Package ‘ds4psy’

April 7, 2022

Type Package

Title Data Science for Psychologists

Version 0.8.0

Date 2022-04-07

Maintainer Hansjoerg Neth <h.neth@uni.kn>

Description All datasets and functions required for the examples and exercises of the book ``Data Science for Psychologists'' (by Hansjoerg Neth, Konstanz University, 2022), available at <https://bookdown.org/hneth/ds4psy/>. The book and course introduce principles and methods of data science to students of psychology and other biological or social sciences. The 'ds4psy' package primarily provides datasets, but also functions for data generation and manipulation (e.g., of text and time data) and graphics that are used in the book and its exercises. All functions included in 'ds4psy' are designed to be explicit and instructive, rather than efficient or elegant.

Depends R (>= 3.5.0)

Imports ggplot2, unikn

Suggests knitr, rmarkdown, spelling

Collate 'util_fun.R' 'time_util_fun.R' 'color_fun.R' 'data.R' 'data_fun.R' 'text_fun.R' 'time_fun.R' 'theme_fun.R' 'plot_fun.R' 'start.R'

Encoding UTF-8

LazyData true

License CC BY-SA 4.0

URL https://bookdown.org/hneth/ds4psy/,
https://github.com/hneth/ds4psy/

BugReports https://github.com/hneth/ds4psy/issues

VignetteBuilder knitr

RoxygenNote 7.1.1

Language en-US

NeedsCompilation no

Author Hansjoerg Neth [aut, cre] (<https://orcid.org/0000-0001-5427-3141>)

Repository CRAN

Date/Publication 2022-04-07 20:22:29 UTC

R topics documented:

Bushisms 4
capitalize 5
caseflip 6
cclass 7
change_time 7
change_tz 9
coin 10
countries 11
count_chars 12
count_chars_words 13
count_words 14
cur_date 15
cur_time 16
data_1 17
data_2 18
data_t1 19
data_t1_de 19
data_t1_tab 20
data_t2 20
data_t3 21
data_t4 22
days_in_month 22
dice 23
dice_2 24
diff_dates 25
diff_times 28
diff_tz 29
ds4psy.guide 31
dt_10 31
exp_num_dt 32
exp_wide 33
falsePosPsy_all 33
fame 35
flowery 36
fruits 36
get_set 37
invert_rules 38
is_equal 39
is_leap_year 40
is_vect 41
is_wholenumber 42

l33t_rul35 43
make_grid 44
map_text_chars 45
map_text_coord 46
map_text_regex 47
metachar 50
num_as_char 51
num_as_ordinal 52
num_equal 53
outliers 54
pal_ds4psy 55
pal_n_sq 56
pi_100k 56
plot_charmap 57
plot_chars 59
plot_fn 62
plot_fun 64
plot_n 65
plot_text 67
plot_tiles 70
posPsy_AHI_CESD 72
posPsy_long 73
posPsy_p_info 74
posPsy_wide 75
read_ascii 76
sample_char 77
sample_date 78
sample_time 79
t3 81
t4 81
table6 82
table7 83
table8 83
table9 84
tb 85
text_to_chars 86
text_to_sentences 87
text_to_words 88
theme_clean 89
theme_ds4psy 91
theme_empty 93
transl33t 95
Trumpisms 96
t_1 97
t_2 97
t_3 98
t_4 98
Umlaut 99

what_date . . . 100
what_month . . . 101
what_time . . . 102
what_wday . . . 104
what_week . . . 105
what_year . . . 106
words_to_text . . . 107
zodiac . . . 108

Index **110**

Bushisms *Data: Bushisms.*

Description

Bushisms contains phrases spoken by or attributed to U.S. president George W. Bush (the 43rd president of the United States, in office from January 2001 to January 2009).

Usage

```
Bushisms
```

Format

A vector of type character with length(Bushisms) = 22.

Source

Data based on https://en.wikipedia.org/wiki/Bushism.

See Also

Other datasets: Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb capitalize *Capitalize initial characters in strings of text* x.

Description

capitalize converts the case of each element's (i.e., character string or word in text) n initial characters to upper or lowercase.

Usage

```
capitalize(x, n = 1, upper = TRUE, as_text = FALSE)
```

Arguments

x	A string of text (required).
n	Number of initial characters to convert. Default: n = 1.
upper	Convert to uppercase? Default: upper = TRUE.
as_text	Treat and return x as a text (i.e., one character string)? Default: as_text = FALSE.

Details

If as_text = TRUE, the input x is merged into one string of text and the arguments are applied to each word.

Value

A character vector.

See Also

caseflip for converting the case of all letters; words_to_text and text_to_words for converting character vectors and texts.

Other text objects and functions: Umlaut, caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
x <- c("Hello world!", "this is a TEST sentence.", "the end.")
capitalize(x)
capitalize(tolower(x))

# Options:
capitalize(x, n = 3)                  # leaves strings intact
capitalize(x, n = 3, as_text = TRUE)  # treats strings as text
```

```
capitalize(x, n = 3, upper = FALSE)   # first n in lowercase
```

---

caseflip — *Flip the case of characters in a string of text* x.

Description

caseflip flips the case of all characters in a string of text x.

Usage

```
caseflip(x)
```

Arguments

x	A string of text (required).

Details

Internally, caseflip uses the letters and LETTERS constants of **base** R and the chartr function for replacing characters in strings of text.

Value

A character vector.

See Also

capitalize for converting the case of initial letters; chartr for replacing characters in strings of text.

Other text objects and functions: Umlaut, capitalize(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
x <- c("Hello world!", "This is a 1st sentence.", "This is the 2nd sentence.", "The end.")
caseflip(x)
```

cclass — *cclass provides character classes (as a named vector).*

Description

`cclass` provides different character classes (as a named character vector).

Usage

```
cclass
```

Format

An object of class `character` of length 6.

Details

`cclass` allows illustrating matching character classes via regular expressions.

See ?`base::regex` for details on regular expressions and ?`"'"` for a list of character constants/quotes in R.

See Also

`metachar` for a vector of metacharacters.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `count_chars_words()`, `count_chars()`, `count_words()`, `invert_rules()`, `l33t_rul35`, `map_text_chars()`, `map_text_coord()`, `map_text_regex()`, `metachar`, `read_ascii()`, `text_to_chars()`, `text_to_sentences()`, `text_to_words()`, `transl33t()`, `words_to_text()`

Examples

```
cclass["hex"]  # select by name
writeLines(cclass["pun"])
grep("[[:alpha:]]", cclass, value = TRUE)
```

change_time — *Change time and time zone (without changing time display).*

Description

`change_time` changes the time and time zone without changing the time display.

Usage

```
change_time(time, tz = "")
```

Arguments

time	Time (as a scalar or vector). If time is not a local time (of the "POSIXlt" class) the function first tries coercing time into "POSIXlt" without changing the time display.
tz	Time zone (as character string). Default: tz = "" (i.e., current system time zone, Sys.timezone()). See OlsonNames() for valid options.

Details

change_time expects inputs to time to be local time(s) (of the "POSIXlt" class) and a valid time zone argument tz (as a string) and returns the same time display (but different actual times) as calendar time(s) (of the "POSIXct" class).

Value

A calendar time of class "POSIXct".

See Also

change_tz function which preserves time but changes time display; Sys.time() function of **base** R.

Other date and time functions: change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
change_time(as.POSIXlt(Sys.time()), tz = "UTC")

# from "POSIXlt" time:
t1 <- as.POSIXlt("2020-01-01 10:20:30", tz = "Europe/Berlin")
change_time(t1, "NZ")
change_time(t1, "US/Pacific")

# from "POSIXct" time:
tc <- as.POSIXct("2020-07-01 12:00:00", tz = "UTC")
change_time(tc, "NZ")

# from "Date":
dt <- as.Date("2020-12-31", tz = "US/Hawaii")
change_time(dt, tz = "NZ")

# from time "string":
ts <- "2020-12-31 20:30:45"
change_time(ts, tz = "US/Pacific")

# from other "string" times:
tx <- "7:30:45"
change_time(tx, tz = "Asia/Calcutta")
ty <- "1:30"
```

```
change_time(ty, tz = "Europe/London")

# convert into local times:
(l1 <- as.POSIXlt("2020-06-01 10:11:12"))
change_tz(change_time(l1, "NZ"), tz = "UTC")
change_tz(change_time(l1, "Europe/Berlin"), tz = "UTC")
change_tz(change_time(l1, "US/Eastern"), tz = "UTC")

# with vector of "POSIXlt" times:
(l2 <- as.POSIXlt("2020-12-31 23:59:55", tz = "US/Pacific"))
(tv <- c(l1, l2))           # uses tz of l1
change_time(tv, "US/Pacific")  # change time and tz
```

---

change_tz *Change time zone (without changing represented time).*

Description

change_tz changes the nominal time zone (i.e., the time display) without changing the actual time.

Usage

```
change_tz(time, tz = "")
```

Arguments

time	Time (as a scalar or vector). If time is not a calendar time (of the "POSIXct" class) the function first tries coercing time into "POSIXct" without changing the denoted time.
tz	Time zone (as character string). Default: tz = "" (i.e., current system time zone, Sys.timezone()). See OlsonNames() for valid options.

Details

change_tz expects inputs to time to be calendar time(s) (of the "POSIXct" class) and a valid time zone argument tz (as a string) and returns the same time(s) as local time(s) (of the "POSIXlt" class).

Value

A local time of class "POSIXlt".

See Also

change_time function which preserves time display but changes time; Sys.time() function of **base** R.

Other date and time functions: change_time(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
change_tz(Sys.time(), tz = "NZ")
change_tz(Sys.time(), tz = "US/Hawaii")

# from "POSIXct" time:
tc <- as.POSIXct("2020-07-01 12:00:00", tz = "UTC")
change_tz(tc, "Australia/Melbourne")
change_tz(tc, "Europe/Berlin")
change_tz(tc, "US/Pacific")

# from "POSIXlt" time:
tl <- as.POSIXlt("2020-07-01 12:00:00", tz = "UTC")
change_tz(tl, "Australia/Melbourne")
change_tz(tl, "Europe/Berlin")
change_tz(tl, "US/Pacific")

# from "Date":
dt <- as.Date("2020-12-31")
change_tz(dt, "NZ")
change_tz(dt, "US/Hawaii")  # Note different date!

# with a vector of "POSIXct" times:
t2 <- as.POSIXct("2020-12-31 23:59:55", tz = "US/Pacific")
tv <- c(tc, t2)
tv  # Note: Both times in tz of tc
change_tz(tv, "US/Pacific")
```

---

coin *Flip a fair coin (with 2 sides "H" and "T") n times.*

Description

`coin` generates a sequence of events that represent the results of flipping a fair coin `n` times.

Usage

```
coin(n = 1, events = c("H", "T"))
```

Arguments

n	Number of coin flips. Default: `n = 1`.
events	Possible outcomes (as a vector). Default: `events = c("H","T")`.

Details

By default, the 2 possible `events` for each flip are "H" (for "heads") and "T" (for "tails").

See Also

Other sampling functions: dice_2(), dice(), sample_char(), sample_date(), sample_time()

Examples

```
# Basics:
coin()
table(coin(n = 100))
table(coin(n = 100, events = LETTERS[1:3]))

# Note an oddity:
coin(10, events = 8:9)  # works as expected, but
coin(10, events = 9:9)  # odd: see sample() for an explanation.

# Limits:
coin(2:3)
coin(NA)
coin(0)
coin(1/2)
coin(3, events = "X")
coin(3, events = NA)
coin(NULL, NULL)
```

---

countries *Data: Names of countries.*

Description

countries is a dataset containing the names of 197 countries (as a vector of text strings).

Usage

```
countries
```

Format

A vector of type character with length(countries) = 197.

Source

Data from https://www.gapminder.org: Original data at https://www.gapminder.org/data/documentation/gd004/.

See Also

Other datasets: Bushisms, Trumpisms, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb count_chars *Count the frequency of characters in a string of text* x.

Description

count_chars provides frequency counts of the characters in a string of text x as a named numeric vector.

Usage

```
count_chars(x, case_sense = TRUE, rm_specials = TRUE, sort_freq = TRUE)
```

Arguments

x	A string of text (required).
case_sense	Boolean: Distinguish lower- vs. uppercase characters? Default: case_sense = TRUE.
rm_specials	Boolean: Remove special characters? Default: rm_specials = TRUE.
sort_freq	Boolean: Sort output by character frequency? Default: sort_freq = TRUE.

Details

If rm_specials = TRUE (as per default), most special (or non-word) characters are removed and not counted. (Note that this currently works without using regular expressions.)

The quantification is case-sensitive and the resulting vector is sorted by name (alphabetically) or by frequency (per default).

Value

A named numeric vector.

See Also

count_words for counting the frequency of words; count_chars_words for counting both characters and words; plot_chars for a corresponding plotting function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
# Default:
x <- c("Hello world!", "This is a 1st sentence.",
       "This is the 2nd sentence.", "THE END.")
count_chars(x)

# Options:
count_chars(x, case_sense = FALSE)
count_chars(x, rm_specials = FALSE)
count_chars(x, sort_freq = FALSE)
```

---

count_chars_words *Count the frequency of characters and words in a string of text* x.

Description

count_chars_words provides frequency counts of the characters and words of a string of text x on a per character basis.

Usage

```
count_chars_words(x, case_sense = TRUE, sep = "|", rm_sep = TRUE)
```

Arguments

x	A string of text (required).
case_sense	Boolean: Distinguish lower- vs. uppercase characters? Default: case_sense = TRUE.
sep	Dummy character(s) to insert between elements/lines when parsing a multi-element character vector x as input. This character is inserted to mark word boundaries in multi-element inputs x (without punctuation at the boundary). It should NOT occur anywhere in x, so that it can be removed again (by rm_sep = TRUE). Default: sep = "|" (i.e., insert a vertical bar between lines).
rm_sep	Should sep be removed from output? Default: rm_sep = TRUE.

Details

count_chars_words calls both count_chars and count_words and maps their results to a data frame that contains a row for each character of x.

The quantifications are case-sensitive. Special characters (e.g., parentheses, punctuation, and spaces) are counted as characters, but removed from word counts.

If input x consists of multiple text strings, they are collapsed with an added " " (space) between them.

Value

A data frame with 4 variables (char, char_freq, word, word_freq).

See Also

count_chars for counting the frequency of characters; count_words for counting the frequency of words; plot_chars for a character plotting function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
s1 <- ("This test is to test this function.")
head(count_chars_words(s1))
head(count_chars_words(s1, case_sense = FALSE))

s3 <- c("A 1st sentence.", "The 2nd sentence.",
        "A 3rd --- and also THE  FINAL --- SENTENCE.")
tail(count_chars_words(s3))
tail(count_chars_words(s3, case_sense = FALSE))
```

---

count_words *Count the frequency of words in a string of text x.*

Description

count_words provides frequency counts of the words in a string of text x as a named numeric vector.

Usage

```
count_words(x, case_sense = TRUE, sort_freq = TRUE)
```

Arguments

x	A string of text (required).
case_sense	Boolean: Distinguish lower- vs. uppercase characters? Default: case_sense = TRUE.
sort_freq	Boolean: Sort output by word frequency? Default: sort_freq = TRUE.

Details

Special (or non-word) characters are removed and not counted.

The quantification is case-sensitive and the resulting vector is sorted by name (alphabetically) or by frequency (per default).

Value

A named numeric vector.

See Also

count_chars for counting the frequency of characters; count_chars_words for counting both characters and words; plot_chars for a character plotting function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
# Default:
s3 <- c("A first sentence.", "The second sentence.",
        "A third --- and also THE FINAL --- SENTENCE.")
count_words(s3)  # case-sensitive, sorts by frequency

# Options:
count_words(s3, case_sense = FALSE)  # case insensitive
count_words(s3, sort_freq = FALSE)   # sorts alphabetically
```

---

cur_date   *Current date (in yyyy-mm-dd or dd-mm-yyyy format).*

Description

cur_date provides a relaxed version of Sys.time() that is sufficient for most purposes.

Usage

```
cur_date(rev = FALSE, as_string = TRUE, sep = "-")
```

Arguments

rev	Boolean: Reverse from "yyyy-mm-dd" to "dd-mm-yyyy" format? Default: rev = FALSE.
as_string	Boolean: Return as character string? Default: as_string = TRUE. If as_string = FALSE, a "Date" object is returned.
sep	Character: Separator to use. Default: sep = "-".

Details

By default, cur_date returns Sys.Date as a character string (using current system settings and sep for formatting). If as_string = FALSE, a "Date" object is returned.

Alternatively, consider using Sys.Date or Sys.time() to obtain the " format according to the ISO 8601 standard.

For more options, see the documentations of the date and Sys.Date functions of **base** R and the formatting options for Sys.time().

Value

A character string or object of class "Date".

See Also

what_date() function to print dates with more options; date() and today() functions of the **lubridate** package; date(), Sys.Date(), and Sys.time() functions of **base** R.

Other date and time functions: change_time(), change_tz(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
cur_date()
cur_date(sep = "/")
cur_date(rev = TRUE)
cur_date(rev = TRUE, sep = ".")

# return a "Date" object:
from <- cur_date(as_string = FALSE)
class(from)
```

---

cur_time *Current time (in hh:mm or hh:mm:ss format).*

Description

cur_time provides a satisficing version of Sys.time() that is sufficient for most purposes.

Usage

```
cur_time(seconds = FALSE, as_string = TRUE, sep = ":")
```

Arguments

seconds	Boolean: Show time with seconds? Default: seconds = FALSE.
as_string	Boolean: Return as character string? Default: as_string = TRUE. If as_string = FALSE, a "POSIXct" object is returned.
sep	Character: Separator to use. Default: sep = ":".

Details

By default, cur_time returns a Sys.time() as a character string (in " using current system settings. If as_string = FALSE, a "POSIXct" (calendar time) object is returned.

For a time zone argument, see the what_time function, or the now() function of the **lubridate** package.

Value

A character string or object of class "POSIXct".

See Also

what_time() function to print times with more options; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
cur_time()
cur_time(seconds = TRUE)
cur_time(sep = ".")

# return a "POSIXct" object:
t <- cur_time(as_string = FALSE)
format(t, "%T %Z")
```

---

data_1 *Data import data_1.*

Description

data_1 is a fictitious dataset to practice importing data (from a DELIMITED file).

Usage

```
data_1
```

Format

A table with 100 cases (rows) and 4 variables (columns).

Source

See DELIMITED data at http://rpository.com/ds4psy/data/data_1.dat.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

data_2 *Data import data_2.*

Description

data_2 is a fictitious dataset to practice importing data (from a FWF file).

Usage

```
data_2
```

Format

A table with 100 cases (rows) and 4 variables (columns).

Source

See FWF data at http://rpository.com/ds4psy/data/data_2.dat.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

data_t1 — *Data table data_t1.*

Description

data_t1 is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

```
data_t1
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/data_t1.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

data_t1_de — *Data import data_t1_de.*

Description

data_t1_de is a fictitious dataset to practice importing data (from a CSV file, de/European style).

