

Package ‘matRiks’

December 19, 2025

Type Package

Title Generates Raven-Like Matrices According to Rules

Version 0.1.5

Maintainer Andrea Brancaccio <andrea.brancaccio01@gmail.com>

Description Generates Raven like matrices according to different rules and the response list associated to the matrix.

The package can generate matrices composed of 4 or 9 cells, along with a response list of 11 elements (the correct response + 10 incorrect responses). The matrices can be generated according to both logical rules (i.e., the relationships between the elements in the matrix are manipulated to create the matrix) and visual-spatial rules (i.e., the visual or spatial characteristics of the elements are manipulated to generate the matrix).

The graphical elements of this package are based on the 'DescTools' package.

This package has been developed within the PRIN2020 Project (Prot. 20209WKCLL) titled ``Computerized, Adaptive and Personalized Assessment of Executive Functions and Fluid Intelligence" and founded by the Italian Ministry of Education and Research.

License MIT + file LICENSE

Encoding UTF-8

Imports DescTools

Suggests devtools, knitr, rmarkdown, testthat (>= 3.0.0), V8

Config/testthat/edition 3

RoxygenNote 7.3.2

VignetteBuilder knitr

NeedsCompilation no

Author Andrea Brancaccio [aut, ctb, cph, cre],
Ottavia M. Epifania [aut, ctb, com],
Debora de Chiusole [ctb]

Repository CRAN

Date/Publication 2025-12-19 17:10:03 UTC

Contents

axe	3
biscuit	4
change_color	5
circle	6
cof	7
correct	10
cross	11
decof	12
dice	13
difference	14
dot	15
draw	16
ellipse	19
hexagon	20
hide	21
hide.figure	21
ic	22
identity	25
lily	26
logic	27
luck	28
malta	29
margin	30
mat_apply	31
maxi	32
miley	33
ninja	34
pacman	35
pentagon	36
phantom	37
pizza_4	37
repetition	40
replace	41
response_list	42
rotate	43
semi_circle_bottom_inv	44
semi_circle_top	46
shade	47
shape	48
show	49
size	50
slice	51
split_mat	52
square	53
star	54
triangle	55

up_petal	57
vertical_eight	58
vertical_s	59
vert_bow_tie	61
vline	63
v_arc_left_up	66
wp	69

Index	71
--------------	-----------

axe	<i>Coordinates of an axe</i>
------------	------------------------------

Description

Define the coordinates for drawing an axe

Usage

```
axe(size.x = 15, pos.x = 0, pos.y = 0, lty = 1, lwd = 3, shd = NA)

s_axe(size.x = 15, pos.x = 0, pos.y = 0, lty = 1, lwd = 3, shd = NA)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 15
pos.x	numeric, define the position on the x axis. Default is 0
pos.y	numeric, define the position on the y axis. Default is 0
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3
shd	character, define the color of the figure. Default is NA, which results in a transparent figure

Value

Return the coordinates for drawing an axe

Return the coordinates for drawing a single axe

Functions

- **s_axe()**: Coordinates of a single axe
Define the coordinates for drawing a single axe, to be used in shape()

Examples

```
# return the default coordinates for drawing an axe
axe()
# change the coordinates for drawing a smaller single axe
axe(size.x = 5)
# return the default coordinates for drawing single axe
s_axe()
# change the coordinates for drawing a smaller single axe
s_axe(size.x = 5)
```

biscuit

Coordinates of a biscuit

Description

Define the coordinates for drawing a biscuit (composed of two hexagons)

Usage

```
biscuit(size.x = 10, size.y = size.x, shd = "black", lwd = 3, lty = 0)

s_biscuit(
  pos.x = 0,
  pos.y = 0,
  size.x = 10,
  size.y = size.x,
  shd = "black",
  lty = 1,
  lwd = 3
)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x
shd	character, define the shading of the figure. Default is black
lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0

Value

Return the coordinates for drawing a biscuit

Return the coordinates for drawing a single biscuit

Functions

- `s_biscuit()`: Coordinates of a single biscuit

Define the coordinates for drawing a single biscuit (composed of two hexagons), to be used in `shape()`

Examples

```
# return the default coordinates for drawing a biscuit
biscuit()
# change the shade of the biscuit
biscuit(shd = "grey", lty = 0)
# return the default coordinates for drawing a single biscuit
s_biscuit()
# change the shade of the single biscuit
biscuit(shd = "grey", lty = 0)
```

change_color

Change shade

Description

Change the shade of a figure

Usage

```
change_color(obj, ...)
## S3 method for class 'figure'
change_color(obj, ...)
```

