

# Package ‘ds4psy’

July 6, 2020

**Type** Package

**Title** Data Science for Psychologists

**Version** 0.4.0

**Date** 2020-07-06

**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, 2020), 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 instructive and entertaining, rather than elegant or efficient.

**Depends** R (>= 3.5.0)

**Imports** ggplot2, cowplot, unkn

**Suggests** knitr, rmarkdown, spelling

**Collate** '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** 2020-07-06 14:50:03 UTC

**R topics documented:**

Bushisms	3
capitalize	4
caseflip	5
cclass	5
change_time	6
change_tz	7
coin	9
countries	10
count_chars	10
count_words	11
cur_date	12
cur_time	13
data_1	14
data_2	15
data_t1	15
data_t1_de	16
data_t1_tab	16
data_t2	17
data_t3	18
data_t4	18
dice	19
dice_2	20
ds4psy.guide	21
dt_10	21
exp_num_dt	22
exp_wide	23
falsePosPsy_all	23
fame	25
flowery	26
fruits	26
is.wholenumber	27
is_leap_year	28
l33t_rul35	29
make_grid	30
metachar	31
num_as_char	31
num_as_ordinal	33
outliers	34
pal_ds4psy	35
pal_n_sq	35
pi_100k	36
plot_fn	37
plot_fun	38
plot_n	39
plot_text	41
plot_tiles	43

posPsy_AHI_CESD . . . . .	45
posPsy_long . . . . .	47
posPsy_p_info . . . . .	48
posPsy_wide . . . . .	49
read_ascii . . . . .	50
sample_char . . . . .	51
sample_date . . . . .	52
sample_time . . . . .	53
t3 . . . . .	54
t4 . . . . .	55
table6 . . . . .	55
table7 . . . . .	56
table8 . . . . .	57
tb . . . . .	57
text_to_sentences . . . . .	58
text_to_words . . . . .	59
theme_ds4psy . . . . .	60
transl33t . . . . .	61
Trumpisms . . . . .	62
t_1 . . . . .	63
t_2 . . . . .	64
t_3 . . . . .	64
t_4 . . . . .	65
Umlaut . . . . .	65
what_date . . . . .	66
what_day . . . . .	68
what_month . . . . .	69
what_time . . . . .	70
what_week . . . . .	71
what_year . . . . .	72
<b>Index</b>	<b>74</b>

---

 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](#), [tb](#)

---

capitalize	<i>capitalize converts the case of each word's n initial characters (typically to upper) in a string of text x.</i>
------------	---

---

**Description**

capitalize converts the case of each word's n initial characters (typically to upper) in a string of text x.

**Usage**

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

**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	Return word vector as text (i.e., one character string)? Default: as_text = TRUE.

**See Also**

[caseflip](#) for converting the case of all letters.

Other text objects and functions: [Umlaut](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_rul35](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**Examples**

```
x <- c("Hello world! This is a 1st TEST sentence. The end.")
capitalize(x)
capitalize(x, n = 3)
capitalize(x, n = 2, upper = FALSE)
capitalize(x, as_text = FALSE)

# Note: A vector of character strings returns the same results:
x <- c("Hello world!", "This is a 1st TEST sentence.", "The end.")
capitalize(x)
```

```
capitalize(x, n = 3)
capitalize(x, n = 2, upper = FALSE)
capitalize(x, as_text = FALSE)
```

---

caseflip	<i>caseflip flips the case of characters in a string of text x.</i>
----------	---

---

### Description

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

### Usage

```
caseflip(x)
```

### Arguments

x                    A string of text (required).