Usage

```
data_t1_de
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/data_t1_de.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

data_t1_tab *Data import data_t1_tab.*

Description

data_t1_tab is a fictitious dataset to practice importing data (from a TAB file).

Usage

```
data_t1_tab
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See TAB-delimited data at http://rpository.com/ds4psy/data/data_t1_tab.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

data_t2 *Data table data_t2.*

Description

data_t2 is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

```
data_t2
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/data_t2.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

`data_t3` *Data table data_t3.*

Description

`data_t3` is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

```
data_t3
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/data_t3.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

data_t4 *Data table data_t4.*

Description

data_t4 is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

```
data_t4
```

Format

A table with 20 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/data_t4.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

days_in_month *How many days are in a month (of given date)?*

Description

days_in_month computes the number of days in the months of given dates (provided as a date or time dt, or number/string denoting a 4-digit year).

Usage

```
days_in_month(dt = Sys.Date(), ...)
```

Arguments

dt	Date or time (scalar or vector). Default: dt = Sys.Date(). Numbers or strings with dates are parsed into 4-digit numbers denoting the year.
...	Other parameters (passed to as.Date()).

Details

The function requires dt as "Dates", rather than month names or numbers, to check for leap years (in which February has 29 days).

Value

A named (numeric) vector.

See Also

is_leap_year to check for leap years; diff_tz for time zone-based time differences; days_in_month function of the **lubridate** package.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
days_in_month()

# Robustness:
days_in_month(Sys.Date())    # Date
days_in_month(Sys.time())    # POSIXct
days_in_month("2020-07-01")  # string
days_in_month(20200901)      # number
days_in_month(c("2020-02-10 01:02:03", "2021-02-11", "2024-02-12"))  # vectors of strings

# For leap years:
ds <- as.Date("2020-02-20") + (365 * 0:4)
days_in_month(ds)  # (2020/2024 are leap years)
```

---

dice — *Throw a fair dice (with a given number of sides) n times.*

Description

dice generates a sequence of events that represent the results of throwing a fair dice (with a given number of events or number of sides) n times.

Usage

```
dice(n = 1, events = 1:6)
```

Arguments

n	Number of dice throws. Default: n = 1.
events	Events to draw from (or number of sides). Default: events = 1:6.

Details

By default, the 6 possible `events` for each throw of the dice are the numbers from 1 to 6.

See Also

Other sampling functions: `coin()`, `dice_2()`, `sample_char()`, `sample_date()`, `sample_time()`

Examples

```
# Basics:
dice()
table(dice(10^4))

# 5-sided dice:
dice(events = 1:5)
table(dice(100, events = 5))

# Strange dice:
dice(5, events = 8:9)
table(dice(100, LETTERS[1:3]))

# Note:
dice(10, 1)
table(dice(100, 2))

# Note an oddity:
dice(10, events = 8:9)  # works as expected, but
dice(10, events = 9:9)  # odd: see sample() for an explanation.

# Limits:
dice(NA)
dice(0)
dice(1/2)
dice(2:3)
dice(5, events = NA)
dice(5, events = 1/2)
dice(NULL, NULL)
```

`dice_2` *Throw a questionable dice (with a given number of sides) n times.*

Description

`dice_2` is a variant of `dice` that generates a sequence of events that represent the results of throwing a dice (with a given number of `sides`) `n` times.

Usage

```
dice_2(n = 1, sides = 6)
```

Arguments

n	Number of dice throws. Default: n = 1.
sides	Number of sides. Default: sides = 6.

Details

Something is wrong with this dice. Can you examine it and measure its problems in a quantitative fashion?

See Also

Other sampling functions: coin(), dice(), sample_char(), sample_date(), sample_time()

Examples

```
# Basics:
dice_2()
table(dice_2(100))

# 10-sided dice:
dice_2(sides = 10)
table(dice_2(100, sides = 10))

# Note:
dice_2(10, 1)
table(dice_2(5000, sides = 5))

# Note an oddity:
dice_2(n = 10, sides = 8:9)  # works, but
dice_2(n = 10, sides = 9:9)  # odd: see sample() for an explanation.
```

---

diff_dates — *Get the difference between two dates (in human units).*

Description

diff_dates computes the difference between two dates (i.e., from some from_date to some to_date) in human measurement units (periods).

Usage

```
diff_dates(
  from_date,
  to_date = Sys.Date(),
  unit = "years",
  as_character = TRUE
)
```

Arguments

`from_date` From date (required, scalar or vector, as "Date"). Date of birth (DOB), assumed to be of class "Date", and coerced into "Date" when of class "POSIXt".

`to_date` To date (optional, scalar or vector, as "Date"). Default: `to_date = Sys.Date()`. Maximum date/date of death (DOD), assumed to be of class "Date", and coerced into "Date" when of class "POSIXt".

`unit` Largest measurement unit for representing results. Units represent human time periods, rather than chronological time differences. Default: `unit = "years"` for completed years, months, and days. Options available:

1. `unit = "years"`: completed years, months, and days (default)
2. `unit = "months"`: completed months, and days
3. `unit = "days"`: completed days

Units may be abbreviated.

`as_character` Boolean: Return output as character? Default: `as_character = TRUE`. If `as_character = FALSE`, results are returned as columns of a data frame and include `from_date` and `to_date`.

Details

`diff_dates` answers questions like "How much time has elapsed between two dates?" or "How old are you?" in human time periods of (full) years, months, and days.

Key characteristics:

- If `to_date` or `from_date` are not "Date" objects, `diff_dates` aims to coerce them into "Date" objects.
- If `to_date` is missing (i.e., `NA`), `to_date` is set to today's date (i.e., `Sys.Date()`).
- If `to_date` is specified, any intermittent missing values (i.e., `NA`) are set to today's date (i.e., `Sys.Date()`). Thus, dead people (with both birth dates and death dates specified) do not age any further, but people still alive (with `is.na(to_date)`, are measured to today's date (i.e., `Sys.Date()`).
- If `to_date` precedes `from_date` (i.e., `from_date > to_date`) computations are performed on swapped days and the result is marked as negative (by a character `"-"`) in the output.
- If the lengths of `from_date` and `to_date` differ, the shorter vector is recycled to the length of the longer one.

By default, `diff_dates` provides output as (signed) character strings. For numeric outputs, use `as_character = FALSE`.

Value

A character vector or data frame (with dates, sign, and numeric columns for units).

See Also

Time spans (`interval` `as.period`) in the **lubridate** package.

Other date and time functions: `change_time()`, `change_tz()`, `cur_date()`, `cur_time()`, `days_in_month()`, `diff_times()`, `diff_tz()`, `is_leap_year()`, `what_date()`, `what_month()`, `what_time()`, `what_wday()`, `what_week()`, `what_year()`, `zodiac()`

Examples

```
y_100 <- Sys.Date() - (100 * 365.25) + -1:1
diff_dates(y_100)

# with "to_date" argument:
y_050 <- Sys.Date() - (50 * 365.25) + -1:1
diff_dates(y_100, y_050)
diff_dates(y_100, y_050, unit = "d") # days (with decimals)

# Time unit and output format:
ds_from <- as.Date("2010-01-01") + 0:2
ds_to   <- as.Date("2020-03-01")  # (2020 is leap year)
diff_dates(ds_from, ds_to, unit = "y", as_character = FALSE)  # years
diff_dates(ds_from, ds_to, unit = "m", as_character = FALSE)  # months
diff_dates(ds_from, ds_to, unit = "d", as_character = FALSE)  # days

# Robustness:
days_cur_year <- 365 + is_leap_year(Sys.Date())
diff_dates(Sys.time() - (1 * (60 * 60 * 24) * days_cur_year))  # for POSIXt times
diff_dates("10-08-11", "20-08-10")   # for strings
diff_dates(20200228, 20200301)       # for numbers (2020 is leap year)

# Recycling "to_date" to length of "from_date":
y_050_2 <- Sys.Date() - (50 * 365.25)
diff_dates(y_100, y_050_2)

# Note maxima and minima:
diff_dates("0000-01-01", "9999-12-31")  # max. d + m + y
diff_dates("1000-06-01", "1000-06-01")  # min. d + m + y

# If from_date == to_date:
diff_dates("2000-01-01", "2000-01-01")

# If from_date > to_date:
diff_dates("2000-01-02", "2000-01-01")  # Note negation "-"
diff_dates("2000-02-01", "2000-01-01", as_character = TRUE)
diff_dates("2001-02-02", "2000-02-02", as_character = FALSE)

# Test random date samples:
f_d <- sample_date(size = 10)
t_d <- sample_date(size = 10)
diff_dates(f_d, t_d, as_character = TRUE)

# Using 'fame' data:
```

```
dob <- as.Date(fame$DOB, format = "%B %d, %Y")
dod <- as.Date(fame$DOD, format = "%B %d, %Y")
head(diff_dates(dob, dod))  # Note: Deceased people do not age further.
head(diff_dates(dob, dod, as_character = FALSE))  # numeric outputs
```

---

diff_times *Get the difference between two times (in human units).*

Description

diff_times computes the difference between two times (i.e., from some from_time to some to_time) in human measurement units (periods).

Usage

```
diff_times(from_time, to_time = Sys.time(), unit = "days", as_character = TRUE)
```

Arguments

from_time	From time (required, scalar or vector, as "POSIXct"). Origin time, assumed to be of class "POSIXct", and coerced into "POSIXct" when of class "Date" or "POSIXlt.
to_time	To time (optional, scalar or vector, as "POSIXct"). Default: to_time = Sys.time(). Maximum time, assumed to be of class "POSIXct", and coerced into "POSIXct" when of class "Date" or "POSIXlt".
unit	Largest measurement unit for representing results. Units represent human time periods, rather than chronological time differences. Default: unit = "days" for completed days, hours, minutes, and seconds. Options available: 1. unit = "years": completed years, months, and days (default) 2. unit = "months": completed months, and days 3. unit = "days": completed days 4. unit = "hours": completed hours 5. unit = "minutes": completed minutes 6. unit = "seconds": completed seconds Units may be abbreviated.
as_character	Boolean: Return output as character? Default: as_character = TRUE. If as_character = FALSE, results are returned as columns of a data frame and include from_date and to_date.

Details

diff_times answers questions like "How much time has elapsed between two dates?" or "How old are you?" in human time periods of (full) years, months, and days.

Key characteristics:

- If to_time or from_time are not "POSIXct" objects, diff_times aims to coerce them into "POSIXct" objects.
- If to_time is missing (i.e., NA), to_time is set to the current time (i.e., Sys.time()).
- If to_time is specified, any intermittent missing values (i.e., NA) are set to the current time (i.e., Sys.time()).
- If to_time precedes from_time (i.e., from_time > to_time) computations are performed on swapped times and the result is marked as negative (by a character "-") in the output.
- If the lengths of from_time and to_time differ, the shorter vector is recycled to the length of the longer one.

By default, diff_times provides output as (signed) character strings. For numeric outputs, use as_character = FALSE.

Value

A character vector or data frame (with times, sign, and numeric columns for units).

See Also

diff_dates for date differences; time spans (an interval as.period) in the **lubridate** package.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
t1 <- as.POSIXct("1969-07-13 13:53 CET")  # (before UNIX epoch)
diff_times(t1, unit = "years", as_character = TRUE)
diff_times(t1, unit = "secs", as_character = TRUE)
```

---

diff_tz *Get the time zone difference between two times.*

Description

diff_tz computes the time difference between two times t1 and t2 that is exclusively due to both times being in different time zones.

Usage

```
diff_tz(t1, t2, in_min = FALSE)
```

Arguments

t1	First time (required, as "POSIXt" time point/moment).
t2	Second time (required, as "POSIXt" time point/moment).
in_min	Return time-zone based time difference in minutes (Boolean)? Default: in_min = FALSE.

Details

diff_tz ignores all differences in nominal times, but allows adjusting time-based computations for time shifts that are due to time zone differences (e.g., different locations, or changes to/from daylight saving time, DST), rather than differences in actual times.

Internally, diff_tz determines and contrasts the POSIX conversion specifications " (in numeric form).

If the lengths of t1 and t2 differ, the shorter vector is recycled to the length of the longer one.

Value

A character (in "HH:MM" format) or numeric vector (number of minutes).

See Also

days_in_month for the number of days in given months; is_leap_year to check for leap years.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
# Time zones differences:
tm <- "2020-01-01 01:00:00"  # nominal time
t1 <- as.POSIXct(tm, tz = "NZ")
t2 <- as.POSIXct(tm, tz = "Europe/Berlin")
t3 <- as.POSIXct(tm, tz = "US/Hawaii")

# as character (in "HH:MM"):
diff_tz(t1, t2)
diff_tz(t2, t3)
diff_tz(t1, t3)

# as numeric (in minutes):
diff_tz(t1, t3, in_min = TRUE)

# Compare local times (POSIXlt):
t4 <- as.POSIXlt(Sys.time(), tz = "NZ")
t5 <- as.POSIXlt(Sys.time(), tz = "Europe/Berlin")
diff_tz(t4, t5)
diff_tz(t4, t5, in_min = TRUE)

# DSL shift: Spring ahead (on 2020-03-29: 02:00:00 > 03:00:00):
```

```
s6 <- "2020-03-29 01:00:00 CET"   # before DSL switch
s7 <- "2020-03-29 03:00:00 CEST"  # after DSL switch
t6 <- as.POSIXct(s6, tz = "Europe/Berlin")  # CET
t7 <- as.POSIXct(s7, tz = "Europe/Berlin")  # CEST

diff_tz(t6, t7)  # 1 hour forwards
diff_tz(t6, t7, in_min = TRUE)
```

---

ds4psy.guide — *Opens user guide of the ds4psy package.*

Description

Opens user guide of the ds4psy package.

Usage

```
ds4psy.guide()
```

---

dt_10 — *Data from 10 Danish people.*

Description

`dt_10` contains precise DOB information of 10 non-existent, but definitely Danish people.

Usage

```
dt_10
```

Format

A table with 10 cases (rows) and 7 variables (columns).

Source

See CSV data file at http://rpository.com/ds4psy/data/dt_10.csv.

See Also

Other datasets: `Bushisms`, `Trumpisms`, `countries`, `data_1`, `data_2`, `data_t1_de`, `data_t1_tab`, `data_t1`, `data_t2`, `data_t3`, `data_t4`, `exp_num_dt`, `exp_wide`, `falsePosPsy_all`, `fame`, `flowery`, `fruits`, `outliers`, `pi_100k`, `posPsy_AHI_CESD`, `posPsy_long`, `posPsy_p_info`, `posPsy_wide`, `t3`, `t4`, `t_1`, `t_2`, `t_3`, `t_4`, `table6`, `table7`, `table8`, `table9`, `tb` exp_num_dt *Data from an experiment with numeracy and date-time variables.*

Description

exp_num_dt is a fictitious dataset describing 1000 non-existing, but surprisingly friendly people.

Usage

```
exp_num_dt
```

Format

A table with 1000 cases (rows) and 15 variables (columns).

Details

Codebook

The table contains 15 columns/variables:

- 1. **name**: Participant initials.
- 2. **gender**: Self-identified gender.
- 3. **bday**: Day (within month) of DOB.
- 4. **bmonth**: Month (within year) of DOB.
- 5. **byear**: Year of DOB.
- 6. **height**: Height (in cm).
- 7. **blood_type**: Blood type.
- 8. **bnt_1** to 11. **bnt_4**: Correct response to BNT question? (1: correct, 0: incorrect).
- 12. **g_iq** and 13. **s_iq**: Scores from two IQ tests (general vs. social).
- 14. **t_1** and 15. **t_2**: Start and end time.

exp_num_dt was generated for analyzing test scores (e.g., IQ, numeracy), for converting data from wide into long format, and for dealing with date- and time-related variables.

Source

See CSV data files at http://rpository.com/ds4psy/data/numeracy.csv and http://rpository.com/ds4psy/data/dt.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

exp_wide *Data exp_wide.*

Description

exp_wide is a fictitious dataset to practice tidying data (here: converting from wide to long format).

Usage

```
exp_wide
```

Format

A table with 10 cases (rows) and 7 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/exp_wide.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

falsePosPsy_all *False Positive Psychology data.*

Description

falsePosPsy_all is a dataset containing the data from 2 studies designed to highlight problematic research practices within psychology.

Usage

```
falsePosPsy_all
```

Format

A table with 78 cases (rows) and 19 variables (columns):

Details

Simmons, Nelson and Simonsohn (2011) published a controversial article with a necessarily false finding. By conducting simulations and 2 simple behavioral experiments, the authors show that flexibility in data collection, analysis, and reporting dramatically increases the rate of false-positive findings.

study Study ID.

id Participant ID.

aged Days since participant was born (based on their self-reported birthday).

aged365 Age in years.

female Is participant a woman? 1: yes, 2: no.

dad Father's age (in years).

mom Mother's age (in years).

potato Did the participant hear the song 'Hot Potato' by The Wiggles? 1: yes, 2: no.

when64 Did the participant hear the song 'When I am 64' by The Beatles? 1: yes, 2: no.

kalimba Did the participant hear the song 'Kalimba' by Mr. Scrub? 1: yes, 2: no.

cond In which condition was the participant? control: Subject heard the song 'Kalimba' by Mr. Scrub; potato: Subject heard the song 'Hot Potato' by The Wiggles; 64: Subject heard the song 'When I am 64' by The Beatles.

root Could participant report the square root of 100? 1: yes, 2: no.

bird Imagine a restaurant you really like offered a 30 percent discount for dining between 4pm and 6pm. How likely would you be to take advantage of that offer? Scale from 1: very unlikely, 7: very likely.

political In the political spectrum, where would you place yourself? Scale: 1: very liberal, 2: liberal, 3: centrist, 4: conservative, 5: very conservative.

quarterback If you had to guess who was chosen the quarterback of the year in Canada last year, which of the following four options would you choose? 1: Dalton Bell, 2: Daryll Clark, 3: Jarious Jackson, 4: Frank Wilczynski.

olddays How often have you referred to some past part of your life as "the good old days"? Scale: 11: never, 12: almost never, 13: sometimes, 14: often, 15: very often.

feelold How old do you feel? Scale: 1: very young, 2: young, 3: neither young nor old, 4: old, 5: very old.

computer Computers are complicated machines. Scale from 1: strongly disagree, to 5: strongly agree.

diner Imagine you were going to a diner for dinner tonight, how much do you think you would like the food? Scale from 1: dislike extremely, to 9: like extremely.

See `https://bookdown.org/hneth/ds4psy/B-2-datasets-false.html` for codebook and more information.

Source

Articles

- Simmons, J.P., Nelson, L.D., & Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, *22*(11), 1359–1366. doi: `10.1177/0956797611417632`
- Simmons, J.P., Nelson, L.D., & Simonsohn, U. (2014). Data from paper "False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant". *Journal of Open Psychology Data*, *2*(1), e1. doi: `10.5334/jopd.aa`

See files at `https://openpsychologydata.metajnl.com/articles/10.5334/jopd.aa/` and the archive at `https://zenodo.org/record/7664` for original dataset.

See Also

Other datasets: `Bushisms`, `Trumpisms`, `countries`, `data_1`, `data_2`, `data_t1_de`, `data_t1_tab`, `data_t1`, `data_t2`, `data_t3`, `data_t4`, `dt_10`, `exp_num_dt`, `exp_wide`, `fame`, `flowery`, `fruits`, `outliers`, `pi_100k`, `posPsy_AHI_CESD`, `posPsy_long`, `posPsy_p_info`, `posPsy_wide`, `t3`, `t4`, `t_1`, `t_2`, `t_3`, `t_4`, `table6`, `table7`, `table8`, `table9`, `tb`

`fame` *Data table fame.*

Description

`fame` is a dataset to practice working with dates.

`fame` contains the names, areas, dates of birth (DOB), and — if applicable — the dates of death (DOD) of famous people.

Usage