Arguments

<code>obj</code>	The figure
<code>...</code>	other arguments

Value

Return the original figure with the inverted shade

Return the original figure with the inverted shade

Methods (by class)

- `change_color(figure)`: Change shade
Change the shade of a figure

Examples

```
# draw a square with inverted color
draw(change_color(square()))
draw(change_color(square()))
```

circle*Coordinates of a circle*

Description

Define the coordinates for drawing a circle

Usage

```
circle(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  shd = NA,
  vis = 1
)
```

Arguments

<code>size.x</code>	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
<code>size.y</code>	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is <code>size.x</code>
<code>pos.x</code>	numeric, position on the x axis. Default is 0
<code>pos.y</code>	numeric, position the y axis, Default is 0
<code>lty</code>	integer, define the line type of the figure, default is 1 (solid line).
<code>lwd</code>	integer, define the line width of the figure. Default is 3
<code>shd</code>	character, define the shading of the figure. Default is NA which results in a transparent figure
<code>vis</code>	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0

Value

Return the coordinates for drawing a circle

Examples

```
# return the default coordinates for drawing a circle
circle()
# change the coordinates for drawing a smaller circle
circle(size.x = 5)
```

cof

Concatenation of figures (method)

Description

Concatenation of different figures to create a new figure

Usage

```
cof(..., name, single)

## S3 method for class 'figure'
cof(..., name = NULL, single = FALSE)

## S3 method for class 'character'
cof(...)

com(...)

## S3 method for class 'matriks'
com(...)

concatenation(...)

## S3 method for class 'list'
concatenation(...)

## S3 method for class 'double'
concatenation(...)

## S3 method for class 'double'
cof(...)

## S3 method for class 'numeric'
cof(...)

## S3 method for class 'character'
concatenation(...)

## S3 method for class 'integer'
concatenation(...)
```

Arguments

...	The to be concatenated
name	character, name of the figure created with cof()
single	logical, force the figure to be a single figure to be used in shape(). Default is FALSE

Value

An object of class figure
An object of class figure
A concatenation of character
An object of class matriks resulting from the hierarchical concatenation of the original matrices
An object of class matriks resulting from the hierarchical concatenation of the original matrices

Methods (by class)

- **cof(figure):** Concatenation of figures (figures)
Concatenation of different figures to create a new figure
- **cof(character):** Concatenation of character
Concatenation of different figures to create a new figure
- **cof(double):** Concatenation of double
- **cof(numeric):** Concatenation of numeric

Functions

- **com():** Concatenation of matrices (Method)
Hierarchical concatenation of 2+ matrices on top of one another. The first matrix is placed on the bottom, the last matrix is placed on top of all other matrices.
- **com(matriks):** Concatenation of matrices
Hierarchical concatenation of 2+ matrices on top of one another. The first matrix is placed on the bottom, the last matrix is placed on top of all other matrices.
- **concatenation():** Concatenation (Method)
- **concatenation(list):** Concatenation of lists
- **concatenation(double):** Concatenation of double
- **concatenation(character):** Concatenation of characters
- **concatenation(integer):** Concatenation of stuff

Examples

```
# concatenate figures without creating a new figure
new_figure <- cof(square(), size(malta(), 2))
# structure of new_figure
str(new_figure)
# concatenate figures and create a new figure
```

```
my_figure <- cof(square(), size(malta(), 2),
                  single = TRUE,
                  name = "my_figure")
# structure of new_figure
str(my_figure)
# concatenate figures without creating a new figure
new_figure <- cof(square(), size(malta(), 2))
# structure of new_figure
str(new_figure)
# concatenate figures and create a new figure
my_figure <- cof(square(), size(malta(), 2),
                  single = TRUE,
                  name = "my_figure")
# structure of new_figure
str(my_figure)
# concatenate figures without creating a new figure
new_figure <- cof(square(), size(malta(), 2))
# structure of new_figure
str(new_figure)
# concatenate figures and create a new figure
my_figure <- cof(square(), size(malta(), 2),
                  single = TRUE,
                  name = "my_figure")
# structure of new_figure
str(my_figure)
# create the first layer matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# create the second matrix
m2 <- mat_apply(size(malta(), 2), vrules = "shade")
# concatenate the matrices
the_mat <- com(m1, m2)
# draw the final matrix
draw(the_mat)
# create the first layer matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# create the second matrix
m2 <- mat_apply(size(malta(), 2), vrules = "shade")
# concatenate the matrices
the_mat <- com(m1, m2)
# draw the final matrix
draw(the_mat)
# concatenate two characters
concatenation("a", "b")
# create some lists
a <- list(letters[c(14,13)], LETTERS[c(4, 3)])
b <- list(letters[c(12, 13)], LETTERS[c(4, 3)])
concatenation(a, b)
# create the first layer matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# create the second matrix
m2 <- mat_apply(size(malta(), 2), vrules = "shade")
# concatenate the matrices
the_mat <- com(m1, m2)
```

```

# draw the final matrix
draw(the_mat)
# create the first layer matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# create the second matrix
m2 <- mat_apply(size(malta(), 2), vrules = "shade")
# concatenate the matrices
the_mat <- com(m1, m2)
# draw the final matrix
draw(the_mat)
# concatenate two numeric
cof(rnorm(1, 25), rnorm(4, 34))
# concatenate two numeric
cof("a", "b", "d")
# concatenate two numeric
cof(1:3, 22:20)