### See Also

[capitalize](#) for converting the case of initial letters.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_ru135](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

### Examples

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

---

cclass	<i>cclass provides character classes (as a named vector).</i>
--------	---

---

### 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.

**See Also**

`metachar` for a vector of metacharacters.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `count_chars()`, `count_words()`, `l33t_ru135`, `metachar`, `read_ascii()`, `text_to_sentences()`, `text_to_words()`, `transl33t()`

**Examples**

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

---

change_time	<i>Change time and time zone (without changing time display).</i>
-------------	---

---

**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, <code>Sys.timezone()</code> ). See <code>OlsonNames()</code> 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

**Examples**

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

# from "POSIXlt" time:
t1 <- as.POSIXlt("2020-07-01 10:00:00", 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:
change_tz(change_time(t1, "NZ"), tz = "UTC")
change_tz(change_time(t1, "Europe/Berlin"), tz = "UTC")
change_tz(change_time(t1, "US/Eastern"), tz = "UTC")

# with vector of "POSIXlt" times:
t2 <- as.POSIXlt("2020-12-31 23:59:55", tz = "US/Pacific")
tv <- c(t1, t2)
tv # uses tz of t1
change_time(tv, "US/Pacific")
```

**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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

**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](#), [tb](#)

---

count\_chars

*count\_chars counts the frequency of characters in a string of text x.*

---

### Description

count\_chars counts the frequency of characters in a string of text x.

### 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.

**See Also**

[count\\_words](#) for counting the frequency of words; [plot\\_text](#) for a corresponding plot function.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_words\(\)](#), [l33t\\_ru135](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**Examples**

```
# Default:
x <- c("Hello!", "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\_words

*count\_words counts the frequency of words in a string of text x.*


---

**Description**

count\_words counts the frequency of words in a string of text x.

**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.

**See Also**

[count\\_chars](#) for counting the frequency of characters; [plot\\_text](#) for a corresponding plot function.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [l33t\\_ru135](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**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	<i>Current date (in yyyy-mm-dd or dd-mm-yyyy format).</i>
----------	---

---

**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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

**Examples**

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

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

---

cur_time	<i>Current time (in hh:mm or hh:mm:ss format).</i>
----------	--

---

**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**

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

**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()`, `is_leap_year()`, `what_date()`, `what_day()`, `what_month()`, `what_time()`, `what_week()`, `what_year()`

**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 data import (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](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`, `tb`

---

data_2	<i>Data import data_2.</i>
--------	----------------------------

---

**Description**

data\_2 is a fictitious dataset to practice data import (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](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](#), [tb](#)

---

data_t1	<i>Data table data_t1.</i>
---------	----------------------------

---

**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](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](#), [tb](#)

---

data_t1_de	<i>Data import data_t1_de.</i>
------------	--------------------------------

---

**Description**

data\_t1\_de is a fictitious dataset to practice data import (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](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](#), [tb](#)

---

data_t1_tab	<i>Data import data_t1_tab.</i>
-------------	---------------------------------

---

**Description**

data\_t1\_tab is a fictitious dataset to practice data import (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](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](#), [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](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](#), [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](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](#), [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](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](#), [tb](#)

---

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.

```

---

ds4psy.guide	<i>Opens user guide of the ds4psy package.</i>
--------------	--

---

### Description

Opens user guide of the ds4psy package.

### Usage

```
ds4psy.guide()
```

---

dt_10	<i>Data from 10 Danish people.</i>
-------	------------------------------------

---

### 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](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](#), [tb](#)

---

exp_num_dt	<i>Data from an experiment with numeracy and date-time variables.</i>
------------	---

---

### 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](#), [tb](#)

---

exp_wide	<i>Data exp_wide.</i>
----------	-----------------------

---

**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](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](#), [tb](#)

---

falsePosPsy_all	<i>False Positive Psychology data.</i>
-----------------	--

---

**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: <https://doi.org/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: <http://doi.org/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](#), [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 38 cases (rows) and 4 variables (columns).

## Source

Student solutions to exercises, dates from <https://en.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](#), [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](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](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](#), [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](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](#), [tb](#)

---

is.wholenumber	<i>Test for whole numbers (i.e., integers).</i>
----------------	---

---

**Description**

`is.wholenumber` tests if `x` contains only integer numbers.

**Usage**

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

**Arguments**

<code>x</code>	Number(s) to test (required, accepts numeric vectors).
<code>tol</code>	Numeric tolerance value. Default: <code>tol = .Machine\$double.eps^0.5</code> (see <code>?Machine</code> 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: [num\\_as\\_char\(\)](#), [num\\_as\\_ordinal\(\)](#)

### 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)
```

---

is_leap_year	<i>Is some year a so-called leap year?</i>
--------------	--

---

### 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 in two ways: 1. by verifying the numeric definition of a "leap year", and 2. by trying to use `as.Date()` for defining a "Date" of Feb-29 in the corresponding year(s).