```
fame
```

Format

A table with 67 cases (rows) and 4 variables (columns).

Source

Student solutions to exercises, dates mostly from `https://www.wikipedia.org/`.

See Also

Other datasets: `Bushisms`, `Trumpisms`, `countries`, `data_1`, `data_2`, `data_t1_de`, `data_t1_tab`, `data_t1`, `data_t2`, `data_t3`, `data_t4`, `dt_10`, `exp_num_dt`, `exp_wide`, `falsePosPsy_all`, `flowery`, `fruits`, `outliers`, `pi_100k`, `posPsy_AHI_CESD`, `posPsy_long`, `posPsy_p_info`, `posPsy_wide`, `t3`, `t4`, `t_1`, `t_2`, `t_3`, `t_4`, `table6`, `table7`, `table8`, `table9`, `tb` flowery *Data: Flowery phrases.*

Description

flowery contains versions and variations of Gertrude Stein's popular phrase "A rose is a rose is a rose".

Usage

```
flowery
```

Format

A vector of type character with length(flowery) = 60.

Details

The phrase stems from Gertrude Stein's poem "Sacred Emily" (written in 1913 and published in 1922, in "Geography and Plays"). The verbatim line in the poem actually reads "Rose is a rose is a rose is a rose".

See https://en.wikipedia.org/wiki/Rose_is_a_rose_is_a_rose_is_a_rose for additional variations and sources.

Source

Data based on https://en.wikipedia.org/wiki/Rose_is_a_rose_is_a_rose_is_a_rose.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

fruits *Data: Names of fruits.*

Description

fruits is a dataset containing the names of 122 fruits (as a vector of text strings).

Usage

```
fruits
```

Format

A vector of type character with length(fruits) = 122.

Details

Botanically, "fruits" are the seed-bearing structures of flowering plants (angiosperms) formed from the ovary after flowering.

In common usage, "fruits" refer to the fleshy seed-associated structures of a plant that taste sweet or sour, and are edible in their raw state.

Source

Data based on https://simple.wikipedia.org/wiki/List_of_fruits.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

get_set *Get a set of x-y coordinates.*

Description

get_set obtains a set of x/y coordinates and returns it (as a data frame).

Usage

```
get_set(n = 1)
```

Arguments

n	Number of set (as an integer from 1 to 4)). Default: n = 1.

Details

Each set stems from Anscombe's Quartet (see datasets::anscombe, hence 1 <= n <= 4) and is returned as an 11 x 2 data frame.

Source

See ?datasets:anscombe for details and references.

See Also

Other data functions: make_grid()

Examples

```
get_set(1)
plot(get_set(2), col = "red")
```

---

invert_rules — *invert_rules inverts a set of encoding rules.*

Description

`invert_rules` allows decoding messages that were encoded by a set of rules `x`.

Usage

```
invert_rules(x)
```

Arguments

`x`	The rules used for encoding a message (as a named vector).

Details

`x` is assumed to be a named vector.

`invert_rules` replaces the elements of `x` by the names of `x`, and vice versa.

A message is issued if the elements of `x` are repeated (i.e., decoding is non-unique).

Value

A character vector.

See Also

`transl33t` for encoding text (e.g., into leet slang); `l33t_rul35` for default rules used.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars_words()`, `count_chars()`, `count_words()`, `l33t_rul35`, `map_text_chars()`, `map_text_coord()`, `map_text_regex()`, `metachar`, `read_ascii()`, `text_to_chars()`, `text_to_sentences()`, `text_to_words()`, `transl33t()`, `words_to_text()`

Examples

```
invert_rules(l33t_rul35)  # Note repeated elements

# Encoding and decoding a message:
(txt_0 <- "Hello world! How are you doing today?")             # message
(txt_1 <- transl33t(txt_0, rules = l33t_rul35))                # encoding
(txt_2 <- transl33t(txt_1, rules = invert_rules(l33t_rul35)))  # decoding
```

is_equal *Test two vectors for pairwise (near) equality.*

Description

is_equal tests if two vectors x and y are pairwise equal.

Usage

```
is_equal(x, y, ...)
```

Arguments

x	1st vector to compare (required).
y	2nd vector to compare (required).
...	Other parameters (passed to num_equal()).

Details

If both x and y are numeric, is_equal calls num_equal(x,y,...) (allowing for some tolerance threshold tol).

Otherwise, x and y are compared by x == y.

is_equal is a safer way to verify the (near) equality of numeric vectors than ==, as numbers may exhibit floating point effects.

See Also

num_equal function for comparing numeric vectors; all.equal function of the R **base** package; near function of the **dplyr** package.

Other utility functions: is_vect(), is_wholenumber(), num_as_char(), num_as_ordinal(), num_equal()

Examples

```
# numeric data:
is_equal(2, sqrt(2)^2)
is_equal(2, sqrt(2)^2, tol = 0)
is_equal(c(2, 3), c(sqrt(2)^2, sqrt(3)^2, 4/2, 9/3))

# other data types:
is_equal((1:3 > 1), (1:3 > 2))                         # logical
is_equal(c("A", "B", "c"), toupper(c("a", "b", "c")))  # character
is_equal(as.Date("2020-08-16"), Sys.Date())            # dates

# as factors:
is_equal((1:3 > 1), as.factor((1:3 > 2)))
```

```
is_equal(c(1, 2, 3), as.factor(c(1, 2, 3)))
is_equal(c("A", "B", "C"), as.factor(c("A", "B", "C")))
```

---

is_leap_year *Is some year a so-called leap year?*

Description

is_leap_year checks whether a given year (provided as a date or time dt, or number/string denoting a 4-digit year) lies in a so-called leap year (i.e., a year containing a date of Feb-29).

Usage

```
is_leap_year(dt)
```

Arguments

dt	Date or time (scalar or vector). Numbers or strings with dates are parsed into 4-digit numbers denoting the year.

Details

When dt is not recognized as "Date" or "POSIXt" object(s), is_leap_year aims to parse a string dt as describing year(s) in a "dddd" (4-digit year) format, as a valid "Date" string (to retrieve the 4-digit year "%Y"), or a numeric dt as 4-digit integer(s).

is_leap_year then solves the task by verifying the numeric definition of a "leap year" (see https://en.wikipedia.org/wiki/Leap_year).

An alternative solution that tried using as.Date() for defining a "Date" of Feb-29 in the corresponding year(s) was removed, as it evaluated NA values as FALSE.

Value

Boolean vector.

Source

See https://en.wikipedia.org/wiki/Leap_year for definition.

See Also

days_in_month for the number of days in given months; diff_tz for time zone-based time differences; leap_year function of the **lubridate** package.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
is_leap_year(2020)
(days_this_year <- 365 + is_leap_year(Sys.Date()))

# from dates:
is_leap_year(Sys.Date())
is_leap_year(as.Date("2022-02-28"))

# from times:
is_leap_year(Sys.time())
is_leap_year(as.POSIXct("2022-10-11 10:11:12"))
is_leap_year(as.POSIXlt("2022-10-11 10:11:12"))

# from non-integers:
is_leap_year(2019.5)

# For vectors:
is_leap_year(2020:2028)

# with dt as strings:
is_leap_year(c("2020", "2021"))
is_leap_year(c("2020-02-29 01:02:03", "2021-02-28 01:02"))

# Note: Invalid date string yields error:
# is_leap_year("2021-02-29")
```

`is_vect` *Test for a vector (i.e., atomic vector or list).*

Description

`is_vect` tests if `x` is a vector.

Usage

```
is_vect(x)
```

Arguments

`x`	Vector(s) to test (required).

Details

`is_vect` does what the **base** R function `is.vector` is **not** designed to do:

- `is_vect()` returns TRUE if `x` is an atomic vector or a list (irrespective of its attributes).
- `is.vector()` returns TRUE if `x` is a vector of the specified `mode` having no attributes other than names, otherwise FALSE.

Internally, the function is a wrapper for `is.atomic(x) | is.list(x)`.

Note that data frames are also vectors.

See the `is_vector` function of the **purrr** package and the **base** R functions `is.atomic`, `is.list`, and `is.vector`, for details.

See Also

`is_vect` function of the **purrr** package; `is.atomic` function of the R **base** package; `is.list` function of the R **base** package; `is.vector` function of the R **base** package.

Other utility functions: `is_equal()`, `is_wholenumber()`, `num_as_char()`, `num_as_ordinal()`, `num_equal()`

Examples

```
# Define 3 types of vectors:
v1 <- 1:3  # (a) atomic vector
names(v1) <- LETTERS[v1]  # with names

v2 <- v1   # (b) copy vector
attr(v2, "my_attr") <- "foo"  # add an attribute
ls <- list(1, 2, "C")  # (c) list

# Compare:
is.vector(v1)
is.list(v1)
is_vect(v1)

is.vector(v2)  # FALSE
is.list(v2)
is_vect(v2)  # TRUE

is.vector(ls)
is.list(ls)
is_vect(ls)

# Data frames are also vectors:
df <- as.data.frame(1:3)
is_vect(df)  # is TRUE
```

is_wholenumber — *Test for whole numbers (i.e., integers).*

Description

`is_wholenumber` tests if `x` contains only integer numbers.

Usage

```
is_wholenumber(x, tol = .Machine$double.eps^0.5)
```

Arguments

x	Number(s) to test (required, accepts numeric vectors).
tol	Numeric tolerance value. Default: `tol = .Machine$double.eps^0.5` (see `?.Machine` for details).

Details

`is_wholenumber` does what the **base** R function `is.integer` is **not** designed to do:

- `is_wholenumber()` returns TRUE or FALSE depending on whether its numeric argument `x` is an integer value (i.e., a "whole" number).
- `is.integer()` returns TRUE or FALSE depending on whether its argument is of integer type, and FALSE if its argument is a factor.

See the documentation of `is.integer` for definition and details.

See Also

`is.integer` function of the R **base** package.

Other utility functions: `is_equal()`, `is_vect()`, `num_as_char()`, `num_as_ordinal()`, `num_equal()`

Examples

```
is_wholenumber(1)    # is TRUE
is_wholenumber(1/2)  # is FALSE
x <- seq(1, 2, by = 0.5)
is_wholenumber(x)

# Compare:
is.integer(1+2)
is_wholenumber(1+2)
```

`l33t_rul35` *l33t_rul35 provides rules for translating text into leet/l33t slang.*

Description

`l33t_rul35` specifies rules for translating characters into other characters (typically symbols) to mimic leet/l33t slang (as a named character vector).

Usage

```
l33t_rul35
```

Format

An object of class character of length 13.

Details

Old (i.e., to be replaced) characters are paste(names(l33t_rul35),collapse = "").

New (i.e., replaced) characters are paste(l33t_rul35,collapse = "").

See https://en.wikipedia.org/wiki/Leet for details.

See Also

transl33t for a corresponding function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

---

make_grid — *Generate a grid of x-y coordinates.*

Description

make_grid generates a grid of x/y coordinates and returns it (as a data frame).

Usage

```
make_grid(x_min = 0, x_max = 2, y_min = 0, y_max = 1)
```

Arguments

x_min	Minimum x coordinate. Default: x_min = 0.
x_max	Maximum x coordinate. Default: x_max = 2.
y_min	Minimum y coordinate. Default: y_min = 0.
y_max	Maximum y coordinate. Default: y_max = 1.

See Also

Other data functions: get_set()

Examples

```
make_grid()
make_grid(x_min = -3, x_max = 3, y_min = -2, y_max = 2)
```

map_text_chars — *map_text_chars maps the characters of a text string into a table (with x/y coordinates).*

Description

map_text_chars parses text (from a text string x) into a table that contains a row for each character and x/y-coordinates corresponding to the character positions in x.

Usage

```
map_text_chars(x, flip_y = FALSE)
```

Arguments

x	The text string(s) to map (required). If length(x) > 1, elements are mapped to different lines (i.e., y-coordinates).
flip_y	Boolean: Should y-coordinates be flipped, so that the lowest line in the text file becomes y = 1, and the top line in the text file becomes y = n_lines? Default: flip_y = FALSE.

Details

map_text_chars creates a data frame with 3 variables: Each character's x- and y-coordinates (from top to bottom) and a variable char for the character at these coordinates.

Note that map_text_chars was originally a part of read_ascii, but has been separated to enable independent access to separate functionalities.

Note that map_text_chars is replaced by the simpler map_text_coord function.

Value

A data frame with 3 variables: Each character's x- and y-coordinates (from top to bottom) and a variable char for the character at this coordinate.

See Also

read_ascii for parsing text from file or user input; plot_chars for a character plotting function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

map_text_coord — *map_text_coord maps the characters of a text string into a table (with x/y-coordinates).*

Description

`map_text_coord` parses text (from a text string `x`) into a table that contains a row for each character and x/y-coordinates corresponding to the character positions in `x`.

Usage

```
map_text_coord(x, flip_y = FALSE, sep = "")
```

Arguments

`x`	The text string(s) to map (required). If `length(x) > 1`, elements are mapped to different lines (i.e., y-coordinates).
`flip_y`	Boolean: Should y-coordinates be flipped, so that the lowest line in the text file becomes `y = 1`, and the top line in the text file becomes `y = n_lines`? Default: `flip_y = FALSE`.
`sep`	Character to insert between the elements of a multi-element character vector as input `x`? Default: `sep = ""` (i.e., add nothing).

Details

`map_text_coord` creates a data frame with 3 variables: Each character's `x`- and `y`-coordinates (from top to bottom) and a variable `char` for the character at these coordinates.

Note that `map_text_coord` was originally a part of `read_ascii`, but has been separated to enable independent access to separate functionalities.

Value

A data frame with 3 variables: Each character's `x`- and `y`-coordinates (from top to bottom) and a variable `char` for the character at this coordinate.

See Also

`map_text_regex` for mapping text to a character table and matching patterns; `plot_charmap` for plotting character maps; `plot_chars` for creating and plotting character maps; `read_ascii` for parsing text from file or user input.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars_words()`, `count_chars()`, `count_words()`, `invert_rules()`, `l33t_rul35`, `map_text_chars()`, `map_text_regex()`, `metachar`, `read_ascii()`, `text_to_chars()`, `text_to_sentences()`, `text_to_words()`, `transl33t()`, `words_to_text()`

Examples

```
map_text_coord("Hello world!")             # 1 line of text
map_text_coord(c("Hello", "world!"))       # 2 lines of text
map_text_coord(c("Hello", " ", "world!"))  # 3 lines of text


## Read text from file:

## Create a temporary file "test.txt":
# cat("Hello world!", "This is a test.",
#     "Can you see this text?", "Good! Please carry on...",
#      file = "test.txt", sep = "\n")

# txt <- read_ascii("test.txt")
# map_text_coord(txt)

# unlink("test.txt")  # clean up (by deleting file).
```

---

map_text_regex   *Map text to character table (allowing for matching patterns).*

Description

map_text_regex parses text (from a file or user input) into a data frame that contains a row for each character of x.

Usage

```
map_text_regex(
  x = NA,
  file = "",
  lbl_hi = NA,
  lbl_lo = NA,
  bg_hi = NA,
  bg_lo = "[[:space:]]",
  lbl_rotate = NA,
  case_sense = TRUE,
  lbl_tiles = TRUE,
  col_lbl = "black",
  col_lbl_hi = pal_ds4psy[[1]],
  col_lbl_lo = pal_ds4psy[[9]],
  col_bg = pal_ds4psy[[7]],
  col_bg_hi = pal_ds4psy[[4]],
  col_bg_lo = "white",
  col_sample = FALSE,
  angle_fg = c(-90, 90),
```

```
  angle_bg = 0
)
```

Arguments

- `x` — The text to map or plot (as a character vector). Different elements denote different lines of text. If `x = NA` (as per default), the `file` argument is used to read a text file or user input from the Console.
- `file` — A text file to read (or its path). If `file = ""` (as per default), `scan` is used to read user input from the Console. If a text file is stored in a sub-directory, enter its path and name here (without any leading or trailing "." or "/").
- `lbl_hi` — Labels to highlight (as regex). Default: `lbl_hi = NA`.
- `lbl_lo` — Labels to de-emphasize (as regex). Default: `lbl_lo = NA`.
- `bg_hi` — Background tiles to highlight (as regex). Default: `bg_hi = NA`.
- `bg_lo` — Background tiles to de-emphasize (as regex). Default: `bg_lo = "[[:space:]]"`.
- `lbl_rotate` — Labels to rotate (as regex). Default: `lbl_rotate = NA`.
- `case_sense` — Boolean: Distinguish lower- vs. uppercase characters in pattern matches? Default: `case_sense = TRUE`.
- `lbl_tiles` — Are character labels shown? This enables pattern matching for (fg) color and angle aesthetics. Default: `lbl_tiles = TRUE` (i.e., show labels).
- `col_lbl` — Default color of text labels. Default: `col_lbl = "black"`.
- `col_lbl_hi` — Highlighting color of text labels. Default: `col_lbl_hi = pal_ds4psy[[1]]`.
- `col_lbl_lo` — De-emphasizing color of text labels. Default: `col_lbl_lo = pal_ds4psy[[9]]`.
- `col_bg` — Default color to fill background tiles. Default: `col_bg = pal_ds4psy[[7]]`.
- `col_bg_hi` — Highlighting color to fill background tiles. Default: `col_bg_hi = pal_ds4psy[[4]]`.
- `col_bg_lo` — De-emphasizing color to fill background tiles. Default: `col_bg_lo = "white"`.
- `col_sample` — Boolean: Sample color vectors (within category)? Default: `col_sample = FALSE`.
- `angle_fg` — Angle(s) for rotating character labels matching the pattern of the `lbl_rotate` expression. Default: `angle_fg = c(-90,90)`. If `length(angle_fg) > 1`, a random value in uniform `range(angle_fg)` is used for every character.
- `angle_bg` — Angle(s) of rotating character labels not matching the pattern of the `lbl_rotate` expression. Default: `angle_bg = 0` (i.e., no rotation). If `length(angle_bg) > 1`, a random value in uniform `range(angle_bg)` is used for every character.

Details

`map_text_regex` allows for regular expression (regex) to match text patterns and create corresponding variables (e.g., for color or orientation).

Five regular expressions and corresponding color and angle arguments allow identifying, marking (highlighting or de-emphasizing), and rotating those sets of characters (i.e, their text labels or fill colors). that match the provided patterns.

The plot generated by `plot_chars` is character-based: Individual characters are plotted at equidistant x-y-positions and the aesthetic settings provided for text labels and tile fill colors.

map_text_regex returns a plot description (as a data frame). Using this output as an input to plot_charmap plots text in a character-based fashion (i.e., individual characters are plotted at equidistant x-y-positions). Together, both functions replace the over-specialized plot_chars and plot_text functions.

Value

A data frame describing a plot.