```

correct*Correct response (Method)*

Description

Isolate the correct response from a matriks

Usage

```

correct(obj)

## S3 method for class 'matriks'
correct(obj)

```

Arguments

obj The matrix

Value

The correct response of a matriks

The correct response of a matriks

Methods (by class)

- **correct(matriks):** Correct response
Isolate the correct response from a matriks

Examples

```

# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat <- mat_apply(triangle(), mat.type = 9, hrule = "size")
# draw the matriks without the correct response
draw(my_mat, hide = TRUE)
# add the correct response
draw(correct(my_mat))

# apply the rotate rule on a pacman for creating a matriks with 4 cells
my_mat <- mat_apply(pacman(), mat.type = 4,
                     vrule = "rotate")
# draw the matriks without the correct response
draw(my_mat, hide = TRUE)
# add the correct response
draw(correct(my_mat))

# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat <- mat_apply(triangle(), mat.type = 9, hrule = "size")
# draw the matriks without the correct response
draw(my_mat, hide = TRUE)
# add the correct response
draw(correct(my_mat))

# apply the rotate rule on a pacman for creating a matriks with 4 cells
my_mat <- mat_apply(pacman(), mat.type = 4,
                     vrule = "rotate")
# draw the matriks without the correct response
draw(my_mat, hide = TRUE)
# add the correct response
draw(correct(my_mat))

```

cross

Coordinates of a cross

Description

Define the coordinates for drawing a cross

Usage

```

cross(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = size.x,
  lwd = 3,
  lty = 1
)
X(size.x = sqrt(square()$size.x[[1]]^2/2), size.y = size.x, lwd = 3, lty = 1)

```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is <code>sqrt(square(\$size.x[[1]])^2 / 2)</code>
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is <code>size.x</code> .
lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 1 (solid line)

Value

Return the coordinates for drawing a cross
 Return the coordinates for drawing an X

Functions

- `X()`: Coordinates of an X
 Define the coordinates for drawing an X

Examples

```
# default coordinates of an horizontal line
cross()
# draw a vertical line with different lty
draw(cross(lty = 2))
# default coordinates of an X
X()
# draw an X with different lty
draw(X(lty = 2))
```

Description

Return the elements composing a figure

Usage

```
decof(obj)
## S3 method for class 'figure'
decof(obj)
```

Arguments

obj	The figure of class figure to be split in its single components
-----	---

Value

A named list of figures of length equal to the total of shapes in a figure (both visible and not visible)
 A named list of figures of length equal to the total of shapes in a figure (both visible and not visible)

Methods (by class)

- `decof(figure)`: Split the elements of a figure
 Return the elements composing a figure

Examples

```
# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat1 <- mat_apply(triangle(), hrules = "size")
my_mat2 <- mat_apply(dot(), hrules = "shade")
my_mat <- com(my_mat1, my_mat2)
# Return the figures composing the first cell of the matriks
decof(my_mat$Sq2)
# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat1 <- mat_apply(triangle(), hrules = "size")
my_mat2 <- mat_apply(dot(), hrules = "shade")
my_mat <- com(my_mat1, my_mat2)
# Return the figures composing the first cell of the matriks
decof(my_mat$Sq2)
```

dice

*Coordinates of a dice with four dots***Description**

Define the coordinates for drawing four dots placed in the vertices of a square

Usage

```
dice(pos.x = 13, pos.y = 13, shd = "black", lwd = 3, lty = 1)

cross_dice(shd = "black", lwd = 3, lty = 1)
```

Arguments

<code>pos.x</code>	numeric, position on the x axis. Default is 13 (-13)
<code>pos.y</code>	numeric, position on the y axis. Default is 13 (-13)
<code>shd</code>	character, define the shading of the figure. Default is black
<code>lwd</code>	integer, define the line width of the figure. Default is 3
<code>lty</code>	integer, define the line type of the figure, default is 1 (solid line).