### Source

See [https://en.wikipedia.org/wiki/Leap\\_year](https://en.wikipedia.org/wiki/Leap_year) for definition.

## See Also

leap\_year function of the **lubridate** package.

Other date and time functions: [change\\_time\(\)](#), [change\\_tz\(\)](#), [cur\\_date\(\)](#), [cur\\_time\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

## Examples

```
is_leap_year(2020)

# 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 would yield error
# is_leap_year("2021-02-29")
```

---

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_ru135), collapse = "")`.

New (i.e., replaced) characters are `paste(l33t_ru135, 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\(\)](#), [count\\_words\(\)](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

---

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

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

### Examples

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

---

metachar	<i>metachar provides R metacharacters (as a character vector).</i>
----------	--

---

**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.

**See Also**

[cclass](#) for a vector of character classes.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_rul35](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**Examples**

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

---

num_as_char	<i>Convert a number into a character sequence.</i>
-------------	--

---

**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.

**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.wholenumber\(\)](#), [num\\_as\\_ordinal\(\)](#)

**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 used:
num_as_char(11.33, n_pre_dec = 0, n_dec = 1)
num_as_char(11.66, n_pre_dec = 1, n_dec = 1)

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

# Beware of bad inputs:
```



```

num_as_char(4, sym = "8")
num_as_char(5, sym = "99")

# Works 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)

```

---

num_as_ordinal	<i>Convert a number into an ordinal character sequence.</i>
----------------	---

---

### 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, accepts numeric vectors).
sep	Decimal separator to use. Default: sep = "" (i.e., no separator).

### Details

The function currently only works for the English language and does not accept 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 not written efficiently or well.

### See Also

toOrdinal() function of the **toOrdinal** package.  
 Other utility functions: [is.wholenumber\(\)](#), [num\\_as\\_char\(\)](#)

### 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))
```

---

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](#), [tb](#)

---

pal_ds4psy	<i>ds4psy default color palette.</i>
------------	--------------------------------------

---

**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	<i>Get n-by-n dedicated colors of a color palette.</i>
----------	--

---

**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: <code>pal = pal_ds4psy</code> .

**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](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](#), [tb](#)

---

plot_fn	<i>A function to plot a plot.</i>
---------	-----------------------------------

---

### 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\\_fun\(\)](#), [plot\\_n\(\)](#), [plot\\_text\(\)](#), [plot\\_tiles\(\)](#), [theme\\_ds4psy\(\)](#)

**Examples**

```
# Basics:
plot_fun()

# Exploring options:
plot_fun(x = 2, A = TRUE)
plot_fun(x = 3, A = FALSE, E = TRUE)
plot_fun(x = 4, A = TRUE, B = TRUE, D = TRUE)
plot_fun(x = 5, A = FALSE, B = TRUE, E = TRUE, f = c("black", "white", "gold"))
plot_fun(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", Bordeaux). Note: Using colors of the <code>unikn</code> 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\\_fn\(\)](#), [plot\\_n\(\)](#), [plot\\_text\(\)](#), [plot\\_tiles\(\)](#), [theme\\_ds4psy\(\)](#)

### 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	<i>Plot n tiles.</i>
--------	----------------------

---

### 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 = <a href="#">pal_ds4psy</a> .
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\\_fn\(\)](#), [plot\\_fun\(\)](#), [plot\\_text\(\)](#), [plot\\_tiles\(\)](#), [theme\\_ds4psy\(\)](#)

## 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 in Console) into a table and then plots all its characters as a tile plot (using **ggplot2**).

## Usage

```

plot_text(
  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

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 = "".
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 unikn::Seebiau).
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: Should lower- and uppercase characters be distinguished? 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.