See Also

map_text_coord for mapping text to a table of character coordinates; plot_charmap for plotting character maps; plot_chars for creating and plotting character maps; plot_text for plotting characters and color tiles by frequency; read_ascii for reading text inputs into a character string.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
## (1) From text string(s):
ts <- c("Hello world!", "This is a test to test this splendid function",
        "Does this work?", "That's good.", "Please carry on.")
sum(nchar(ts))

# (a) simple use:
map_text_regex(ts)

# (b) matching patterns (regex):
map_text_regex(ts, lbl_hi = "\\b\\w{4}\\b", bg_hi = "[good|test]",
               lbl_rotate = "[^aeiou]", angle_fg = c(-45, +45))

## (2) From user input:
# map_text_regex()  # (enter text in Console)

## (3) From text file:
# cat("Hello world!", "This is a test file.",
#     "Can you see this text?",
#     "Good! Please carry on...",
#     file = "test.txt", sep = "\n")
#
# map_text_regex(file = "test.txt")  # default
# map_text_regex(file = "test.txt", lbl_hi = "[[:upper:]]", lbl_lo = "[[:punct:]]",
#                col_lbl_hi = "red", col_lbl_lo = "blue")
#
# map_text_regex(file = "test.txt", lbl_hi = "[aeiou]", col_lbl_hi = "red",
#                col_bg = "white", bg_hi = "see")  # mark vowels and "see" (in bg)
# map_text_regex(file = "test.txt", bg_hi = "[aeiou]", col_bg_hi = "gold")  # mark (bg of) vowels
#
# # Label options:
```

```
# map_text_regex(file = "test.txt", bg_hi = "see", lbl_tiles = FALSE)
# map_text_regex(file = "test.txt", angle_bg = c(-20, 20))
#
# unlink("test.txt")  # clean up (by deleting file).
```

---

metachar *metachar provides metacharacters (as a character vector).*

Description

`metachar` provides the metacharacters of extended regular expressions (as a character vector).

Usage

```
metachar
```

Format

An object of class `character` of length 12.

Details

`metachar` allows illustrating the notion of meta-characters in regular expressions (and provides corresponding exemplars).

See ?`base::regex` for details on regular expressions and ?`"'"` for a list of character constants/quotes in R.

See Also

`cclass` for a vector of character classes.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars_words()`, `count_chars()`, `count_words()`, `invert_rules()`, `l33t_rul35`, `map_text_chars()`, `map_text_coord()`, `map_text_regex()`, `read_ascii()`, `text_to_chars()`, `text_to_sentences()`, `text_to_words()`, `transl33t()`, `words_to_text()`

Examples

```
metachar
length(metachar)  # 12
nchar(paste0(metachar, collapse = ""))  # 12
```

`num_as_char` *Convert a number into a character sequence.*

Description

`num_as_char` converts a number into a character sequence (of a specific length).

Usage

```
num_as_char(x, n_pre_dec = 2, n_dec = 2, sym = "0", sep = ".")
```

Arguments

`x`	Number(s) to convert (required, accepts numeric vectors).
`n_pre_dec`	Number of digits before the decimal separator. Default: `n_pre_dec = 2`. This value is used to add zeros to the front of numbers. If the number of meaningful digits prior to decimal separator is greater than `n_pre_dec`, this value is ignored.
`n_dec`	Number of digits after the decimal separator. Default: `n_dec = 2`.
`sym`	Symbol to add to front or back. Default: `sym = 0`. Using `sym = " "` or `sym = "_"` can make sense, digits other than `"0"` do not.
`sep`	Decimal separator to use. Default: `sep = "."`.

Details

The arguments `n_pre_dec` and `n_dec` set a number of desired digits before and after the decimal separator `sep`. `num_as_char` tries to meet these digit numbers by adding zeros to the front and end of `x`. However, when `n_pre_dec` is lower than the number of relevant (pre-decimal) digits, all relevant digits are shown.

`n_pre_dec` also works for negative numbers, but the minus symbol is not counted as a (pre-decimal) digit.

Caveat: Note that this function illustrates how numbers, characters, `for` loops, and `paste()` can be combined when writing functions. It is not written efficiently or well.

See Also

Other utility functions: `is_equal()`, `is_vect()`, `is_wholenumber()`, `num_as_ordinal()`, `num_equal()`

Examples

```
num_as_char(1)
num_as_char(10/3)
num_as_char(1000/6)

# rounding down:
num_as_char((1.3333), n_pre_dec = 0, n_dec = 0)
num_as_char((1.3333), n_pre_dec = 2, n_dec = 0)
```

```
num_as_char((1.3333), n_pre_dec = 2, n_dec = 1)

# rounding up:
num_as_char(1.6666, n_pre_dec = 1, n_dec = 0)
num_as_char(1.6666, n_pre_dec = 1, n_dec = 1)
num_as_char(1.6666, n_pre_dec = 2, n_dec = 2)
num_as_char(1.6666, n_pre_dec = 2, n_dec = 3)

# Note: If n_pre_dec is too small, actual number is kept:
num_as_char(11.33, n_pre_dec = 0, n_dec = 1)
num_as_char(11.66, n_pre_dec = 1, n_dec = 1)

# Note:
num_as_char(1, sep = ",")
num_as_char(2, sym = " ")
num_as_char(3, sym = " ", n_dec = 0)

# for vectors:
num_as_char(1:10/1, n_pre_dec = 1, n_dec = 1)
num_as_char(1:10/3, n_pre_dec = 2, n_dec = 2)

# for negative numbers (adding relevant pre-decimals):
mix <- c(10.33, -10.33, 10.66, -10.66)
num_as_char(mix, n_pre_dec = 1, n_dec = 1)
num_as_char(mix, n_pre_dec = 1, n_dec = 0)

# Beware of bad inputs:
num_as_char(4, sym = "8")
num_as_char(5, sym = "99")
```

`num_as_ordinal` *Convert a number into an ordinal character sequence.*

Description

`num_as_ordinal` converts a given (cardinal) number into an ordinal character sequence.

Usage

```
num_as_ordinal(x, sep = "")
```

Arguments

`x`	Number(s) to convert (required, scalar or vector).
`sep`	Decimal separator to use. Default: `sep = ""` (i.e., no separator).

Details

The function currently only works for the English language and does not accepts inputs that are characters, dates, or times.

Note that the `toOrdinal()` function of the **toOrdinal** package works for multiple languages and provides a `toOrdinalDate()` function.

Caveat: Note that this function illustrates how numbers, characters, `for` loops, and `paste()` can be combined when writing functions. It is instructive, but not written efficiently or well (see the function definition for an alternative solution using vector indexing).

See Also

`toOrdinal()` function of the **toOrdinal** package.

Other utility functions: `is_equal()`, `is_vect()`, `is_wholenumber()`, `num_as_char()`, `num_equal()`

Examples

```
num_as_ordinal(1:4)
num_as_ordinal(10:14)    # all with "th"
num_as_ordinal(110:114)  # all with "th"
num_as_ordinal(120:124)  # 4 different suffixes
num_as_ordinal(1:15, sep = "-")  # using sep

# Note special cases:
num_as_ordinal(NA)
num_as_ordinal("1")
num_as_ordinal(Sys.Date())
num_as_ordinal(Sys.time())
num_as_ordinal(seq(1.99, 2.14, by = .01))
```

`num_equal` *Test two numeric vectors for pairwise (near) equality.*

Description

`num_equal` tests if two numeric vectors `x` and `y` are pairwise equal (within some tolerance value ‘tol‘).

Usage

```
num_equal(x, y, tol = .Machine$double.eps^0.5)
```

Arguments

`x`	1st numeric vector to compare (required, assumes a numeric vector).
`y`	2nd numeric vector to compare (required, assumes a numeric vector).
`tol`	Numeric tolerance value. Default: `tol = .Machine$double.eps^0.5` (see `?.Machine` for details).

Details

num_equal is a safer way to verify the (near) equality of numeric vectors than ==, as numbers may exhibit floating point effects.

See Also

is_equal function for generic vectors; all.equal function of the R **base** package; near function of the **dplyr** package.

Other utility functions: is_equal(), is_vect(), is_wholenumber(), num_as_char(), num_as_ordinal()

Examples

```
num_equal(2, sqrt(2)^2)

# Recycling:
num_equal(c(2, 3), c(sqrt(2)^2, sqrt(3)^2, 4/2, 9/3))

# Contrast:
.1 == .3/3
num_equal(.1, .3/3)

# Contrast:
v <- c(.9 - .8, .8 - .7, .7 - .6, .6 - .5,
       .5 - .4, .4 - .3, .3 - .2, .2 -.1, .1)
unique(v)
.1 == v
num_equal(.1, v)
```

---

outliers *Outlier data.*

Description

outliers is a fictitious dataset containing the id, sex, and height of 1000 non-existing, but otherwise normal people.

Usage

```
outliers
```

Format

A table with 100 cases (rows) and 3 variables (columns).

Details

Codebook

id Participant ID (as character code)

sex Gender (female vs. male)

height Height (in cm)

Source

See CSV data at http://rpository.com/ds4psy/data/out.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

pal_ds4psy *ds4psy default color palette.*

Description

pal_ds4psy provides a dedicated color palette.

Usage

```
pal_ds4psy
```

Format

An object of class data.frame with 1 rows and 11 columns.

Details

By default, pal_ds4psy is based on pal_unikn of the **unikn** package.

See Also

Other color objects and functions: pal_n_sq()

pal_n_sq — *Get n-by-n dedicated colors of a color palette.*

Description

pal_n_sq returns n^2 dedicated colors of a color palette pal (up to a maximum of n = "all" colors).

Usage

```
pal_n_sq(n = "all", pal = pal_ds4psy)
```

Arguments

n	The desired number colors of pal (as a number) or the character string "all" (to get all colors of pal). Default: n = "all".
pal	A color palette (as a data frame). Default: pal = pal_ds4psy.

Details

Use the more specialized function unikn::usecol for choosing n dedicated colors of a known color palette.

See Also

plot_tiles to plot tile plots.

Other color objects and functions: pal_ds4psy

Examples

```
pal_n_sq(1)  #  1 color: seeblau3
pal_n_sq(2)  #  4 colors
pal_n_sq(3)  #  9 colors (5: white)
pal_n_sq(4)  # 11 colors (6: white)
```

pi_100k — *Data: 100k digits of pi.*

Description

pi_100k is a dataset containing the first 100k digits of pi.

Usage

```
pi_100k
```

Format

A character of `nchar(pi_100k) = 100001`.

Source

See TXT data at http://rpository.com/ds4psy/data/pi_100k.txt.

Original data at http://www.geom.uiuc.edu/~huberty/math5337/groupe/digits.html.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

plot_charmap — *Plot a character map as a tile plot with text labels.*

Description

`plot_charmap` plots a character map and some aesthetics as a tile plot with text labels (using **ggplot2**).

Usage

```
plot_charmap(
  x = NA,
  file = "",
  lbl_tiles = TRUE,
  col_lbl = "black",
  angle = 0,
  cex = 3,
  fontface = 1,
  family = "sans",
  col_bg = "grey80",
  borders = FALSE,
  border_col = "white",
  border_size = 0.5
)
```

Arguments

x	A character map, as generated by `map_text_coord` or `map_text_regex` (as df). Alternatively, some text to map or plot (as a character vector). Different elements denote different lines of text. If `x = NA` (as per default), the `file` argument is used to read a text file or user input from the Console.

file	A text file to read (or its path). If file = "" (as per default), scan is used to read user input from the Console. If a text file is stored in a sub-directory, enter its path and name here (without any leading or trailing "." or "/").
lbl_tiles	Add character labels to tiles? Default: lbl_tiles = TRUE (i.e., show labels).
col_lbl	Default color of text labels (unless specified as a column col_fg of x). Default: col_lbl = "black".
angle	Default angle of text labels (unless specified as a column of x). Default: angle = 0.
cex	Character size (numeric). Default: cex = 3.
fontface	Font face of text labels (numeric). Default: fontface = 1, (from 1 to 4).
family	Font family of text labels (name). Default: family = "sans". Alternative options: "sans", "serif", or "mono".
col_bg	Default color to fill background tiles (unless specified as a column col_bg of x). Default: col_bg = "grey80".
borders	Boolean: Add borders to tiles? Default: borders = FALSE (i.e., no borders).
border_col	Color of tile borders. Default: border_col = "white".
border_size	Size of tile borders. Default: border_size = 0.5.

Details

plot_charmap is based on plot_chars. As it only contains the plotting-related parts, it assumes a character map generated by map_text_regex as input.

The plot generated by plot_charmap is character-based: Individual characters are plotted at equidistant x-y-positions and aesthetic variables are used for text labels and tile fill colors.

Value

A plot generated by **ggplot2**.

See Also

plot_chars for creating and plotting character maps; plot_text for plotting characters and color tiles by frequency; map_text_regex for mapping text to a character table and matching patterns; map_text_coord for mapping text to a table of character coordinates; read_ascii for reading text inputs into a character string; pal_ds4psy for default color palette.

Other plot functions: plot_chars(), plot_fn(), plot_fun(), plot_n(), plot_text(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# (0) Prepare:
ts <- c("Hello world!", "This is a test to test this splendid function",
        "Does this work?", "That's good.", "Please carry on.")
sum(nchar(ts))

# (1) From character map:
```

```
# (a) simple:
cm_1 <- map_text_coord(x = ts, flip_y = TRUE)
plot_charmap(cm_1)

# (b) pattern matching (regex):
cm_2 <- map_text_regex(ts, lbl_hi = "\\b\\w{4}\\b", bg_hi = "[good|test]",
                       lbl_rotate = "[^aeiou]", angle_fg = c(-45, +45))
plot_charmap(cm_2)

# (2) Alternative inputs:
# (a) From text string(s):
plot_charmap(ts)

# (b) From user input:
# plot_charmap()  # (enter text in Console)

# (c) From text file:
# cat("Hello world!", "This is a test file.",
#      "Can you see this text?",
#      "Good! Please carry on...",
#      file = "test.txt", sep = "\n")

# plot_charmap(file = "test.txt")

# unlink("test.txt")  # clean up (by deleting file).
```

`plot_chars` *Plot text characters (from file or user input) and match patterns.*

Description

`plot_chars` parses text (from a file or user input) into a table and then plots its individual characters as a tile plot (using **ggplot2**).

Usage

```
plot_chars(
  x = NA,
  file = "",
  lbl_hi = NA,
  lbl_lo = NA,
  bg_hi = NA,
  bg_lo = "[[:space:]]",
  lbl_rotate = NA,
  case_sense = TRUE,
  lbl_tiles = TRUE,
  angle_fg = c(-90, 90),
  angle_bg = 0,
```

```
  cex = 3,
  fontface = 1,
  family = "sans",
  col_lbl = "black",
  col_lbl_hi = pal_ds4psy[[1]],
  col_lbl_lo = pal_ds4psy[[9]],
  col_bg = pal_ds4psy[[7]],
  col_bg_hi = pal_ds4psy[[4]],
  col_bg_lo = "white",
  col_sample = FALSE,
  borders = FALSE,
  border_col = "white",
  border_size = 0.5
)
```

Arguments

x	The text to plot (as a character vector). Different elements denote different lines of text. If x = NA (as per default), the file argument is used to read a text file or user input from the Console.
file	A text file to read (or its path). If file = "" (as per default), scan is used to read user input from the Console. If a text file is stored in a sub-directory, enter its path and name here (without any leading or trailing "." or "/").
lbl_hi	Labels to highlight (as regex). Default: lbl_hi = NA.
lbl_lo	Labels to de-emphasize (as regex). Default: lbl_lo = NA.
bg_hi	Background tiles to highlight (as regex). Default: bg_hi = NA.
bg_lo	Background tiles to de-emphasize (as regex). Default: bg_lo = "[[:space:]]".
lbl_rotate	Labels to rotate (as regex). Default: lbl_rotate = NA.
case_sense	Boolean: Distinguish lower- vs. uppercase characters in pattern matches? Default: case_sense = TRUE.
lbl_tiles	Add character labels to tiles? Default: lbl_tiles = TRUE (i.e., show labels).
angle_fg	Angle(s) for rotating character labels matching the pattern of the lbl_rotate expression. Default: angle_fg = c(-90,90). If length(angle_fg) > 1, a random value in uniform range(angle_fg) is used for every character.
angle_bg	Angle(s) of rotating character labels not matching the pattern of the lbl_rotate expression. Default: angle_bg = 0 (i.e., no rotation). If length(angle_bg) > 1, a random value in uniform range(angle_bg) is used for every character.
cex	Character size (numeric). Default: cex = 3.
fontface	Font face of text labels (numeric). Default: fontface = 1, (from 1 to 4).
family	Font family of text labels (name). Default: family = "sans". Alternative options: "sans", "serif", or "mono".
col_lbl	Default color of text labels. Default: col_lbl = "black".
col_lbl_hi	Highlighting color of text labels. Default: col_lbl_hi = pal_ds4psy[[1]].
col_lbl_lo	De-emphasizing color of text labels. Default: col_lbl_lo = pal_ds4psy[[9]].

col_bg	Default color to fill background tiles. Default: col_bg = pal_ds4psy[[7]].
col_bg_hi	Highlighting color to fill background tiles. Default: col_bg_hi = pal_ds4psy[[4]].
col_bg_lo	De-emphasizing color to fill background tiles. Default: col_bg_lo = "white".
col_sample	Boolean: Sample color vectors (within category)? Default: col_sample = FALSE.
borders	Boolean: Add borders to tiles? Default: borders = FALSE (i.e., no borders).
border_col	Color of tile borders. Default: border_col = "white".
border_size	Size of tile borders. Default: border_size = 0.5.

Details

plot_chars blurs the boundary between a text and its graphical representation by combining options for matching patterns of text with visual features for displaying characters (e.g., their color or orientation).

plot_chars is based on plot_text, but provides additional support for detecting and displaying characters (i.e., text labels, their orientation, and color options) based on matching regular expression (regex).

Internally, plot_chars is a wrapper that calls (1) map_text_regex for creating a character map (allowing for matching patterns for some aesthetics) and (2) plot_charmap for plotting this character map.

However, in contrast to plot_charmap, plot_chars invisibly returns a description of the plot (as a data frame).

The plot generated by plot_chars is character-based: Individual characters are plotted at equidistant x-y-positions and the aesthetic settings provided for text labels and tile fill colors.

Five regular expressions and corresponding color and angle arguments allow identifying, marking (highlighting or de-emphasizing), and rotating those sets of characters (i.e, their text labels or fill colors). that match the provided patterns.

Value

An invisible data frame describing the plot.

See Also

plot_charmap for plotting character maps; plot_text for plotting characters and color tiles by frequency; map_text_coord for mapping text to a table of character coordinates; map_text_regex for mapping text to a character table and matching patterns; read_ascii for reading text inputs into a character string; pal_ds4psy for default color palette.