Value

Return the coordinates for drawing a dice with 4 dots

The coordinates for drawing a dice with 4 dots

Functions

- `cross_dice()`: Coordinates of a cross dice with four dots
Define the coordinates for drawing four dots placed in the vertices of a luck

Examples

```
# return the default coordinates for drawing a dot
dice()
# change the shade of the dice
dice(shd = "grey")
# return the default coordinates for drawing a dot
cross_dice()

# change the shade of the cross dice

cross_dice(shd = "grey")
```

difference

*Difference distractor (Method)***Description**

Generate difference distractor from a matriks

Usage

```
difference(obj, seed, ...)
## S3 method for class 'matriks'
difference(obj, seed = 666, ...)
```

Arguments

<code>obj</code>	matriks, The matriks for which the distractor is generated
<code>seed</code>	seed
<code>...</code>	other arguments

Value

An object of class figure that is the difference distractor of a matrix

An object of class figure that is the difference distractor of a matrix

Methods (by class)

- `difference(matriks)`: Difference distractors

Examples

```
# create a matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the difference distractor
draw(difference(m1))
# create a matrix
m1 <- mat_apply(hexagon(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the difference distractor
draw(difference(m1))
```

dot*Coordinates of a dot*

Description

Define the coordinates for drawing a dot

Usage

```
dot(
  size.x = 2,
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lwd = 3,
  lty = 1,
  shd = "black",
  vis = 1
)
```

Arguments

<code>size.x</code>	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 2
<code>size.y</code>	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is <code>size.x</code>
<code>pos.x</code>	numeric, position on the x axis. Default is 0
<code>pos.y</code>	numeric, position the y axis, Default is 0
<code>lwd</code>	integer, define the line width of the figure. Default is 3

lty	integer, define the line type of the figure, default is 1 (solid line).
shd	character, define the shading of the figure. Default is black
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0

Value

Return the coordinates for drawing a dot

Examples

```
# return the default coordinates for drawing a dot
dot()

# change the shade of the dot

dot(shd = "grey")
```

Description

Draws single figures, matrices with 9 or 4 cells, or response list of a matriks

Usage

```
draw(
  obj,
  main = NULL,
  canvas = TRUE,
  hide = FALSE,
  bg = "white",
  mar = c(1, 1, 1, 1),
  xlim = 16,
  ...
)

## S3 method for class 'figure'
draw(
  obj,
  main = NULL,
  canvas = TRUE,
  hide = FALSE,
  bg = "white",
  mar = c(1, 1, 1, 1),
  xlim = 16,
```

```
  ...
)

## S3 method for class 'matriks'
draw(
  obj,
  main = NULL,
  canvas = TRUE,
  hide = FALSE,
  bg = "white",
  mar = c(1, 1, 1, 1),
  xlim = 16,
  ...
)

## S3 method for class 'responses'
draw(
  obj,
  main = FALSE,
  canvas = TRUE,
  hide = FALSE,
  bg = "white",
  mar = c(1, 1, 1, 1),
  xlim = 16,
  distractors = NULL,
  labels = NULL,
  print = FALSE,
  frow = NULL,
  ...
)
```

Arguments

obj	The figure/matriks/response list to be drawn
main	logical, print the title of the drawing. Default is FALSE
canvas	logical, draw the figure on a new canvas. Default is TRUE
hide	logical, hide the cell corresponding to the correct response. Default is FALSE
bg	character, define the color background. Default is white
mar	numeric vector, change margins of the canvas
xlim	numeric, change the length of the x axis
...	other arguments
distractors	integer, default length of the response list. Works also with a character vector with the labels of the desired distractors.
labels	character, alternative labels to be printed as the main title of the distractors (main = TRUE). The labels must have the same length as the vector of distractors.
print	logical, print all the distractors together (default, FALSE) or one by one (TRUE)

frow numeric, vector of length 2 (nrow, ncol), response options will be drawn in a nrow x ncol array. Further details in ‘par()‘ documentation