### See Also

[read\\_ascii](#) for reading text into a table; [pal\\_ds4psy](#) for default color palette.

Other plot functions: [plot\\_fn\(\)](#), [plot\\_fun\(\)](#), [plot\\_n\(\)](#), [plot\\_tiles\(\)](#), [theme\\_ds4psy\(\)](#)

**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) Plot text (from file):
# plot_text("test.txt")

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

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

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

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

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

## (c) Plot text input (from console):
# plot_text()

```

---

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 = <a href="#">pal_ds4psy</a> .
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\\_fn\(\)](#), [plot\\_fun\(\)](#), [plot\\_n\(\)](#), [plot\\_text\(\)](#), [theme\\_ds4psy\(\)](#)

**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 <https://doi.org/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](#), [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 <https://doi.org/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](#), [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 <https://doi.org/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](#), [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 <https://doi.org/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](#), [tb](#)

---

read\_ascii

*read\_ascii parses text (from a file) into a table.*

---

**Description**

`read_ascii` parses text (from a file or from user input in Console) into a table that contains a row for each character.

**Usage**

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

**Arguments**

<code>file</code>	The text file to read (or its path). If <code>file = ""</code> (the default), <code>scan</code> 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: <code>file = ""</code> .
<code>flip_y</code>	Boolean: Should y-coordinates be flipped, so that the lowest line in the text file becomes <code>y = 1</code> , and the top line in the text file becomes <code>y = n_lines</code> ? Default: <code>flip_y = FALSE</code> .

**Details**

`read_ascii` 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 this coordinate.

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.

**See Also**

[plot\\_text](#) for a corresponding plot function.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_rul35](#), [metachar](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**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", flip_y = 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	<i>Draw a sample of n random characters (from given characters).</i>
-------------	--

---

**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).

**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(n = 1, from = "1970-01-01", to = Sys.Date())
```

**Arguments**

n	Number dates to draw. Default: n = 1.
from	Earliest date (as string). Default: from = "1970-01-01".
to	Latest date (as string). Default: to = Sys.Date().

**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).

**See Also**

Other sampling functions: [coin\(\)](#), [dice\\_2\(\)](#), [dice\(\)](#), [sample\\_char\(\)](#), [sample\\_time\(\)](#)

**Examples**

```

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

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

```

---

<code>sample_time</code>	<i>Draw a sample of n random times (from a given range).</i>
--------------------------	--

---

**Description**

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

**Usage**

```

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

```

**Arguments**

<code>n</code>	Number dates to draw. Default: <code>n = 1</code> .
<code>from</code>	Earliest date (as string). Default: <code>from = "1970-01-01 00:00:00"</code> .
<code>to</code>	Latest date (as string). Default: <code>to = Sys.time()</code> .
<code>as_POSIXct</code>	Boolean: Return calendar time ("POSIXct") object? Default: <code>as_POSIXct = TRUE</code> . If <code>as_POSIXct = FALSE</code> , a local time ("POSIXlt") object is returned (as a list).
<code>tz</code>	Time zone. Default: <code>tz = ""</code> (i.e., current system time zone, see <code>Sys.timezone()</code> ). Use <code>tz = "UTC"</code> for Universal Time, Coordinated.

**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 `to = Sys.time()` (current time).

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.)

**See Also**

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

**Examples**

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

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

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

# Time zones:
sample_time(n = 3, tz = "UTC")
sample_time(n = 3, tz = "US/Pacific")

# Note: Oddity with sample():
sort(sample_time(n = 10, from = "2020-12-31 00:00:00 CET",
                 to = "2020-12-31 00:00:00 CET")) # 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](#), [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](#), [tb](#)

---

table6

*Data table6.*

---

**Description**

table6 is a fictitious dataset to practice tidying data.

**Usage**

table6

**Format**

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

**Details**

This dataset is a 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](#), [tb](#)

---

table7
--------

---

*Data table7.*

**Description**

table7 is a fictitious dataset to practice tidying data.