Other plot functions: plot_charmap(), plot_fn(), plot_fun(), plot_n(), plot_text(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# (A) From text string(s):
plot_chars(x = c("Hello world!", "Does this work?",
                 "That's good.", "Please carry on..."))
```

```
# (B) From user input:
# plot_chars()  # (enter text in Console)

# (C) From text file:
# Create and use a text file:
# cat("Hello world!", "This is a test file.",
#     "Can you see this text?",
#     "Good! Please carry on...",
#     file = "test.txt", sep = "\n")

# plot_chars(file = "test.txt")  # default
# plot_chars(file = "test.txt", lbl_hi = "[[:upper:]]", lbl_lo = "[[:punct:]]",
#            col_lbl_hi = "red", col_lbl_lo = "blue")

# plot_chars(file = "test.txt", lbl_hi = "[aeiou]", col_lbl_hi = "red",
#            col_bg = "white", bg_hi = "see")  # mark vowels and "see" (in bg)
# plot_chars(file = "test.txt", bg_hi = "[aeiou]", col_bg_hi = "gold")  # mark (bg of) vowels

## Label options:
# plot_chars(file = "test.txt", bg_hi = "see", lbl_tiles = FALSE)
# plot_chars(file = "test.txt", cex = 5, family = "mono", fontface = 4, lbl_angle = c(-20, 20))

## Note: plot_chars() invisibly returns a description of the plot (as df):
# tb <- plot_chars(file = "test.txt", lbl_hi = "[aeiou]", lbl_rotate = TRUE)
# head(tb)

# unlink("test.txt")  # clean up (by deleting file).


## (B) From text file (in subdir):
# plot_chars(file = "data-raw/txt/hello.txt")  # requires txt file
# plot_chars(file = "data-raw/txt/ascii.txt", lbl_hi = "[2468]", bg_lo = "[[:digit:]]",
#            col_lbl_hi = "red", cex = 10, fontface = 2)

## (C) User input:
# plot_chars()  # (enter text in Console)
```

`plot_fn` *A function to plot a plot.*

Description

`plot_fn` is a function that uses parameters for plotting a plot.

Usage

```
plot_fn(
  x = NA,
```

```
  y = 1,
  A = TRUE,
  B = FALSE,
  C = TRUE,
  D = FALSE,
  E = FALSE,
  F = FALSE,
  f = c(rev(pal_seeblau), "white", pal_pinky),
  g = "white"
)
```

Arguments

x	A (natural) number. Default: x = NA.
y	A (decimal) number. Default: y = 1.
A	Boolean. Default: A = TRUE.
B	Boolean. Default: B = FALSE.
C	Boolean. Default: C = TRUE.
D	Boolean. Default: D = FALSE.
E	Boolean. Default: E = FALSE.
F	Boolean. Default: F = FALSE.
f	A color palette (e.g., as a vector). Default: f = c(rev(pal_seeblau),"white",pal_pinky). Note: Using colors of the unikn package by default.
g	A color (e.g., as a character). Default: g = "white".

Details

plot_fn is deliberately kept cryptic and obscure to illustrate how function parameters can be explored.

plot_fn also shows that brevity in argument names should not come at the expense of clarity. In fact, transparent argument names are absolutely essential for understanding and using a function.

plot_fn currently requires pal_seeblau and pal_pinky (from the **unikn** package) for its default colors.

See Also

plot_fun for a related function; pal_ds4psy for color palette.

Other plot functions: plot_charmap(), plot_chars(), plot_fun(), plot_n(), plot_text(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# Basics:
plot_fn()

# Exploring options:
```

```
plot_fn(x = 2, A = TRUE)
plot_fn(x = 3, A = FALSE, E = TRUE)
plot_fn(x = 4, A = TRUE,  B = TRUE, D = TRUE)
plot_fn(x = 5, A = FALSE, B = TRUE, E = TRUE, f = c("black", "white", "gold"))
plot_fn(x = 7, A = TRUE,  B = TRUE, F = TRUE, f = c("steelblue", "white", "forestgreen"))
```

---

plot_fun *Another function to plot some plot.*

Description

plot_fun is a function that provides options for plotting a plot.

Usage

```
plot_fun(
  a = NA,
  b = TRUE,
  c = TRUE,
  d = 1,
  e = FALSE,
  f = FALSE,
  g = FALSE,
  c1 = c(rev(pal_seeblau), "white", pal_grau, "black", Bordeaux),
  c2 = "black"
)
```

Arguments

a	A (natural) number. Default: a = NA.
b	Boolean. Default: b = TRUE.
c	Boolean. Default: c = TRUE.
d	A (decimal) number. Default: d = 1.0.
e	Boolean. Default: e = FALSE.
f	Boolean. Default: f = FALSE.
g	Boolean. Default: g = FALSE.
c1	A color palette (e.g., as a vector). Default: c1 = c(rev(pal_seeblau),"white",pal_grau,"black",Bo Note: Using colors of the unikn package by default.
c2	A color (e.g., as a character). Default: c2 = "black".

Details

plot_fun is deliberately kept cryptic and obscure to illustrate how function parameters can be explored.

plot_fun also shows that brevity in argument names should not come at the expense of clarity. In fact, transparent argument names are absolutely essential for understanding and using a function.

plot_fun currently requires pal_seeblau, pal_grau, and Bordeaux (from the **unikn** package) for its default colors.

See Also

plot_fn for a related function; pal_ds4psy for color palette.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_n(), plot_text(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# Basics:
plot_fun()

# Exploring options:
plot_fun(a = 3, b = FALSE, e = TRUE)
plot_fun(a = 4, f = TRUE, g = TRUE, c1 = c("steelblue", "white", "firebrick"))
```

---

plot_n *Plot n tiles.*

Description

plot_n plots a row or column of n tiles on fixed or polar coordinates.

Usage

```
plot_n(
  n = NA,
  row = TRUE,
  polar = FALSE,
  pal = pal_ds4psy,
  sort = TRUE,
  borders = TRUE,
  border_col = "black",
  border_size = 0,
  lbl_tiles = FALSE,
  lbl_title = FALSE,
  rseed = NA,
  save = FALSE,
```

```
  save_path = "images/tiles",
  prefix = "",
  suffix = ""
)
```

Arguments

n	Basic number of tiles (on either side).
row	Plot as a row? Default: row = TRUE (else plotted as a column).
polar	Plot on polar coordinates? Default: polar = FALSE (i.e., using fixed coordinates).
pal	A color palette (automatically extended to n colors). Default: pal = pal_ds4psy.
sort	Sort tiles? Default: sort = TRUE (i.e., sorted tiles).
borders	Add borders to tiles? Default: borders = TRUE (i.e., use borders).
border_col	Color of borders (if borders = TRUE). Default: border_col = "black".
border_size	Size of borders (if borders = TRUE). Default: border_size = 0 (i.e., invisible).
lbl_tiles	Add numeric labels to tiles? Default: lbl_tiles = FALSE (i.e., no labels).
lbl_title	Add numeric label (of n) to plot? Default: lbl_title = FALSE (i.e., no title).
rseed	Random seed (number). Default: rseed = NA (using random seed).
save	Save plot as png file? Default: save = FALSE.
save_path	Path to save plot (if save = TRUE). Default: save_path = "images/tiles".
prefix	Prefix to plot name (if save = TRUE). Default: prefix = "".
suffix	Suffix to plot name (if save = TRUE). Default: suffix = "".

Details

Note that a polar row makes a tasty pie, whereas a polar column makes a target plot.

See Also

pal_ds4psy for default color palette.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_fun(), plot_text(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# (1) Basics (as ROW or COL):
plot_n()  # default plot (random n, row = TRUE, with borders, no labels)
plot_n(row = FALSE)  # default plot (random n, with borders, no labels)

plot_n(n = 4, sort = FALSE)      # random order
plot_n(n = 6, borders = FALSE)   # no borders
plot_n(n = 8, lbl_tiles = TRUE,  # with tile +
       lbl_title = TRUE)         # title labels

# Set colors:
```

```
plot_n(n = 5, row = TRUE, lbl_tiles = TRUE, lbl_title = TRUE,
       pal = c("orange", "white", "firebrick"),
       border_col = "white", border_size = 2)

# Fixed rseed:
plot_n(n = 4, sort = FALSE, borders = FALSE,
       lbl_tiles = TRUE, lbl_title = TRUE, rseed = 101)

# (2) polar plot (as PIE or TARGET):
plot_n(polar = TRUE)  # PIE plot (with borders, no labels)
plot_n(polar = TRUE, row = FALSE)  # TARGET plot (with borders, no labels)

plot_n(n = 4, polar = TRUE, sort = FALSE)      # PIE in random order
plot_n(n = 5, polar = TRUE, row = FALSE, borders = FALSE)   # TARGET no borders
plot_n(n = 5, polar = TRUE, lbl_tiles = TRUE)  # PIE with tile labels
plot_n(n = 5, polar = TRUE, row = FALSE, lbl_title = TRUE)  # TARGET with title label

# plot_n(n = 4, row = TRUE, sort = FALSE, borders = TRUE,
#        border_col = "white", border_size = 2,
#        polar = TRUE, rseed = 132)
# plot_n(n = 4, row = FALSE, sort = FALSE, borders = TRUE,
#        border_col = "white", border_size = 2,
#        polar = TRUE, rseed = 134)
```

`plot_text` *Plot text characters (from file or user input).*

Description

`plot_text` parses text (from a file or from user input) and plots its individual characters as a tile plot (using **ggplot2**).

Usage

```
plot_text(
  x = NA,
  file = "",
  char_bg = " ",
  lbl_tiles = TRUE,
  lbl_rotate = FALSE,
  cex = 3,
  fontface = 1,
  family = "sans",
  col_lbl = "black",
  col_bg = "white",
  pal = pal_ds4psy[1:5],
  pal_extend = TRUE,
  case_sense = FALSE,
```

```
  borders = TRUE,
  border_col = "white",
  border_size = 0.5
)
```

Arguments

x	The text to plot (as a character vector). Different elements denote different lines of text. If x = NA (as per default), the file argument is used to read a text file or scan user input (entering text in Console).
file	A text file to read (or its path). If file = "" (as per default), scan is used to read user input from the Console. If a text file is stored in a sub-directory, enter its path and name here (without any leading or trailing "." or "/").
char_bg	Character used as background. Default: char_bg = " ". If char_bg = NA, the most frequent character is used.
lbl_tiles	Add character labels to tiles? Default: lbl_tiles = TRUE (i.e., show labels).
lbl_rotate	Rotate character labels? Default: lbl_rotate = FALSE (i.e., no rotation).
cex	Character size (numeric). Default: cex = 3.
fontface	Font face of text labels (numeric). Default: fontface = 1, (from 1 to 4).
family	Font family of text labels (name). Default: family = "sans". Alternative options: "sans", "serif", or "mono".
col_lbl	Color of text labels. Default: col_lbl = "black" (if lbl_tiles = TRUE).
col_bg	Color of char_bg (if defined), or the most frequent character in text (typically " "). Default: col_bg = "white".
pal	Color palette for filling tiles of text (used in order of character frequency). Default: pal = pal_ds4psy[1:5] (i.e., shades of Seeblau).
pal_extend	Boolean: Should pal be extended to match the number of different characters in text? Default: pal_extend = TRUE. If pal_extend = FALSE, only the tiles of the length(pal) most frequent characters will be filled by the colors of pal.
case_sense	Boolean: Distinguish lower- vs. uppercase characters? Default: case_sense = FALSE.
borders	Boolean: Add borders to tiles? Default: borders = TRUE (i.e., use borders).
border_col	Color of borders (if borders = TRUE). Default: border_col = "white".
border_size	Size of borders (if borders = TRUE). Default: border_size = 0.5.

Details

plot_text blurs the boundary between a text and its graphical representation by adding visual options for coloring characters based on their frequency counts. (Note that plot_chars provides additional support for matching regular expressions.)

plot_text is character-based: Individual characters are plotted at equidistant x-y-positions with color settings for text labels and tile fill colors.

By default, the color palette pal (used for tile fill colors) is scaled to indicate character frequency.

plot_text invisibly returns a description of the plot (as a data frame).

Value

An invisible data frame describing the plot.

See Also

plot_charmap for plotting character maps; plot_chars for creating and plotting character maps; map_text_coord for mapping text to a table of character coordinates; map_text_regex for mapping text to a character table and matching patterns; read_ascii for parsing text from file or user input; pal_ds4psy for default color palette.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_fun(), plot_n(), plot_tiles(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# (A) From text string(s):
plot_text(x = c("Hello", "world!"))
plot_text(x = c("Hello world!", "How are you today?"))

# (B) From user input:
# plot_text()  # (enter text in Console)

# (C) From text file:
## Create a temporary file "test.txt":
# cat("Hello world!", "This is a test file.",
#     "Can you see this text?",
#     "Good! Please carry on...",
#     file = "test.txt", sep = "\n")

# plot_text(file = "test.txt")

## Set colors, pal_extend, and case_sense:
# cols <- c("steelblue", "skyblue", "lightgrey")
# cols <- c("firebrick", "olivedrab", "steelblue", "orange", "gold")
# plot_text(file = "test.txt", pal = cols, pal_extend = TRUE)
# plot_text(file = "test.txt", pal = cols, pal_extend = FALSE)
# plot_text(file = "test.txt", pal = cols, pal_extend = FALSE, case_sense = TRUE)

## Customize text and grid options:
# plot_text(file = "test.txt", col_lbl = "darkblue", cex = 4, family = "sans", fontface = 3,
#           pal = "gold1", pal_extend = TRUE, border_col = NA)
# plot_text(file = "test.txt", family = "serif", cex = 6, lbl_rotate = TRUE,
#           pal = NA, borders = FALSE)
# plot_text(file = "test.txt", col_lbl = "white", pal = c("green3", "black"),
#           border_col = "black", border_size = .2)

## Color ranges:
# plot_text(file = "test.txt", pal = c("red2", "orange", "gold"))
# plot_text(file = "test.txt", pal = c("olivedrab4", "gold"))

# unlink("test.txt")  # clean up.
```

```
## (B) From text file (in subdir):
# plot_text(file = "data-raw/txt/hello.txt")  # requires txt file
# plot_text(file = "data-raw/txt/ascii.txt", cex = 5,
#           col_bg = "grey", char_bg = "-")

## (C) From user input:
# plot_text()  # (enter text in Console)
```

---

plot_tiles *Plot n-by-n tiles.*

Description

plot_tiles plots an area of n-by-n tiles on fixed or polar coordinates.

Usage

```
plot_tiles(
  n = NA,
  pal = pal_ds4psy,
  sort = TRUE,
  borders = TRUE,
  border_col = "black",
  border_size = 0.2,
  lbl_tiles = FALSE,
  lbl_title = FALSE,
  polar = FALSE,
  rseed = NA,
  save = FALSE,
  save_path = "images/tiles",
  prefix = "",
  suffix = ""
)
```

Arguments

n	Basic number of tiles (on either side).
pal	Color palette (automatically extended to n x n colors). Default: pal = pal_ds4psy.
sort	Boolean: Sort tiles? Default: sort = TRUE (i.e., sorted tiles).
borders	Boolean: Add borders to tiles? Default: borders = TRUE (i.e., use borders).
border_col	Color of borders (if borders = TRUE). Default: border_col = "black".
border_size	Size of borders (if borders = TRUE). Default: border_size = 0.2.

lbl_tiles	Boolean: Add numeric labels to tiles? Default: lbl_tiles = FALSE (i.e., no labels).
lbl_title	Boolean: Add numeric label (of n) to plot? Default: lbl_title = FALSE (i.e., no title).
polar	Boolean: Plot on polar coordinates? Default: polar = FALSE (i.e., using fixed coordinates).
rseed	Random seed (number). Default: rseed = NA (using random seed).
save	Boolean: Save plot as png file? Default: save = FALSE.
save_path	Path to save plot (if save = TRUE). Default: save_path = "images/tiles".
prefix	Prefix to plot name (if save = TRUE). Default: prefix = "".
suffix	Suffix to plot name (if save = TRUE). Default: suffix = "".

See Also

pal_ds4psy for default color palette.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_fun(), plot_n(), plot_text(), theme_clean(), theme_ds4psy(), theme_empty()

Examples

```
# (1) Tile plot:
plot_tiles()  # default plot (random n, with borders, no labels)

plot_tiles(n = 4, sort = FALSE)      # random order
plot_tiles(n = 6, borders = FALSE)   # no borders
plot_tiles(n = 8, lbl_tiles = TRUE,  # with tile +
           lbl_title = TRUE)         # title labels

# Set colors:
plot_tiles(n = 4, pal = c("orange", "white", "firebrick"),
           lbl_tiles = TRUE, lbl_title = TRUE,
           sort = TRUE)
plot_tiles(n = 6, sort = FALSE, border_col = "white", border_size = 2)

# Fixed rseed:
plot_tiles(n = 4, sort = FALSE, borders = FALSE,
           lbl_tiles = TRUE, lbl_title = TRUE,
           rseed = 101)

# (2) polar plot:
plot_tiles(polar = TRUE)  # default polar plot (with borders, no labels)

plot_tiles(n = 4, polar = TRUE, sort = FALSE)   # random order
plot_tiles(n = 6, polar = TRUE, sort = TRUE,    # sorted and with
           lbl_tiles = TRUE, lbl_title = TRUE)  # tile + title labels
plot_tiles(n = 4, sort = FALSE, borders = TRUE,
           border_col = "white", border_size = 2,
           polar = TRUE, rseed = 132)           # fixed rseed
```

`posPsy_AHI_CESD` *Positive Psychology: AHI CESD data.*

Description

`posPsy_AHI_CESD` is a dataset containing answers to the 24 items of the Authentic Happiness Inventory (AHI) and answers to the 20 items of the Center for Epidemiological Studies Depression (CES-D) scale (Radloff, 1977) for multiple (1 to 6) measurement occasions.

Usage

```
posPsy_AHI_CESD
```

Format

A table with 992 cases (rows) and 50 variables (columns).

Details

Codebook

- 1. **id**: Participant ID.
- 2. **occasion**: Measurement occasion: 0: Pretest (i.e., at enrolment), 1: Posttest (i.e., 7 days after pretest), 2: 1-week follow-up, (i.e., 14 days after pretest, 7 days after posttest), 3: 1-month follow-up, (i.e., 38 days after pretest, 31 days after posttest), 4: 3-month follow-up, (i.e., 98 days after pretest, 91 days after posttest), 5: 6-month follow-up, (i.e., 189 days after pretest, 182 days after posttest).
- 3. **elapsed.days**: Time since enrolment measured in fractional days.
- 4. **intervention**: Type of intervention: 3 positive psychology interventions (PPIs), plus 1 control condition: 1: "Using signature strengths", 2: "Three good things", 3: "Gratitude visit", 4: "Recording early memories" (control condition).
- 5.-28. (from **ahi01** to **ahi24**): Responses on 24 AHI items.
- 29.-48. (from **cesd01** to **cesd20**): Responses on 20 CES-D items.
- 49. **ahiTotal**: Total AHI score.
- 50. **cesdTotal**: Total CES-D score.

See codebook and references at `https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html`.

Source

Articles

- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R., & Schüz, B. (2017). Web-based positive psychology interventions: A reexamination of effectiveness. *Journal of Clinical Psychology*, *73*(3), 218–232. doi: `10.1002/jclp.22328`

- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R. and Schüz, B. (2018). Data from, 'Web-based positive psychology interventions: A reexamination of effectiveness'. *Journal of Open Psychology Data*, *6*(1). doi: `10.5334/jopd.35`

See `https://openpsychologydata.metajnl.com/articles/10.5334/jopd.35/` for details and doi: 10.6084/m9.figshare.1577563.v1 for original dataset.

Additional references at `https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html`.

See Also

`posPsy_long` for a corrected version of this file (in long format).

Other datasets: `Bushisms`, `Trumpisms`, `countries`, `data_1`, `data_2`, `data_t1_de`, `data_t1_tab`, `data_t1`, `data_t2`, `data_t3`, `data_t4`, `dt_10`, `exp_num_dt`, `exp_wide`, `falsePosPsy_all`, `fame`, `flowery`, `fruits`, `outliers`, `pi_100k`, `posPsy_long`, `posPsy_p_info`, `posPsy_wide`, `t3`, `t4`, `t_1`, `t_2`, `t_3`, `t_4`, `table6`, `table7`, `table8`, `table9`, `tb`

`posPsy_long` *Positive Psychology: AHI CESD corrected data (in long format).*

Description

`posPsy_long` is a dataset containing answers to the 24 items of the Authentic Happiness Inventory (AHI) and answers to the 20 items of the Center for Epidemiological Studies Depression (CES-D) scale (see Radloff, 1977) for multiple (1 to 6) measurement occasions.

Usage