Value

- A graphic
- A graphic of the figure
- A graphic of the matriks
- A graphic of the matriks

Methods (by class)

- **draw(figure):** Draw figure
Draw a figure
- **draw(matriks):** Draw Matriks
Draw a matriks
- **draw(responses):** Draw response list
Draw the response list of a matriks

Examples

```
# draw a circle
draw(circle())
# draw a circle inside the first circle
draw(size(circle(), 2), canvas = FALSE)
# draw a circle
draw.figure(circle())

# draw a circle inside the other
draw.figure(size(circle(), 2), canvas = FALSE)
# draw a matriks
my_mat <- mat_apply(cof(circle(), luck(), pacman()), "shade", "shape")
draw(my_mat)
# generate a matriks
my_mat1 <- mat_apply(cof(s_axe(), luck(), pacman()), "rotate", "shape")
my_mat2 <- mat_apply(dot(), "shade", "shape")
my_mat <- com(my_mat1, my_mat2)
# generate a response list
my_resp <- response_list(my_mat)
# draw response list
draw(my_resp)
```

ellipse	<i>Coordinates of an ellipse</i>
---------	----------------------------------

Description

Define the coordinates for drawing an ellipse

Usage

```
ellipse(  
  size.x = 10,  
  size.y = 7,  
  rot = 0,  
  shd = NA,  
  pos.x = 0,  
  pos.y = 0,  
  vis = 1,  
  lty = 1,  
  lwd = 3  
)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 7
rot	define the rotation. Default is 0
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing a ellipse

Examples

```
# return the default coordinates for drawing an ellipse  
ellipse()  
# change the coordinates for drawing a smaller ellipse  
ellipse(size.x = 5, size.y = 3)
```

hexagon	<i>Coordinates of a hexagon</i>
---------	---------------------------------

Description

Define the coordinates for drawing an hexagon

Usage

```
hexagon(
  size.x = 15,
  size.y = size.x,
  rot = 0,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  vis = 1,
  lty = 1,
  lwd = 3
)
```

Arguments

size.x	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 15
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x
rot	define the rotation. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing an hexagon

Examples

```
# return the default coordinates for drawing a hexagon
hexagon()
# change the coordinates for drawing a smaller hexagon
hexagon(size.x = 10)
```

hide	<i>Hide figures (Method)</i>
------	------------------------------

Description

Change the visibility of a figure from 1 to 0

Usage

```
hide(obj, index)
```

Arguments

obj	A figure composed of different figures
index	integer, the index of the element to hide

Value

The starting object with a hidden figure

Examples

```
# concatenate three figures into an object
my_shapes <- cof(square(), triangle(), slice())
# draw object
draw(my_shapes)
# hide the triangle
draw(hide(my_shapes, 2))
```

hide.figure	<i>Hide figures</i>
-------------	---------------------

Description

Change the visibility of a figure from 1 to 0

Usage

```
## S3 method for class 'figure'
hide(obj, index = "Full")
```

Arguments

obj	A figure composed of different figures
index	integer, the index of the element to hide

Value

The starting object with a hidden figure

Examples

```
# concatenate three figures into an object
my_shapes <- cof(square(), triangle(), slice())
# draw object
draw(my_shapes)
# hide the triangle
draw(hide(my_shapes, 2))
```

ic

Incomplete correlate distractors (method)

Description

Generate incomplete correlate flip distractor from a matriks

Usage

```
ic(obj)

## S3 method for class 'matriks'
ic(obj, ...)

ic_flip(obj, ...)

## S3 method for class 'matriks'
ic_flip(obj, ...)

ic_inc(obj, ...)

## S3 method for class 'matriks'
ic_inc(obj, ...)

ic_neg(obj, ...)

## S3 method for class 'matriks'
ic_neg(obj, ...)

ic_size(obj, ...)