**Usage**

```
table7
```

**Format**

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

**Details**

This dataset is a 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](#), [tb](#)



---

table8	<i>Data table8.</i>
--------	---------------------

---

**Description**

table8 is a fictitious dataset to practice tidying data.

**Usage**

```
table8
```

**Format**

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

**Details**

This dataset is a 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](#), [tb](#)

---

tb	<i>Data table tb.</i>
----	-----------------------

---

**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](#)

---

text_to_sentences	<i>text_to_sentences splits a string of text x (consisting of one or more character strings) into a vector of its constituting sentences.</i>
-------------------	---

---

## Description

text\_to\_sentences splits at given punctuation marks (as a regular expression, default: split\_delim = "\\.|\\?|!") and removes empty leading and trailing spaces before returning a vector of the remaining character sequences (as the sentences).

## Usage

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

## Arguments

x	A string of text (required), typically a character vector.
split_delim	Sentence delimiters (as regex) used to split x into substrings. By default, split_delim = "\\. \\? !".
force_delim	Boolean: Enforce splitting at split_delim? If force_delim = FALSE (as per default), the function assumes a standard sentence-splitting pattern: split_delim is followed by a single space and a capital letter. If force_delim = TRUE, splits at split_delim are enforced (regardless of spacing or capitalization).

## Details

The Boolean `force_delim` distinguishes between two splitting modes:

1. If `force_delim = FALSE` (as per default), the function assumes a standard sentence-splitting pattern: A sentence delimiter in `split_delim` must be followed by a single space 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` uses `strsplit` to split strings.

## See Also

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

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars()`, `count_words()`, `l33t_rul35`, `metachar`, `read_ascii()`, `text_to_words()`, `transl33t()`

## 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)
text_to_sentences("Dr. Who is problematic.")
```

---

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

---

## Description

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

## Usage

```
text_to_words(x)
```

## Arguments

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

## Details

Internally, `text_to_words` uses `strsplit` to split strings.

## See Also

[text\\_to\\_sentences](#) for splitting text into a vector of sentences; [count\\_words](#) for counting the frequency of words; [strsplit](#) for splitting strings.

Other text objects and functions: [Umlaut](#), [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_rul35](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [transl33t\(\)](#)

## Examples

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

---

theme\_ds4psy

*ds4psy default plot theme (using ggplot2 and unikn).*

---

## Description

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

## Usage

```
theme_ds4psy(
  col_title = "black",
  base_size = 11,
  base_family = "",
  base_line_size = base_size/20,
  base_rect_size = base_size/20
)
```

**Arguments**

<code>col_title</code>	Color of title (text) elements (optional, numeric). Default: <code>col_title = "black"</code> . Consider using <code>col_title = unikn::pal_seeblau[[4]]</code> in combination with black or grey data points.
<code>base_size</code>	Base font size (optional, numeric). Default: <code>base_size = 11</code> .
<code>base_family</code>	Base font family (optional, character). Default: <code>base_family = ""</code> .
<code>base_line_size</code>	Base line size (optional, numeric). Default: <code>base_line_size = base_size/20</code> .
<code>base_rect_size</code>	Base rectangle size (optional, numeric). Default: <code>base_rect_size = base_size/20</code> .

**Details**

The theme is lightweight and no-nonsense, but somewhat opinionated (e.g., in using mostly grey scales to allow emphasizing data points with color accents).

**See Also**

`unikn::theme_unikn` for the source of the current theme.

Other plot functions: [plot\\_fn\(\)](#), [plot\\_fun\(\)](#), [plot\\_n\(\)](#), [plot\\_text\(\)](#), [plot\\_tiles\(\)](#)

**Examples**