```
posPsy_long
```

Format

A table with 990 cases (rows) and 50 variables (columns).

Details

This dataset is a corrected version of `posPsy_AHI_CESD` and in long-format.

Source

Articles

- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R., & Schüz, B. (2017). Web-based positive psychology interventions: A reexamination of effectiveness. *Journal of Clinical Psychology*, *73*(3), 218–232. doi: `10.1002/jclp.22328`
- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R. and Schüz, B. (2018). Data from, 'Web-based positive psychology interventions: A reexamination of effectiveness'. *Journal of Open Psychology Data*, *6*(1). doi: `10.5334/jopd.35`

See https://openpsychologydata.metajnl.com/articles/10.5334/jopd.35/ for details and doi: 10.6084/m9.figshare.1577563.v1 for original dataset.

Additional references at https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html.

See Also

posPsy_AHI_CESD for source of this file and codebook information; posPsy_wide for a version of this file (in wide format).

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

posPsy_p_info *Positive Psychology: Participant data.*

Description

posPsy_p_info is a dataset containing details of 295 participants.

Usage

```
posPsy_p_info
```

Format

A table with 295 cases (rows) and 6 variables (columns).

Details

id Participant ID.

intervention Type of intervention: 3 positive psychology interventions (PPIs), plus 1 control condition: 1: "Using signature strengths", 2: "Three good things", 3: "Gratitude visit", 4: "Recording early memories" (control condition).

sex Sex: 1 = female, 2 = male.

age Age (in years).

educ Education level: Scale from 1: less than 12 years, to 5: postgraduate degree.

income Income: Scale from 1: below average, to 3: above average.

See codebook and references at https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html.

Source

Articles

- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R., & Schüz, B. (2017). Web-based positive psychology interventions: A reexamination of effectiveness. *Journal of Clinical Psychology*, *73*(3), 218–232. doi: 10.1002/jclp.22328
- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R. and Schüz, B. (2018). Data from, 'Web-based positive psychology interventions: A reexamination of effectiveness'. *Journal of Open Psychology Data*, *6*(1). doi: 10.5334/jopd.35

See https://openpsychologydata.metajnl.com/articles/10.5334/jopd.35/ for details and doi: 10.6084/m9.figshare.1577563.v1 for original dataset.

Additional references at https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

`posPsy_wide` *Positive Psychology: All corrected data (in wide format).*

Description

`posPsy_wide` is a dataset containing answers to the 24 items of the Authentic Happiness Inventory (AHI) and answers to the 20 items of the Center for Epidemiological Studies Depression (CES-D) scale (see Radloff, 1977) for multiple (1 to 6) measurement occasions.

Usage

```
posPsy_wide
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 295 rows and 294 columns.

Details

This dataset is based on posPsy_AHI_CESD and posPsy_long, but is in wide format.

Source

Articles

- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R., & Schüz, B. (2017). Web-based positive psychology interventions: A reexamination of effectiveness. *Journal of Clinical Psychology*, *73*(3), 218–232. doi: 10.1002/jclp.22328
- Woodworth, R. J., O'Brien-Malone, A., Diamond, M. R. and Schüz, B. (2018). Data from, 'Web-based positive psychology interventions: A reexamination of effectiveness'. *Journal of Open Psychology Data*, *6*(1). doi: 10.5334/jopd.35

See https://openpsychologydata.metajnl.com/articles/10.5334/jopd.35/ for details and doi: 10.6084/m9.figshare.1577563.v1 for original dataset.

Additional references at https://bookdown.org/hneth/ds4psy/B-1-datasets-pos.html.

See Also

posPsy_AHI_CESD for the source of this file, posPsy_long for a version of this file (in long format).

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

read_ascii   *read_ascii parses text (from file or user input) into string(s) of text.*

Description

read_ascii parses text inputs (from a file or from user input in the Console) into a character vector.

Usage

```
read_ascii(file = "", quiet = FALSE)
```

Arguments

file	The text file to read (or its path). If file = "" (the default), scan is used to read user input from the Console. If a text file is stored in a sub-directory, enter its path and name here (without any leading or trailing "." or "/"). Default: file = "".
quiet	Boolean: Provide user feedback? Default: quiet = FALSE.

Details

Different lines of text are represented by different elements of the character vector returned.

The getwd function is used to determine the current working directory. This replaces the **here** package, which was previously used to determine an (absolute) file path.

Note that read_ascii originally contained map_text_coord, but has been separated to enable independent access to separate functionalities.

Value

A character vector, with its elements denoting different lines of text.

See Also

map_text_coord for mapping text to a table of character coordinates; plot_chars for a character plotting function.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
## Create a temporary file "test.txt":
# cat("Hello world!", "This is a test.",
#     "Can you see this text?",
#     "Good! Please carry on...",
#     file = "test.txt", sep = "\n")

## (a) Read text (from file):
# read_ascii("test.txt")
# read_ascii("test.txt", quiet = TRUE)  # y flipped

# unlink("test.txt")  # clean up (by deleting file).


## (b) Read text (from file in subdir):
# read_ascii("data-raw/txt/ascii.txt")  # requires txt file

## (c) Scan user input (from console):
# read_ascii()
```

---

sample_char   *Draw a sample of n random characters (from given characters).*

Description

sample_char draws a sample of n random characters from a given range of characters.

Usage

```
sample_char(x_char = c(letters, LETTERS), n = 1, replace = FALSE, ...)
```

Arguments

`x_char`	Population of characters to sample from. Default: `x_char = c(letters,LETTERS)`.
`n`	Number of characters to draw. Default: `n = 1`.
`replace`	Boolean: Sample with replacement? Default: `replace = FALSE`.
`...`	Other arguments. (Use for specifying `prob`, as passed to `sample()`.)

Details

By default, `sample_char` draws `n = 1` a random alphabetic character from `x_char = c(letters,LETTERS)`.

As with `sample()`, the sample size `n` must not exceed the number of available characters `nchar(x_char)`, unless `replace = TRUE` (i.e., sampling with replacement).

Value

A text string (scalar character vector).

See Also

Other sampling functions: `coin()`, `dice_2()`, `dice()`, `sample_date()`, `sample_time()`

Examples

```
sample_char()  # default
sample_char(n = 10)
sample_char(x_char = "abc", n = 10, replace = TRUE)
sample_char(x_char = c("x y", "6 9"), n =  6, replace = FALSE)
sample_char(x_char = c("x y", "6 9"), n = 20, replace = TRUE)

# Biased sampling:
sample_char(x_char = "abc", n = 20, replace = TRUE,
             prob = c(3/6, 2/6, 1/6))

# Note: By default, n must not exceed nchar(x_char):
sample_char(n = 52, replace = FALSE)    # works, but
# sample_char(n = 53, replace = FALSE)  # would yield ERROR;
sample_char(n = 53, replace = TRUE)     # works again.
```

`sample_date` *Draw a sample of n random dates (from a given range).*

Description

`sample_date` draws a sample of `n` random dates from a given range.

Usage

```
sample_date(from = "1970-01-01", to = Sys.Date(), size = 1, ...)
```

Arguments

`from`	Earliest date (as "Date" or string). Default: `from = "1970-01-01"` (as a scalar).
`to`	Latest date (as "Date" or string). Default: `to = Sys.Date()` (as a scalar).
`size`	Size of date samples to draw. Default: `size = 1`.
`...`	Other arguments. (Use for specifying `replace`, as passed to `sample()`.)

Details

By default, `sample_date` draws `n = 1` random date (as a "Date" object) in the range `from = "1970-01-01"` `to = Sys.Date()` (current date).

Both `from` and `to` currently need to be scalars (i.e., with a length of 1).

Value

A vector of class "Date".

See Also

Other sampling functions: `coin()`, `dice_2()`, `dice()`, `sample_char()`, `sample_time()`

Examples

```
sample_date()
sort(sample_date(size = 10))
sort(sample_date(from = "2020-02-28", to = "2020-03-01",
     size = 10, replace = TRUE))  # 2020 is a leap year

# Note: Oddity with sample():
sort(sample_date(from = "2020-01-01", to = "2020-01-01", size = 10, replace = TRUE))  # range of 0!
# see sample(9:9, size = 10, replace = TRUE)
```

`sample_time` *Draw a sample of n random times (from a given range).*

Description

`sample_time` draws a sample of `n` random times from a given range.

Usage

```
sample_time(
  from = "1970-01-01 00:00:00",
  to = Sys.time(),
  size = 1,
  as_POSIXct = TRUE,
  tz = "",
  ...
)
```

Arguments

from	Earliest date-time (as string). Default: from = "1970-01-01 00:00:00" (as a scalar).
to	Latest date-time (as string). Default: to = Sys.time() (as a scalar).
size	Size of time samples to draw. Default: size = 1.
as_POSIXct	Boolean: Return calendar time ("POSIXct") object? Default: as_POSIXct = TRUE. If as_POSIXct = FALSE, a local time ("POSIXlt") object is returned (as a list).
tz	Time zone. Default: tz = "" (i.e., current system time zone, see Sys.timezone()). Use tz = "UTC" for Universal Time, Coordinated.
...	Other arguments. (Use for specifying replace, as passed to sample().)

Details

By default, sample_time draws n = 1 random calendar time (as a "POSIXct" object) in the range from = "1970-01-01 00:00:00" to = Sys.time() (current time).

Both from and to currently need to be scalars (i.e., with a length of 1).

If as_POSIXct = FALSE, a local time ("POSIXlt") object is returned (as a list).

The tz argument allows specifying time zones (see Sys.timezone() for current setting and OlsonNames() for options.)

Value

A vector of class "POSIXct" or "POSIXlt".

See Also

Other sampling functions: coin(), dice_2(), dice(), sample_char(), sample_date()

Examples

```
# Basics:
sample_time()
sample_time(size = 10)

# Specific ranges:
sort(sample_time(from = (Sys.time() - 60), size = 10))  # within last minute
sort(sample_time(from = (Sys.time() - 1 * 60 * 60), size = 10))  # within last hour
sort(sample_time(from = Sys.time(), to = (Sys.time() + 1 * 60 * 60),
     size = 10, replace = FALSE))  # within next hour
sort(sample_time(from = "2020-12-31 00:00:00 CET", to = "2020-12-31 00:00:01 CET",
                 size = 10, replace = TRUE))  # within 1 sec range

# Local time (POSIXlt) objects (as list):
(lt_sample <- sample_time(as_POSIXct = FALSE))
unlist(lt_sample)

# Time zones:
```

```
sample_time(size = 3, tz = "UTC")
sample_time(size = 3, tz = "US/Pacific")

# Note: Oddity with sample():
sort(sample_time(from = "2020-12-31 00:00:00 CET", to = "2020-12-31 00:00:00 CET",
     size = 10, replace = TRUE))  # range of 0!
# see sample(9:9, size = 10, replace = TRUE)
```

---

t3 *Data table t3.*

Description

t3 is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

t3

Format

A table with 10 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t3.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

t4 *Data table t4.*

Description

t4 is a fictitious dataset to practice importing and joining data (from a CSV file).

Usage

t4

Format

A table with 10 cases (rows) and 4 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t4.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

---

table6 *Data table6.*

Description

table6 is a fictitious dataset to practice reshaping and tidying data.

Usage

```
table6
```

Format

A table with 6 cases (rows) and 2 variables (columns).

Details

This dataset is a further variant of the table1 to table5 datasets of the **tidyr** package.

Source

See CSV data at http://rpository.com/ds4psy/data/table6.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table7, table8, table9, tb table7 *Data table7.*

Description

`table7` is a fictitious dataset to practice reshaping and tidying data.

Usage

```
table7
```

Format

A table with 6 cases (rows) and 1 (horrendous) variable (column).

Details

This dataset is a further variant of the `table1` to `table5` datasets of the **tidyr** package.

Source

See CSV data at http://rpository.com/ds4psy/data/table7.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table8, table9, tb

table8 *Data table8.*

Description

`table9` is a fictitious dataset to practice reshaping and tidying data.

Usage

```
table8
```

Format

A table with 3 cases (rows) and 5 variables (columns).

Details

This dataset is a further variant of the table1 to table5 datasets of the **tidyr** package.

Source

See CSV data at http://rpository.com/ds4psy/data/table8.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table9, tb

---

table9 *Data table9.*

Description

table9 is a fictitious dataset to practice reshaping and tidying data.

Usage

```
table9
```

Format

A 3 x 2 x 2 array (of type "xtabs") with 2940985206 elements (frequency counts).

Details

This dataset is a further variant of the table1 to table5 datasets of the **tidyr** package.

Source

Generated by using stats::xtabs(formula = count ~.,data = tidyr::table2).

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, tb `tb` *Data table tb.*

Description

`tb` is a fictitious dataset describing 100 non-existing, but otherwise ordinary people.

Usage

```
tb
```

Format

A table with 100 cases (rows) and 5 variables (columns).

Details

Codebook

The table contains 5 columns/variables:

- 1. **id**: Participant ID.
- 2. **age**: Age (in years).
- 3. **height**: Height (in cm).
- 4. **shoesize**: Shoesize (EU standard).
- 5. **IQ**: IQ score (according Raven's Regressive Tables).

`tb` was originally created to practice loops and iterations (as a CSV file).

Source

See CSV data file at http://rpository.com/ds4psy/data/tb.csv.

See Also

Other datasets: `Bushisms`, `Trumpisms`, `countries`, `data_1`, `data_2`, `data_t1_de`, `data_t1_tab`, `data_t1`, `data_t2`, `data_t3`, `data_t4`, `dt_10`, `exp_num_dt`, `exp_wide`, `falsePosPsy_all`, `fame`, `flowery`, `fruits`, `outliers`, `pi_100k`, `posPsy_AHI_CESD`, `posPsy_long`, `posPsy_p_info`, `posPsy_wide`, `t3`, `t4`, `t_1`, `t_2`, `t_3`, `t_4`, `table6`, `table7`, `table8`, `table9`

text_to_chars — *Split string(s) of text x into its characters.*

Description

text_to_chars splits a string of text x (consisting of one or more character strings) into a vector of its individual characters.

Usage

```
text_to_chars(x, rm_specials = FALSE, sep = "")
```

Arguments

x	A string of text (required).
rm_specials	Boolean: Remove special characters? Default: rm_specials = TRUE.
sep	Character to insert between the elements of a multi-element character vector as input x? Default: sep = "" (i.e., add nothing).

Details

If rm_specials = TRUE, most special (or non-word) characters are removed. (Note that this currently works without using regular expressions.)

Value

A character vector (containing individual characters).

See Also

text_to_sentences for splitting text into a vector of sentences; text_to_words for splitting text into a vector of words; count_chars for counting the frequency of characters; count_words for counting the frequency of words; strsplit for splitting strings.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
s3 <- c("A 1st sentence.", "The 2nd sentence.",
        "A 3rd --- and  FINAL --- sentence.")
text_to_chars(s3)
text_to_chars(s3, sep = "\n")
text_to_chars(s3, rm_specials = TRUE)
```

---

text_to_sentences *Split strings of text* x *into sentences.*

Description

text_to_sentences splits text x (consisting of one or more character strings) into a vector of its constituting sentences.

Usage

```
text_to_sentences(
  x,
  sep = " ",
  split_delim = "\\.|\\?|!",
  force_delim = FALSE
)
```

Arguments

x	A string of text (required), typically a character vector.
sep	A character inserted as separator/delimiter between elements when collapsing multi-element strings of x. Default: sep = " " (i.e., insert 1 space between elements).
split_delim	Sentence delimiters (as regex) used to split the collapsed string of x into substrings. Default: split_delim = "\.|\?|!" (rather than "[[:punct:]]").
force_delim	Boolean: Enforce splitting at split_delim? If force_delim = FALSE (as per default), a standard sentence-splitting pattern is assumed: split_delim is followed by one or more blank spaces and a capital letter. If force_delim = TRUE, splits at split_delim are enforced (without considering spacing or capitalization).

Details

The splits of x will occur at given punctuation marks (provided as a regular expression, default: split_delim = "\.|\?|!"). Empty leading and trailing spaces are removed before returning a vector of the remaining character sequences (i.e., the sentences).

The Boolean argument force_delim distinguishes between two splitting modes:

1. If force_delim = FALSE (as per default), a standard sentence-splitting pattern is assumed: A sentence delimiter in split_delim must be followed by one or more blank spaces and a capital letter starting the next sentence. Sentence delimiters in split_delim are not removed from the output.
2. If force_delim = TRUE, the function enforces splits at each delimiter in split_delim. For instance, any dot (i.e., the metacharacter "\.") is interpreted as a full stop, so that sentences containing dots mid-sentence (e.g., for abbreviations, etc.) are split into parts. Sentence delimiters in split_delim are removed from the output.

Internally, text_to_sentences first uses paste to collapse strings (adding sep between elements) and then strsplit to split strings at split_delim.

Value

A character vector (of sentences).

See Also

text_to_words for splitting text into a vector of words; text_to_chars for splitting text into a vector of characters; count_words for counting the frequency of words; strsplit for splitting strings.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_words(), transl33t(), words_to_text()

Examples

```
x <- c("A first sentence. Exclamation sentence!",
       "Any questions? But etc. can be tricky. A fourth --- and final --- sentence.")
text_to_sentences(x)
text_to_sentences(x, force_delim = TRUE)

# Changing split delimiters:
text_to_sentences(x, split_delim = "\\.")  # only split at "."

text_to_sentences("Buy apples, berries, and coconuts.")
text_to_sentences("Buy apples, berries; and coconuts.",
                  split_delim = ",|;|\\.", force_delim = TRUE)

text_to_sentences(c("123. 456? 789! 007 etc."), force_delim = TRUE)

# Split multi-element strings (w/o punctuation):
e3 <- c("12", "34", "56")
text_to_sentences(e3, sep = " ")  # Default: Collapse strings adding 1 space, but:
text_to_sentences(e3, sep = ".", force_delim = TRUE)  # insert sep and force split.

# Punctuation within sentences:
text_to_sentences("Dr. who is left intact.")
text_to_sentences("Dr. Who is problematic.")
```

---

text_to_words *Split string(s) of text x into words.*

Description

text_to_words splits a string of text x (consisting of one or more character strings) into a vector of its constituting words.

Usage

```
text_to_words(x)
```

Arguments

x	A string of text (required), typically a character vector.

Details

`text_to_words` removes all (standard) punctuation marks and empty spaces in the resulting text parts, before returning a vector of the remaining character symbols (as its words).

Internally, `text_to_words` uses `strsplit` to split strings at punctuation marks (`split = "[[:punct:]]"`) and blank spaces (`split = "( ){1,}"`).

Value

A character vector (of words).

