Events_Remaining
Value

a one row data frame containing an improved entry

---

%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, 32
  %\in\%(%\notin\%), 80
  %\notin\%, 80

add_event_dummy_row, 4
add_row_numbers, 4, 21, 57–60
age_format, 5
age_format_helper, 5, 6

clean_events, 6
coalessce_many, 7, 7, 8
coalessce_many_helper, 7
collect_relay_swimmers, 8
collect_relay_swimmers_old, 8
collect_relay_swimmers_omega, 9
collect_relay_swimmers_splash, 10
correct_split_distance, 10
correct_split_distance_helper, 11
correct_split_length
  (correct_split_distance), 10
course_convert, 12, 15
course_convert_DF, 13, 66
course_convert_df (course_convert_DF), 13
course_convert_helper, 14
discard_errors, 15
dive_place, 16
draw_bracket, 17

event_parse, 18
event_parse_ISL, 19

fill_down, 19
fill_left, 20
fold, 20
format_results, 21

generate_row_to_add, 21
generate_row_to_add, 21
get_mode, 22

heat_parse_omega, 23
hy3_parse, 23, 25, 31
hy3_places, 24
hy3_times, 25
hytek_clean_strings, 25
hytek_length_3_DQ_sort, 26
hytek_length_3_sort, 26
hytek_length_4_DQ_sort, 27
hytek_length_4_sort, 27
hytek_length_5_sort, 28
hytek_length_6_sort, 28
hytek_length_7_sort, 29
hytek_length_8_sort, 30
hytek_length_9_sort, 30

interleave_results, 31
is_link_broken, 32

King200Breast, 32

lines_sort, 33
list_breaker, 33
list_to_list_names, 34
list_transform, 34

make_lineup, 35
make_lineup_helper, 36
make_lineup_helper_2, 37
mmss_format, 38, 48

na_pad, 40
name_reorder, 39

place, 40

reaction_times_parse, 42
read_htm, 42
read_hy3, 43
read_pdf, 43
Read_Results, 44
read_results, 21, 24, 57–60, 67, 69–71, 73–75, 78
read_results (Read_RESULTS), 44
read_results_flag, 45
replacement_entries, 45
results_score, 46
sec_format, 38, 47
sec_format_helper, 48
splash_clean_strings, 49
splash_collect_splits, 50
splash_determine_indent_length, 50
splash_length_10_sort, 51
splash_length_11_sort, 51
splash_length_12_sort, 52
splash_length_4_sort, 52
splash_length_5_sort, 53
splash_length_6_sort, 54
splash_length_7_sort, 54
splash_length_8_sort, 55
splash_length_9_sort, 56
splits_parse, 56, 61
splits_parse_ISL, 57
splits_parse_omega_relays, 58
splits_parse_splash, 58, 59, 60
splits_parse_splash_helper_1, 59
splits_parse_splash_helper_2, 59
splits_parse_splash_relays, 60
splits_reform, 61
splits_rename_omega, 61
splits_to_cumulative, 62, 64
splits_to_cumulative_helper_recalc, 63
splits_to_lap, 62, 64
splits_to_lap_helper_recalc, 65
Swim_Parse, 66
swim_parse, 5, 18, 21, 24, 42, 44, 57
swim_parse (Swim_Parse), 66
swim_parse_hytek, 68
Swim_Parse_ISL (swim_parse_ISL), 69
swim_parse_ISL, 19, 57, 69
swim_parse_old, 70
swim_parse_omega, 23, 58, 61, 72
swim_parse_samms, 73
swim_parse_splash, 7, 59, 60, 74
swim_place, 75
SwimmerR-defunct, 66
SwimmerR-deprecated, 66
tie_rescore, 77
toptimes_parse_hytek, 77
undo_interleave, 78
update_rank_helper, 79