```
# Plotting iris dataset (using ggplot2 and theme_ds4psy):

library("ggplot2") # theme_ds4psy requires loading ggplot2

ggplot(datasets::iris) +
  geom_jitter(aes(x = Petal.Length, y = Petal.Width, color = Species), size = 3, alpha = 2/3) +
  labs(title = "Iris species",
       caption = "Data from datasets::iris") +
  theme_ds4psy(col_title = "black", base_size = 11)
```

---

`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 = <code>l33t_rul35</code> .
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.

**See Also**

`l33t_rul35` for default rules used.

Other text objects and functions: `Umlaut`, `capitalize()`, `caseflip()`, `cclass`, `count_chars()`, `count_words()`, `l33t_rul35`, `metachar`, `read_ascii()`, `text_to_sentences()`, `text_to_words()`

**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 words frequently used by U.S. president Donald J. Trump (the 45th and current president of the United States, as of May 2020).

**Usage**

```
Trumpisms
```

**Format**

A vector of type character with `length(Trumpisms) = 96`.

**Source**

Data originally based on <https://www.yourdictionary.com/slideshow/donald-trump-20-most-frequently-used-words.html> and expanded by public speeches and Twitter tweets on <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](#), [tb](#)

---

t_1	<i>Data t_1.</i>
-----	------------------

---

**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](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](#), [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](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](#), [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](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](#), [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](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](#), [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\\_\(diacritic\)](https://en.wikipedia.org/wiki/Diaeresis_(diacritic)) and [https://en.wikipedia.org/wiki/Germanic\\_umlaut](https://en.wikipedia.org/wiki/Germanic_umlaut).

**See Also**

Other text objects and functions: [capitalize\(\)](#), [caseflip\(\)](#), [cclass](#), [count\\_chars\(\)](#), [count\\_words\(\)](#), [l33t\\_rul35](#), [metachar](#), [read\\_ascii\(\)](#), [text\\_to\\_sentences\(\)](#), [text\\_to\\_words\(\)](#), [transl33t\(\)](#)

**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\_day() 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

### 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("2020-12-24 01:02:03 CET", "2020-12-31 23:59:59")
what_date(ts)
what_date(ts, rev = TRUE, sep = ".")
what_date(ts, rev = TRUE, month_form = "b")

# with time zone:
ts <- ISOdate(2020, 12, 24, c(0, 12)) # midnight and midday UTC
what_date(when = ts, tz = "US/Hawaii")
```

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

---

what_day	<i>What day (of the week) is it?</i>
----------	--------------------------------------

---

## Description

what\_day provides a satisfying version of to determine the day of the week corresponding to a given date.

## Usage

```
what_day(when = Sys.Date(), abbr = 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.
abbr	Boolean: Return abbreviated? Default: abbr = FALSE.

## Details

what\_day returns 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

## Examples

```
what_day()
what_day(abbr = TRUE)

what_day(when = 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_day(when = ds)
what_day(when = ds, abbr = TRUE)

# with time vector (strings of POSIXct times):
```

```
ts <- c("2020-12-25 10:11:12 CET", "2020-12-31 23:59:59")
what_day(ts)
```

---

what_month	<i>What month is it?</i>
------------	--------------------------

---

## Description

what\_month provides a satisfying 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

## 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 satisfying 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_week\(\)](#), [what\\_year\(\)](#)

**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

# return "POSIXct" object(s):
t2 <- what_time("2020-02-29 12:30:45", as_string = FALSE, tz = "US/Hawaii")
format(t2, "%T %Z (UTF %z)")

```

---

what\_week

*What week is it?*


---

**Description**

what\_week provides a satisfying 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\_day() 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_year\(\)](#)

**Examples**

```

what_week()
what_week(as_integer = TRUE)

# Other dates/times:
d1 <- as.Date("2019-08-23")
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 satisfying 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\(\)](#), [is\\_leap\\_year\(\)](#), [what\\_date\(\)](#), [what\\_day\(\)](#), [what\\_month\(\)](#), [what\\_time\(\)](#), [what\\_week\(\)](#)

**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)
```