See Also

`text_to_words` for splitting a text into its words; `text_to_sentences` for splitting text into a vector of sentences; `text_to_chars` for splitting text into a vector of characters; `count_words` for counting the frequency of words; `strsplit` for splitting strings.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars_words()`, `count_chars()`, `count_words()`, `invert_rules()`, `l33t_rul35`, `map_text_chars()`, `map_text_coord()`, `map_text_regex()`, `metachar`, `read_ascii()`, `text_to_chars()`, `text_to_sentences()`, `transl33t()`, `words_to_text()`

Examples

```
# Default:
x <- c("Hello!", "This is a 1st sentence.", "This is the 2nd sentence.", "The end.")
text_to_words(x)
```

---

theme_clean — *A clean alternative theme for ggplot2.*

Description

`theme_clean` provides an alternative **ds4psy** theme to use in **ggplot2** commands.

Usage

```
theme_clean(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  col_title = grey(0, 1),
  col_panel = grey(0.85, 1),
  col_gridx = grey(1, 1),
  col_gridy = grey(1, 1),
  col_ticks = grey(0.1, 1)
)
```

Arguments

base_size	Base font size (optional, numeric). Default: base_size = 11.
base_family	Base font family (optional, character). Default: base_family = "". Options include "mono", "sans" (default), and "serif".
base_line_size	Base line size (optional, numeric). Default: base_line_size = base_size/22.
base_rect_size	Base rectangle size (optional, numeric). Default: base_rect_size = base_size/22.
col_title	Color of plot title (and tag). Default: col_title = grey(.0,1) (i.e., "black").
col_panel	Color of panel background(s). Default: col_panel = grey(.85,1) (i.e., light "grey").
col_gridx	Color of (major) panel lines (through x/vertical). Default: col_gridx = grey(1.0,1) (i.e., "white").
col_gridy	Color of (major) panel lines (through y/horizontal). Default: col_gridy = grey(1.0,1) (i.e., "white").
col_ticks	Color of axes text and ticks. Default: col_ticks = grey(.10,1) (i.e., near "black").

Details

theme_clean is more minimal than theme_ds4psy and fills panel backgrounds with a color col_panel.

This theme works well for plots with multiple panels, strong colors and bright color accents, but is of limited use with transparent colors.

Value

A **ggplot2** theme.

See Also

theme_ds4psy for default theme.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_fun(), plot_n(), plot_text(), plot_tiles(), theme_ds4psy(), theme_empty()

Examples

```
# Plotting iris dataset (using ggplot2, theme_grau, and unikn colors):

library('ggplot2')  # theme_clean() requires ggplot2
library('unikn')    # for colors and usecol() function

ggplot(datasets::iris) +
 geom_jitter(aes(x = Sepal.Length, y = Sepal.Width, color = Species), size = 3, alpha = 3/4) +
  facet_wrap(~Species) +
  scale_color_manual(values = usecol(pal = c(Pinky, Karpfenblau, Seegruen))) +
  labs(tag = "B",
       title = "Iris sepals",
       caption = "Data from datasets::iris") +
  coord_fixed(ratio = 3/2) +
  theme_clean()
```

---

theme_ds4psy   *A basic and flexible plot theme (using ggplot2 and unikn).*

Description

theme_ds4psy provides a generic **ds4psy** theme to use in **ggplot2** commands.

Usage

```
theme_ds4psy(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  col_title = grey(0, 1),
  col_txt_1 = grey(0.1, 1),
  col_txt_2 = grey(0.2, 1),
  col_txt_3 = grey(0.1, 1),
  col_bgrnd = "transparent",
  col_panel = grey(1, 1),
  col_strip = "transparent",
  col_axes = grey(0, 1),
  col_gridx = grey(0.75, 1),
  col_gridy = grey(0.75, 1),
  col_brdrs = "transparent"
)
```

Arguments

base_size	Base font size (optional, numeric). Default: base_size = 11.
base_family	Base font family (optional, character). Default: base_family = "". Options include "mono", "sans" (default), and "serif".
base_line_size	Base line size (optional, numeric). Default: base_line_size = base_size/22.
base_rect_size	Base rectangle size (optional, numeric). Default: base_rect_size = base_size/22.
col_title	Color of plot title (and tag). Default: col_title = grey(.0,1) (i.e., "black").
col_txt_1	Color of primary text (headings and axis labels). Default: col_title = grey(.1,1).
col_txt_2	Color of secondary text (caption, legend, axes labels/ticks). Default: col_title = grey(.2,1).
col_txt_3	Color of other text (facet strip labels). Default: col_title = grey(.1,1).
col_bgrnd	Color of plot background. Default: col_bgrnd = "transparent".
col_panel	Color of panel background(s). Default: col_panel = grey(1.0,1) (i.e., "white").
col_strip	Color of facet strips. Default: col_strip = "transparent".
col_axes	Color of (x and y) axes. Default: col_axes = grey(.00,1) (i.e., "black").
col_gridx	Color of (major and minor) panel lines (through x/vertical). Default: col_gridx = grey(.75,1) (i.e., light "grey").
col_gridy	Color of (major and minor) panel lines (through y/horizontal). Default: col_gridy = grey(.75,1) (i.e., light "grey").
col_brdrs	Color of (panel and strip) borders. Default: col_brdrs = "transparent".

Details

The theme is lightweight and no-nonsense, but somewhat opinionated (e.g., in using transparency and grid lines, and relying on grey tones for emphasizing data with color accents).

Basic sizes and the colors of text elements, backgrounds, and lines can be specified. However, excessive customization rarely yields aesthetic improvements over the standard **ggplot2** themes.

Value

A **ggplot2** theme.

See Also

unikn::theme_unikn inspired the current theme.

Other plot functions: plot_charmap(), plot_chars(), plot_fn(), plot_fun(), plot_n(), plot_text(), plot_tiles(), theme_clean(), theme_empty()

Examples

```
# Plotting iris dataset (using ggplot2 and unikn):

library('ggplot2')  # theme_ds4psy() requires ggplot2
library('unikn')    # for colors and usecol() function

ggplot(datasets::iris) +
  geom_jitter(aes(x = Petal.Length, y = Petal.Width, color = Species), size = 3, alpha = 2/3) +
  scale_color_manual(values = usecol(pal = c(Pinky, Seeblau, Seegruen))) +
  labs(title = "Iris petals",
       subtitle = "The subtitle of this plot",
       caption = "Data from datasets::iris") +
  theme_ds4psy()

ggplot(datasets::iris) +
  geom_jitter(aes(x = Sepal.Length, y = Sepal.Width, color = Species), size = 3, alpha = 2/3) +
  facet_wrap(~Species) +
  scale_color_manual(values = usecol(pal = c(Pinky, Seeblau, Seegruen))) +
  labs(tag = "A",
       title = "Iris sepals",
       subtitle = "Demo plot with facets and default colors",
       caption = "Data from datasets::iris") +
  coord_fixed(ratio = 3/2) +
  theme_ds4psy()

# A unikn::Seeblau look:

ggplot(datasets::iris) +
  geom_jitter(aes(x = Sepal.Length, y = Sepal.Width, color = Species), size = 3, alpha = 2/3) +
  facet_wrap(~Species) +
  scale_color_manual(values = usecol(pal = c(Pinky, Seeblau, Seegruen))) +
  labs(tag = "B",
       title = "Iris sepals",
       subtitle = "Demo plot in unikn::Seeblau colors",
       caption = "Data from datasets::iris") +
  coord_fixed(ratio = 3/2) +
  theme_ds4psy(col_title = pal_seeblau[[4]], col_strip = pal_seeblau[[1]], col_brdrs = Grau)
```

`theme_empty` *A basic and flexible plot theme (using ggplot2 and unikn).*

Description

`theme_empty` provides an empty (blank) theme to use in **ggplot2** commands.

Usage

```
theme_empty(
  font_size = 12,
  font_family = "",
  rel_small = 12/14,
  plot_mar = c(0, 0, 0, 0)
)
```

Arguments

`font_size`	Overall font size. Default: `font_size = 12`.
`font_family`	Base font family. Default: `font_family = ""`.
`rel_small`	Relative size of smaller text. Default: `rel_small = 10/12`.
`plot_mar`	Plot margin sizes (on top, right, bottom, left). Default: `plot_mar = c(0,0,0,0)` (in lines).

Details

`theme_empty` shows nothing but the plot panel.

`theme_empty` is based on `theme_nothing` of the **cowplot** package and uses `theme_void` of the **ggplot2** package.

Value

A **ggplot2** theme.

See Also

`cowplot::theme_nothing` is the inspiration and source of this theme.

Other plot functions: `plot_charmap()`, `plot_chars()`, `plot_fn()`, `plot_fun()`, `plot_n()`, `plot_text()`, `plot_tiles()`, `theme_clean()`, `theme_ds4psy()`

Examples

```
# Plotting iris dataset (using ggplot2):

library('ggplot2')  # theme_empty() requires ggplot2

ggplot(datasets::iris) +
 geom_point(aes(x = Petal.Length, y = Petal.Width, color = Species), size = 4, alpha = 1/2) +
  scale_color_manual(values = c("firebrick3", "deepskyblue3", "olivedrab3")) +
  labs(title = "NOT SHOWN: Title",
       subtitle = "NOT SHOWN: Subtitle",
       caption = "NOT SHOWN: Data from datasets::iris") +
  theme_empty(plot_mar = c(2, 0, 1, 0))  # margin lines (top, right, bot, left)
```

transl33t *transl33t translates text into leet slang.*

Description

transl33t translates text into leet (or l33t) slang given a set of rules.

Usage

```
transl33t(txt, rules = l33t_rul35, in_case = "no", out_case = "no")
```

Arguments

txt	The text (character string) to translate.
rules	Rules which existing character in txt is to be replaced by which new character (as a named character vector). Default: rules = l33t_rul35.
in_case	Change case of input string txt. Default: in_case = "no". Set to "lo" or "up" for lower or uppercase, respectively.
out_case	Change case of output string. Default: out_case = "no". Set to "lo" or "up" for lower or uppercase, respectively.

Details

The current version of transl33t only uses base R commands, rather than the **stringr** package.

Value

A character vector.

See Also

l33t_rul35 for default rules used; invert_rules for inverting rules.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), words_to_text()

Examples

```
# Use defaults:
transl33t(txt = "hello world")
transl33t(txt = c(letters))
transl33t(txt = c(LETTERS))

# Specify rules:
transl33t(txt = "hello world",
          rules = c("e" = "3", "l" = "1", "o" = "0"))

# Set input and output case:
transl33t(txt = "hello world", in_case = "up",
          rules = c("e" = "3", "l" = "1", "o" = "0"))  # e only capitalized
transl33t(txt = "hEllo world", in_case = "lo", out_case = "up",
          rules = c("e" = "3", "l" = "1", "o" = "0"))  # e transl33ted
```

Trumpisms *Data: Trumpisms.*

Description

Trumpisms contains frequent words and characteristic phrases by U.S. president Donald J. Trump (the 45th president of the United States, in office from January 20, 2017, to January 20, 2021).

Usage

```
Trumpisms
```

Format

A vector of type character with length(Trumpisms) = 168 (on 2021-01-28).

Source

Data originally based on a collection of *Donald Trump's 20 most frequently used words* on https://www.yourdictionary.com and expanded by interviews, public speeches, and Twitter tweets at https://twitter.com/realDonaldTrump.

See Also

Other datasets: Bushisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, t_4, table6, table7, table8, table9, tb

t_1 *Data t_1.*

Description

t_1 is a fictitious dataset to practice tidying data.

Usage

```
t_1
```

Format

A table with 8 cases (rows) and 9 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t_1.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_2, t_3, t_4, table6, table7, table8, table9, tb

t_2 *Data t_2.*

Description

t_2 is a fictitious dataset to practice tidying data.

Usage

```
t_2
```

Format

A table with 8 cases (rows) and 5 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t_2.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_3, t_4, table6, table7, table8, table9, tb

---

t_3 *Data t_3.*

Description

t_3 is a fictitious dataset to practice tidying data.

Usage

```
t_3
```

Format

A table with 16 cases (rows) and 6 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t_3.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_4, table6, table7, table8, table9, tb

---

t_4 *Data t_4.*

Description

t_4 is a fictitious dataset to practice tidying data.

Usage

```
t_4
```

Format

A table with 16 cases (rows) and 8 variables (columns).

Source

See CSV data at http://rpository.com/ds4psy/data/t_4.csv.

See Also

Other datasets: Bushisms, Trumpisms, countries, data_1, data_2, data_t1_de, data_t1_tab, data_t1, data_t2, data_t3, data_t4, dt_10, exp_num_dt, exp_wide, falsePosPsy_all, fame, flowery, fruits, outliers, pi_100k, posPsy_AHI_CESD, posPsy_long, posPsy_p_info, posPsy_wide, t3, t4, t_1, t_2, t_3, table6, table7, table8, table9, tb

---

Umlaut *Umlaut provides German Umlaut letters (as Unicode characters).*

Description

Umlaut provides the German Umlaut letters (aka. diaeresis/diacritic) as a named character vector.

Usage

```
Umlaut
```

Format

An object of class character of length 7.

Details

For Unicode details, see https://home.unicode.org/,

For details on German Umlaut letters (aka. diaeresis/diacritic), see https://en.wikipedia.org/wiki/Diaeresis_(diacritic) and https://en.wikipedia.org/wiki/Germanic_umlaut.

See Also

Other text objects and functions: capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t(), words_to_text()

Examples

```
Umlaut
names(Umlaut)

paste0("Hansj", Umlaut["o"], "rg i", Umlaut["s"], "t s", Umlaut["u"], "sse ", Umlaut["A"], "pfel.")
paste0("Das d", Umlaut["u"], "nne M", Umlaut["a"], "dchen l", Umlaut["a"], "chelt.")
paste0("Der b", Umlaut["o"], "se Mann macht ", Umlaut["u"], "blen ", Umlaut["A"], "rger.")
paste0("Das ", Umlaut["U"], "ber-Ich ist ", Umlaut["a"], "rgerlich.")
```

what_date — *What date is it?*

Description

what_date provides a satisficing version of Sys.Date() that is sufficient for most purposes.

Usage

```
what_date(
  when = NA,
  rev = FALSE,
  as_string = TRUE,
  sep = "-",
  month_form = "m",
  tz = ""
)
```

Arguments

when	Date(s) (as a scalar or vector). Default: when = NA. Using as.Date(when) to convert strings into dates, and Sys.Date(), if when = NA.
rev	Boolean: Reverse date (to Default: rev = FALSE.
as_string	Boolean: Return as character string? Default: as_string = TRUE. If as_string = FALSE, a "Date" object is returned.
sep	Character: Separator to use. Default: sep = "-".
month_form	Character: Month format. Default: month_form = "m" for numeric month (01-12). Use month_form = "b" for short month name and month_form = "B" for full month name (in current locale).
tz	Time zone. Default: tz = "" (i.e., current system time zone, see Sys.timezone()). Use tz = "UTC" for Coordinated Universal Time.

Details

By default, what_date returns either Sys.Date() or the dates provided by when as a character string (using current system settings and sep for formatting). If as_string = FALSE, a "Date" object is returned.

The tz argument allows specifying time zones (see Sys.timezone() for current setting and OlsonNames() for options.)

However, tz is merely used to represent the dates provided to the when argument. Thus, there currently is no active conversion of dates into other time zones (see the today function of **lubridate** package).

Value

A character string or object of class "Date".

See Also

what_wday() function to obtain (week)days; what_time() function to obtain times; cur_time() function to print the current time; cur_date() function to print the current date; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_month(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
what_date()
what_date(sep = "/")
what_date(rev = TRUE)
what_date(rev = TRUE, sep = ".")
what_date(rev = TRUE, sep = " ", month_form = "B")

# with "POSIXct" times:
what_date(when = Sys.time())

# with time vector (of "POSIXct" objects):
ts <- c("1969-07-13 13:53 CET", "2020-12-31 23:59:59")
what_date(ts)
what_date(ts, rev = TRUE, sep = ".")
what_date(ts, rev = TRUE, month_form = "b")

# return a "Date" object:
dt <- what_date(as_string = FALSE)
class(dt)

# with time zone:
ts <- ISOdate(2020, 12, 24, c(0, 12))  # midnight and midday UTC
what_date(when = ts, tz = "US/Hawaii", as_string = FALSE)
```

---

what_month *What month is it?*

Description

what_month provides a satisficing version of to determine the month corresponding to a given date.

Usage

```
what_month(when = Sys.Date(), abbr = FALSE, as_integer = FALSE)
```

Arguments

when	Date (as a scalar or vector). Default: when = NA. Using as.Date(when) to convert strings into dates, and Sys.Date(), if when = NA.
abbr	Boolean: Return abbreviated? Default: abbr = FALSE.
as_integer	Boolean: Return as integer? Default: as_integer = FALSE.

Details

what_month returns the month of when or Sys.Date() (as a name or number).

See Also

what_week() function to obtain weeks; what_date() function to obtain dates; cur_time() function to print the current time; cur_date() function to print the current date; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_time(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
what_month()
what_month(abbr = TRUE)
what_month(as_integer = TRUE)

# with date vector (as characters):
ds <- c("2020-01-01", "2020-02-29", "2020-12-24", "2020-12-31")
what_month(when = ds)
what_month(when = ds, abbr = TRUE, as_integer = FALSE)
what_month(when = ds, abbr = TRUE, as_integer = TRUE)

# with time vector (strings of POSIXct times):
ts <- c("2020-02-29 10:11:12 CET", "2020-12-31 23:59:59")
what_month(ts)
```

---

what_time *What time is it?*

Description

what_time provides a satisficing version of Sys.time() that is sufficient for most purposes.

Usage

```
what_time(when = NA, seconds = FALSE, as_string = TRUE, sep = ":", tz = "")
```

Arguments

when	Time (as a scalar or vector). Default: when = NA. Returning Sys.time(), if when = NA.
seconds	Boolean: Show time with seconds? Default: seconds = FALSE.
as_string	Boolean: Return as character string? Default: as_string = TRUE. If as_string = FALSE, a "POSIXct" object is returned.
sep	Character: Separator to use. Default: sep = ":".
tz	Time zone. Default: tz = "" (i.e., current system time zone, see Sys.timezone()). Use tz = "UTC" for Coordinated Universal Time.

Details

By default, what_time prints a simple version of when or Sys.time() as a character string (in " using current default system settings. If as_string = FALSE, a "POSIXct" (calendar time) object is returned.

The tz argument allows specifying time zones (see Sys.timezone() for current setting and OlsonNames() for options.)

However, tz is merely used to represent the times provided to the when argument. Thus, there currently is no active conversion of times into other time zones (see the now function of **lubridate** package).

Value

A character string or object of class "POSIXct".

See Also

cur_time() function to print the current time; cur_date() function to print the current date; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_wday(), what_week(), what_year(), zodiac()

Examples

```
what_time()

# with vector (of "POSIXct" objects):
tm <- c("2020-02-29 01:02:03", "2020-12-31 14:15:16")
what_time(tm)

# with time zone:
ts <- ISOdate(2020, 12, 24, c(0, 12))  # midnight and midday UTC
t1 <- what_time(when = ts, tz = "US/Hawaii")
t1  # time display changed, due to tz

# return "POSIXct" object(s):
# Same time in differen tz:
```

```
t2 <- what_time(as.POSIXct("2020-02-29 10:00:00"), as_string = FALSE, tz = "US/Hawaii")
format(t2, "%F %T %Z (UTF %z)")
# from string:
t3 <- what_time("2020-02-29 10:00:00", as_string = FALSE, tz = "US/Hawaii")
format(t3, "%F %T %Z (UTF %z)")
```

---

what_wday *What day of the week is it?*

Description

what_wday provides a satisficing version of to determine the day of the week corresponding to a given date.

Usage

```
what_wday(when = Sys.Date(), abbr = FALSE)
```

Arguments

when	Date (as a scalar or vector). Default: when = Sys.Date(). Aiming to convert when into "Date" if a different object class is provided.
abbr	Boolean: Return abbreviated? Default: abbr = FALSE.

Details

what_wday returns the name of the weekday of when or of Sys.Date() (as a character string).

See Also

what_date() function to obtain dates; what_time() function to obtain times; cur_time() function to print the current time; cur_date() function to print the current date; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_week(), what_year(), zodiac()

Examples

```
what_wday()
what_wday(abbr = TRUE)

what_wday(Sys.Date() + -1:1)  # Date (as vector)
what_wday(Sys.time())         # POSIXct
what_wday("2020-02-29")       # string (of valid date)
what_wday(20200229)           # number (of valid date)
```

```
# date vector (as characters):
ds <- c("2020-01-01", "2020-02-29", "2020-12-24", "2020-12-31")
what_wday(when = ds)
what_wday(when = ds, abbr = TRUE)

# time vector (strings of POSIXct times):
ts <- c("1969-07-13 13:53 CET", "2020-12-31 23:59:59")
what_wday(ts)

# fame data:
greta_dob <- as.Date(fame[grep(fame$name, pattern = "Greta") , ]$DOB, "%B %d, %Y")
what_wday(greta_dob)  # Friday, of course.
```

`what_week` *What week is it?*

Description

`what_week` provides a satisficing version of to determine the week corresponding to a given date.

Usage

```
what_week(when = Sys.Date(), unit = "year", as_integer = FALSE)
```

Arguments

`when`	Date (as a scalar or vector). Default: `when = Sys.Date()`. Using `as.Date(when)` to convert strings into dates if a different `when` is provided.
`unit`	Character: Unit of week? Possible values are `"month"`, `"year"`. Default: `unit = "year"` (for week within year).
`as_integer`	Boolean: Return as integer? Default: `as_integer = FALSE`.

Details

`what_week` returns the week of `when` or `Sys.Date()` (as a name or number).

See Also

`what_wday()` function to obtain (week)days; `what_date()` function to obtain dates; `cur_time()` function to print the current time; `cur_date()` function to print the current date; `now()` function of the **lubridate** package; `Sys.time()` function of **base** R.

Other date and time functions: `change_time()`, `change_tz()`, `cur_date()`, `cur_time()`, `days_in_month()`, `diff_dates()`, `diff_times()`, `diff_tz()`, `is_leap_year()`, `what_date()`, `what_month()`, `what_time()`, `what_wday()`, `what_year()`, `zodiac()`

Examples

```
what_week()
what_week(as_integer = TRUE)

# Other dates/times:
d1 <- as.Date("2020-12-24")
what_week(when = d1, unit = "year")
what_week(when = d1, unit = "month")

what_week(Sys.time())  # with POSIXct time

# with date vector (as characters):
ds <- c("2020-01-01", "2020-02-29", "2020-12-24", "2020-12-31")
what_week(when = ds)
what_week(when = ds, unit = "month", as_integer = TRUE)
what_week(when = ds, unit = "year", as_integer = TRUE)

# with time vector (strings of POSIXct times):
ts <- c("2020-12-25 10:11:12 CET", "2020-12-31 23:59:59")
what_week(ts)
```

---

what_year *What year is it?*

Description

what_year provides a satisficing version of to determine the year corresponding to a given date.

Usage

```
what_year(when = Sys.Date(), abbr = FALSE, as_integer = FALSE)
```

Arguments

when	Date (as a scalar or vector). Default: when = NA. Using as.Date(when) to convert strings into dates, and Sys.Date(), if when = NA.
abbr	Boolean: Return abbreviated? Default: abbr = FALSE.
as_integer	Boolean: Return as integer? Default: as_integer = FALSE.

Details

what_year returns the year of when or Sys.Date() (as a name or number).

See Also

what_week() function to obtain weeks; what_month() function to obtain months; cur_time() function to print the current time; cur_date() function to print the current date; now() function of the **lubridate** package; Sys.time() function of **base** R.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), zodiac()

Examples

```
what_year()
what_year(abbr = TRUE)
what_year(as_integer = TRUE)

# with date vectors (as characters):
ds <- c("2020-01-01", "2020-02-29", "2020-12-24", "2020-12-31")
what_year(when = ds)
what_year(when = ds, abbr = TRUE, as_integer = FALSE)
what_year(when = ds, abbr = TRUE, as_integer = TRUE)

# with time vector (strings of POSIXct times):
ts <- c("2020-02-29 10:11:12 CET", "2020-12-31 23:59:59")
what_year(ts)
```

---

words_to_text *Paste or collapse words* x *into a text.*

Description

words_to_text pastes or collapses a character string x into a text.

Usage

```
words_to_text(x, collapse = " ")
```

Arguments

x	A string of text (required), typically a character vector.
collapse	A character string to separate the elements of x in the resulting text. Default: collapse = " ".

Details

Internally, words_to_text only invokes the **base** R function paste with the collapse argument.

Value

A text (as a collapsed character vector).

See Also

text_to_words for splitting a text into its words; text_to_sentences for splitting text into a vector of sentences; text_to_chars for splitting text into a vector of characters; count_words for counting the frequency of words; strsplit for splitting strings.

Other text objects and functions: Umlaut, capitalize(), caseflip(), cclass, count_chars_words(), count_chars(), count_words(), invert_rules(), l33t_rul35, map_text_chars(), map_text_coord(), map_text_regex(), metachar, read_ascii(), text_to_chars(), text_to_sentences(), text_to_words(), transl33t()

Examples

```
x <- c("Hello world!", "A 1st sentence.", "A 2nd sentence.", "The end.")
words_to_text(x)
cat(words_to_text(x, collapse = "\n"))
```

---

zodiac — *Get zodiac (corresponding to date x).*

Description

zodiac provides the tropical zodiac sign or symbol for given date(s) x.

Usage

```
zodiac(
  x,
  out = "en",
  zodiac_swap_mmdd = c(120, 219, 321, 421, 521, 621, 723, 823, 923, 1023, 1123, 1222)
)
```

Arguments

x	Date (as a scalar or vector, required). If x is not a date (of class "Date"), the function tries to coerce x into a "Date".
out	Output format (as character). Available output formats are: English/Latin (out = "en", by default), German/Deutsch (out = "de"), HTML (out = "html"), or Unicode (out = "Unicode") symbols.
zodiac_swap_mmdd	Monthly dates on which the 12 zodiac signs switch (in mmdd format, ordered chronologically within a calendar year). Default: zodiac_swap_mmdd = c(0120,0219,0321,0421,0521,

Details

zodiac is flexible by providing different output formats (in Latin/English, German, or Unicode/HTML, see out) and allowing to adjust the calendar dates on which a new zodiac is assigned (via zodiac_swap_mmdd).

Value

Zodiac label or symbol (as a factor).

Source

See https://en.wikipedia.org/wiki/Zodiac or https://de.wikipedia.org/wiki/Tierkreiszeichen for alternative date ranges.

See Also

Zodiac() function of the **DescTools** package.

Other date and time functions: change_time(), change_tz(), cur_date(), cur_time(), days_in_month(), diff_dates(), diff_times(), diff_tz(), is_leap_year(), what_date(), what_month(), what_time(), what_wday(), what_week(), what_year()

Examples

```
zodiac(Sys.Date())

# Works with vectors:
dt <- sample_date(size = 10)
zodiac(dt)
levels(zodiac(dt))

# Alternative outputs:
zodiac(dt, out = "de")  # German/deutsch
zodiac(dt, out = "Unicode")  # Unicode
zodiac(dt, out = "HTML")     # HTML

# Alternative date breaks:
zodiac("2000-08-23")  # 0823 is "Virgo" by default
zodiac("2000-08-23",  # change to 0824 (i.e., August 24):
       zodiac_swap_mmdd = c(0120, 0219, 0321, 0421, 0521, 0621,
                            0723, 0824, 0923, 1023, 1123, 1222))
```

Index

∗ **color objects and functions**
pal_ds4psy, 55
pal_n_sq, 56
∗ **data functions**
get_set, 37
make_grid, 44
∗ **datasets**
Bushisms, 4
cclass, 7
countries, 11
data_1, 17
data_2, 18
data_t1, 19
data_t1_de, 19
data_t1_tab, 20
data_t2, 20
data_t3, 21
data_t4, 22
dt_10, 31
exp_num_dt, 32
exp_wide, 33
falsePosPsy_all, 33
fame, 35
flowery, 36
fruits, 36
l33t_rul35, 43
metachar, 50
outliers, 54
pal_ds4psy, 55
pi_100k, 56
posPsy_AHI_CESD, 72
posPsy_long, 73
posPsy_p_info, 74
posPsy_wide, 75
t3, 81
t4, 81
t_1, 97
t_2, 97
t_3, 98
t_4, 98
table6, 82
table7, 83
table8, 83
table9, 84
tb, 85
Trumpisms, 96
Umlaut, 99
∗ **date and time functions**
change_time, 7
change_tz, 9
cur_date, 15
cur_time, 16
days_in_month, 22
diff_dates, 25
diff_times, 28
diff_tz, 29
is_leap_year, 40
what_date, 100
what_month, 101
what_time, 102
what_wday, 104
what_week, 105
what_year, 106
zodiac, 108
∗ **plot functions**
plot_charmap, 57
plot_chars, 59
plot_fn, 62
plot_fun, 64
plot_n, 65
plot_text, 67
plot_tiles, 70
theme_clean, 89
theme_ds4psy, 91
theme_empty, 93
∗ **sampling functions**
coin, 10
dice, 23

dice_2, 24
sample_char, 77
sample_date, 78
sample_time, 79
∗ **text objects and functions**
capitalize, 5
caseflip, 6
cclass, 7
count_chars, 12
count_chars_words, 13
count_words, 14
invert_rules, 38
l33t_rul35, 43
map_text_chars, 45
map_text_coord, 46
map_text_regex, 47
metachar, 50
read_ascii, 76
text_to_chars, 86
text_to_sentences, 87
text_to_words, 88
transl33t, 95
Umlaut, 99
words_to_text, 107
∗ **utility functions**
is_equal, 39
is_vect, 41
is_wholenumber, 42
num_as_char, 51
num_as_ordinal, 52
num_equal, 53

all.equal, 39, 54

Bushisms, 4, 11, 18–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99

capitalize, 5, 6, 7, 12, 14, 15, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
caseflip, 5, 6, 7, 12, 14, 15, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
cclass, 5, 6, 7, 12, 14, 15, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
change_time, 7, 9, 16, 17, 23, 27, 29, 30, 40, 101–105, 107, 109
change_tz, 8, 9, 16, 17, 23, 27, 29, 30, 40, 101–105, 107, 109
coin, 10, 24, 25, 78–80
count_chars, 5–7, 12, 13–15, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
count_chars_words, 5–7, 12, 13, 15, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
count_words, 5–7, 12–14, 14, 38, 44–46, 49, 50, 77, 86, 88, 89, 95, 99, 108
countries, 4, 11, 18–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
cur_date, 8, 9, 15, 17, 23, 27, 29, 30, 40, 101–105, 107, 109
cur_time, 8, 9, 16, 16, 23, 27, 29, 30, 40, 101–105, 107, 109

data_1, 4, 11, 17, 18–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_2, 4, 11, 18, 18, 19–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t1, 4, 11, 18, 19, 20–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t1_de, 4, 11, 18, 19, 19, 20–22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t1_tab, 4, 11, 18–20, 20, 21, 22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t2, 4, 11, 18–20, 20, 21, 22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t3, 4, 11, 18–21, 21, 22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
data_t4, 4, 11, 18–21, 22, 31–33, 35–37, 55, 57, 73–76, 81–85, 96–99
days_in_month, 8, 9, 16, 17, 22, 27, 29, 30, 40, 101–105, 107, 109
dice, 11, 23, 24, 25, 78–80
dice_2, 11, 24, 24, 78–80
diff_dates, 8, 9, 16, 17, 23, 25, 29, 30, 40, 101–105, 107, 109
diff_times, 8, 9, 16, 17, 23, 27, 28, 30, 40, 101–105, 107, 109
diff_tz, 8, 9, 16, 17, 23, 27, 29, 29, 40, 101–105, 107, 109
ds4psy.guide, 31
dt_10, 4, 11, 18–22, 31, 32, 33, 35–37, 55, 57, 73–76, 81–85, 96–99

exp_num_dt, 4, 11, 18–22, 31, 32, 33, 35–37, 55, 57, 73–76, 81–85, 96–99
exp_wide, 4, 11, 18–22, 31, 32, 33, 35–37, 55, 57, 73–76, 81–85, 96–99

falsePosPsy_all, *4*, *11*, *18–22*, *31–33*, 33, *35–37*, *55*, *57*, *73–76*, *81–85*, *96–99*
fame, *4*, *11*, *18–22*, *31–33*, *35*, 35, *36, 37*, *55*, *57*, *73–76*, *81–85*, *96–99*
flowery, *4*, *11*, *18–22*, *31–33*, *35*, 36, *37*, *55*, *57*, *73–76*, *81–85*, *96–99*
fruits, *4*, *11*, *18–22*, *31–33*, *35, 36*, 36, *55*, *57*, *73–76*, *81–85*, *96–99*

get_set, 37, *44*

invert_rules, *5–7*, *12*, *14, 15*, 38, *44–46*, *49, 50*, *77*, *86*, *88, 89*, *95*, *99*, *108*
is.atomic, *42*
is.integer, *43*
is.list, *42*
is.vector, *42*
is_equal, 39, *42, 43*, *51*, *53, 54*
is_leap_year, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, 40, *101–105*, *107*, *109*
is_vect, *39*, 41, *43*, *51*, *53, 54*
is_wholenumber, *39*, *42*, 42, *51*, *53, 54*

l33t_rul35, *5–7*, *12*, *14, 15*, *38*, 43, *45, 46*, *49, 50*, *77*, *86*, *88, 89*, *95*, *99*, *108*

make_grid, *37*, 44
map_text_chars, *5–7*, *12*, *14, 15*, *38*, *44*, 45, *46*, *49, 50*, *77*, *86*, *88, 89*, *95*, *99*, *108*
map_text_coord, *5–7*, *12*, *14, 15*, *38*, *44, 45*, 46, *49, 50*, *57, 58*, *61*, *69*, *76, 77*, *86*, *88, 89*, *95*, *99*, *108*
map_text_regex, *5–7*, *12*, *14, 15*, *38*, *44–46*, 47, *50*, *57, 58*, *61*, *69*, *77*, *86*, *88, 89*, *95*, *99*, *108*
metachar, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49*, 50, *77*, *86*, *88, 89*, *95*, *99*, *108*

num_as_char, *39*, *42, 43*, 51, *53, 54*
num_as_ordinal, *39*, *42, 43*, *51*, 52, *54*
num_equal, *39*, *42, 43*, *51*, *53*, 53

outliers, *4*, *11*, *18–22*, *31–33*, *35–37*, 54, *57*, *73–76*, *81–85*, *96–99*

pal_ds4psy, 55, *56*, *58*, *61*, *63*, *65, 66*, *69–71*
pal_n_sq, *55*, 56
paste, *88*, *107*
pi_100k, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, 56, *73–76*, *81–85*, *96–99*
plot_charmap, *46*, *49*, 57, *61*, *63*, *65, 66*, *69*, *71*, *90*, *92*, *94*
plot_chars, *12*, *14, 15*, *45, 46*, *49*, *58*, 59, *63*, *65, 66*, *68, 69*, *71*, *77*, *90*, *92*, *94*
plot_fn, *58*, *61*, 62, *65, 66*, *69*, *71*, *90*, *92*, *94*
plot_fun, *58*, *61*, *63*, 64, *66*, *69*, *71*, *90*, *92*, *94*
plot_n, *58*, *61*, *63*, *65*, 65, *69*, *71*, *90*, *92*, *94*
plot_text, *49*, *58*, *61*, *63*, *65, 66*, 67, *71*, *90*, *92*, *94*
plot_tiles, *56*, *58*, *61*, *63*, *65, 66*, *69*, 70, *90*, *92*, *94*
posPsy_AHI_CESD, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, 72, *73–76*, *81–85*, *96–99*
posPsy_long, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73*, 73, *75, 76*, *81–85*, *96–99*
posPsy_p_info, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73, 74*, 74, *76*, *81–85*, *96–99*
posPsy_wide, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–75*, 75, *81–85*, *96–99*

read_ascii, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *58*, *61*, *69*, 76, *86*, *88, 89*, *95*, *99*, *108*

sample_char, *11*, *24, 25*, 77, *79, 80*
sample_date, *11*, *24, 25*, *78*, 78, *80*
sample_time, *11*, *24, 25*, *78, 79*, 79
strsplit, *86*, *88, 89*, *108*

t3, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, 81, *82–85*, *96–99*
t4, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81*, 81, *82–85*, *96–99*
t_1, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–85*, *96*, 97, *98, 99*
t_2, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–85*, *96, 97*, 97, *98, 99*
t_3, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–85*, *96–98*, 98, *99*
t_4, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–85*, *96–98*, 98
table6, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81, 82*, 82, *83–85*, *96–99*
table7, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81, 82*, 83, *84, 85*, *96–99*
table8, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–83*, 83, *84, 85*, *96–99*

table9, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–84*, 84, *85*, *96–99*
tb, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–84*, 85, *96–99*
text_to_chars, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, 86, *88, 89*, *95*, *99*, *108*
text_to_sentences, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, *86*, 87, *89*, *95*, *99*, *108*
text_to_words, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, *86*, *88*, 88, *89*, *95*, *99*, *108*
theme_clean, *58*, *61*, *63*, *65, 66*, *69*, *71*, 89, *92*, *94*
theme_ds4psy, *58*, *61*, *63*, *65, 66*, *69*, *71*, *90*, 91, *94*
theme_empty, *58*, *61*, *63*, *65, 66*, *69*, *71*, *90*, *92*, 93
transl33t, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, *86*, *88, 89*, 95, *99*, *108*
Trumpisms, *4*, *11*, *18–22*, *31–33*, *35–37*, *55*, *57*, *73–76*, *81–85*, 96, *97–99*

Umlaut, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, *86*, *88, 89*, *95*, 99, *108*

what_date, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, 100, *102–105*, *107*, *109*
what_month, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101*, 101, *103–105*, *107*, *109*
what_time, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101, 102*, 102, *104, 105*, *107*, *109*
what_wday, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101–103*, 104, *105*, *107*, *109*
what_week, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101–104*, 105, *107*, *109*
what_year, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101–105*, 106, *109*
words_to_text, *5–7*, *12*, *14, 15*, *38*, *44–46*, *49, 50*, *77*, *86*, *88, 89*, *95*, *99*, 107

zodiac, *8, 9*, *16, 17*, *23*, *27*, *29, 30*, *40*, *101–105*, *107*, 108