# Index

## \* color objects and functions

pal\_ds4psy, 35

pal\_n\_sq, 35

## \* data functions

make\_grid, 30

## \* datasets

Bushisms, 3

cclass, 5

countries, 10

data\_1, 14

data\_2, 15

data\_t1, 15

data\_t1\_de, 16

data\_t1\_tab, 16

data\_t2, 17

data\_t3, 18

data\_t4, 18

dt\_10, 21

exp\_num\_dt, 22

exp\_wide, 23

falsePosPsy\_all, 23

fame, 25

flowery, 26

fruits, 26

l33t\_rul35, 29

metachar, 31

outliers, 34

pal\_ds4psy, 35

pi\_100k, 36

posPsy\_AHI\_CESD, 45

posPsy\_long, 47

posPsy\_p\_info, 48

posPsy\_wide, 49

t3, 54

t4, 55

t\_1, 63

t\_2, 64

t\_3, 64

t\_4, 65

table6, 55

table7, 56

table8, 57

tb, 57

Trumpisms, 62

Umlaut, 65

## \* date and time functions

change\_time, 6

change\_tz, 7

cur\_date, 12

cur\_time, 13

is\_leap\_year, 28

what\_date, 66

what\_day, 68

what\_month, 69

what\_time, 70

what\_week, 71

what\_year, 72

## \* plot functions

plot\_fn, 37

plot\_fun, 38

plot\_n, 39

plot\_text, 41

plot\_tiles, 43

theme\_ds4psy, 60

## \* sampling functions

coin, 9

dice, 19

dice\_2, 20

sample\_char, 51

sample\_date, 52

sample\_time, 53

## \* text objects and functions

capitalize, 4

caseflip, 5

cclass, 5

count\_chars, 10

count\_words, 11

l33t\_rul35, 29

- metachar, 31
- read\_ascii, 50
- text\_to\_sentences, 58
- text\_to\_words, 59
- transl33t, 61
- Umlaut, 65
- \* **utility functions**
  - is.wholenumber, 27
  - num\_as\_char, 31
  - num\_as\_ordinal, 33
- Bushisms, 3, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- capitalize, 4, 5, 6, 11, 12, 30, 31, 50, 59, 60, 62, 66
- caseflip, 4, 5, 6, 11, 12, 30, 31, 50, 59, 60, 62, 66
- cclass, 4, 5, 5, 11, 12, 30, 31, 50, 59, 60, 62, 66
- change\_time, 6, 8, 13, 14, 29, 67–71, 73
- change\_tz, 7, 7, 13, 14, 29, 67–71, 73
- coin, 9, 19, 20, 52, 54
- count\_chars, 4–6, 10, 12, 30, 31, 50, 59, 60, 62, 66
- count\_words, 4–6, 11, 11, 30, 31, 50, 59, 60, 62, 66
- countries, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- cur\_date, 7, 8, 12, 14, 29, 67–71, 73
- cur\_time, 7, 8, 13, 13, 29, 67–71, 73
- data\_1, 4, 10, 14, 15–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_2, 4, 10, 14, 15, 16–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t1, 4, 10, 14, 15, 15, 16–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t1\_de, 4, 10, 14–16, 16, 17–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t1\_tab, 4, 10, 14–16, 16, 17–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t2, 4, 10, 14–17, 17, 18, 19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t3, 4, 10, 14–17, 18, 19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- data\_t4, 4, 10, 14–18, 18, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- dice, 9, 19, 20, 52, 54
- dice\_2, 9, 19, 20, 52, 54
- ds4psy.guide, 21
- dt\_10, 4, 10, 14–19, 21, 22, 23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- exp\_num\_dt, 4, 10, 14–19, 21, 22, 23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- exp\_wide, 4, 10, 14–19, 21, 22, 23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- falsePosPsy\_all, 4, 10, 14–19, 21–23, 23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- fame, 4, 10, 14–19, 21–23, 25, 25, 26, 27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- flowery, 4, 10, 14–19, 21–23, 25, 26, 27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- fruits, 4, 10, 14–19, 21–23, 25, 26, 26, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- is.integer, 27, 28
- is.wholenumber, 27, 32, 33
- is\_leap\_year, 7, 8, 13, 14, 28, 67–71, 73
- l33t\_rul35, 4–6, 11, 12, 29, 31, 50, 59, 60, 62, 66
- make\_grid, 30
- metachar, 4–6, 11, 12, 30, 31, 50, 59, 60, 62, 66
- num\_as\_char, 28, 31, 33
- num\_as\_ordinal, 28, 32, 33
- outliers, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- pal\_ds4psy, 35, 35, 36, 38–40, 42, 44
- pal\_n\_sq, 35, 35
- pi\_100k, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47, 49, 50, 55–58, 63–65
- plot\_fn, 37, 39, 40, 42, 44, 61
- plot\_fun, 38, 38, 40, 42, 44, 61
- plot\_n, 38, 39, 39, 42, 44, 61
- plot\_text, 11, 12, 38–40, 41, 44, 50, 61

- plot\_tiles, 36, 38–40, 42, 43, 61  
 posPsy\_AHI\_CESD, 4, 10, 14–19, 21–23,  
     25–27, 34, 36, 45, 47, 49, 50, 55–58,  
     63–65  
 posPsy\_long, 4, 10, 14–19, 21–23, 25–27, 34,  
     36, 46, 47, 49, 50, 55–58, 63–65  
 posPsy\_p\_info, 4, 10, 14–19, 21–23, 25–27,  
     34, 36, 46, 47, 48, 50, 55–58, 63–65  
 posPsy\_wide, 4, 10, 14–19, 21–23, 25–27, 34,  
     36, 46, 47, 49, 49, 55–58, 63–65  
  
 read\_ascii, 4–6, 11, 12, 30, 31, 42, 50, 59,  
     60, 62, 66  
  
 sample\_char, 9, 19, 20, 51, 52, 54  
 sample\_date, 9, 19, 20, 52, 52, 54  
 sample\_time, 9, 19, 20, 52, 53  
 strsplit, 59, 60  
  
 t3, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47,  
     49, 50, 54, 55–58, 63–65  
 t4, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47,  
     49, 50, 55, 55, 56–58, 63–65  
 t\_1, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46,  
     47, 49, 50, 55–58, 63, 63, 64, 65  
 t\_2, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46,  
     47, 49, 50, 55–58, 63, 64, 65  
 t\_3, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46,  
     47, 49, 50, 55–58, 63, 64, 64, 65  
 t\_4, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46,  
     47, 49, 50, 55–58, 63–65, 65  
 table6, 4, 10, 14–19, 21–23, 25–27, 34, 36,  
     46, 47, 49, 50, 55, 55, 56–58, 63–65  
 table7, 4, 10, 14–19, 21–23, 25–27, 34, 36,  
     46, 47, 49, 50, 55, 56, 56, 57, 58,  
     63–65  
 table8, 4, 10, 14–19, 21–23, 25–27, 34, 36,  
     46, 47, 49, 50, 55, 56, 57, 58, 63–65  
 tb, 4, 10, 14–19, 21–23, 25–27, 34, 36, 46, 47,  
     49, 50, 55–57, 57, 63–65  
 text\_to\_sentences, 4–6, 11, 12, 30, 31, 50,  
     58, 60, 62, 66  
 text\_to\_words, 4–6, 11, 12, 30, 31, 50, 59,  
     59, 62, 66  
 theme\_ds4psy, 38–40, 42, 44, 60  
 transl33t, 4–6, 11, 12, 30, 31, 50, 59, 60, 61,  
     66  
 Trumpisms, 4, 10, 14–19, 21–23, 25–27, 34,  
     36, 46, 47, 49, 50, 55–58, 62, 63–65  
  
 Umlaut, 4–6, 11, 12, 30, 31, 50, 59, 60, 62, 65  
  
 what\_date, 7, 8, 13, 14, 29, 66, 68–71, 73  
 what\_day, 7, 8, 13, 14, 29, 67, 68, 69–71, 73  
 what\_month, 7, 8, 13, 14, 29, 67, 68, 69, 70,  
     71, 73  
 what\_time, 7, 8, 13, 14, 29, 67–69, 70, 71, 73  
 what\_week, 7, 8, 13, 14, 29, 67–70, 71, 73  
 what\_year, 7, 8, 13, 14, 29, 67–71, 72