## S3 method for class 'matriks'
ic_size(obj, ...)
```

Arguments

obj	matriks, The matriks for which the distractor is generated
...	other arguments

Value

An object of class responses of length 4, which contains the incomplete correlate distractors of a matriks (IC-Inc, IC-Flip, IC-Neg, IC-Size). If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class responses of length 4, which contains the incomplete correlate distractors of a matriks. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate flip distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate flip distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate incomplete distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate incomplete distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate negative distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate negative distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate size distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class figure that is the incomplete correlate size distractor of a matrix. If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

Methods (by class)

- **ic(matriks):** Incomplete correlate distractors
Generate incomplete correlate flip distractor from a matriks

Functions

- **ic_flip()**: Incomplete correlate flip distractor (method)
Generate incomplete correlate flip distractor from a matriks
- **ic_flip(matriks)**: Incomplete correlate flip distractor
Generate incomplete correlate flip distractor from a matriks
- **ic_inc()**: Incomplete correlate incomplete distractor (method)
Generate incomplete correlate incomplete distractor from a matriks
- **ic_inc(matriks)**: Incomplete correlate incomplete distractor
Generate incomplete correlate incomplete distractor from a matriks
- **ic_neg()**: Incomplete correlate negative distractor (method)
Generate incomplete negative incomplete distractor from a matriks
- **ic_neg(matriks)**: Incomplete correlate negative distractor
Generate incomplete negative incomplete distractor from a matriks
- **ic_size()**: Incomplete correlate size distractor (method)
Generate incomplete size incomplete distractor from a matriks
- **ic_size(matriks)**: Incomplete correlate size
Generate incomplete correlate size distractor of a matrix

Examples

```

# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
draw(mat)
# draw the incomplete correlate distractors
draw(ic(mat))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
draw(mat)
# draw the incomplete correlate distractors
draw(ic(mat))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the incomplete correalate flip distractor
draw(ic_flip(m1))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the incomplete correalate flip distractor

```

```

draw(ic_flip(m1))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
draw(mat)
# draw the incomplete correlate incomplete distractor
draw(ic_inc(mat))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
draw(mat)
# draw the incomplete correlate incomplete distractor
draw(ic_inc(mat))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the incomplete correlate negative distractor
draw(ic_neg(m1))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the matrix
draw(m1)
# draw the incomplete correlate negative distractor
draw(ic_neg(m1))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the incomplete correlate size distractor
draw(ic_size(m1))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
# draw the incomplete correlate size distractor
draw(ic_size(m1))

```

identity

Identity rule (Method)

Description

Apply an identity rule to the figures in a matrix (i.e., no changes)

Usage

```

identity(fig, ...)

## S3 method for class 'figure'
identity(fig, ...)

```

Arguments

fig Vector of figures obtained with the concatenation of figures function ('cof()').
 Three figures are needed.
 ... Other arguments

Value

An object composed of figures combined according to an identity rule

Methods (by class)

- `identity(figure)`: Identity figure

Examples

```
# generate a matrix with 9 squares
draw(mat_apply(square(), hrules = "identity"))
# generate a matrix with 9 squares
draw(mat_apply(square(), hrules = "identity"))
```

lily

Define the coordinates of a lily

Description

Define the coordinates for drawing the circle arches composing a lily

Usage

```
lily(lwd = 3, lty = 1)
s_lily(lwd = 3, lty = 1)
```

Arguments

lwd integer, define the line width of the figure. Default is 3
 lty integer, define the line type of the figure, default is 1 (solid line)

Value

Return the coordinates for drawing the circle arches composing a lily

Return the coordinates for drawing the circle arches composing a single lily, to be used in `shape()`

Functions

- `s_lily()`: Define the coordinates a single lily
 Define the coordinates for drawing the circle arches composing a single lily, to be used in `shape()`

Examples

```
# return the default coordinates drawing the circle arches composing a lily
lily()
# change the line type of the lily
lily(lty = 3)
# return the default coordinates for drawing a single lily
s_lily()
# change the line type of the single lily
s_lily(lty = 3)
```

logic	<i>Logical rules (Method)</i>
-------	-------------------------------

Description

Apply logical rules (intersection–AND, union–OR, symmetrical difference–XOR) to a concatenation of figures

Usage

```
logic(fig, n, rule, seed, ...)
## S3 method for class 'figure'
logic(fig, n = 1, rule = "logic", seed = 1, ...)
```

Arguments

fig	Vector of figures obtained with the concatenation of figures function ('cof()'). Three figures are needed.
n	integer, defines the elements of the logical expression. n=1 and n=2 are the concatenations of figures to which the logical operation is applied. n=3 is the result of the operation.
rule	character, logic rule to be applied, either 'AND', 'OR', 'XOR'
seed	integer, Set the random seed so that the permutations are consistent
...	Other arguments

Value

An object that is the logical combination of the figures

An object that is the logical combination of the figures

Methods (by class)

- `logic(figure)`: Logical rules
Apply logical rules (intersection–AND, union–OR, symmetrical difference–XOR) to a concatenation of figures

Examples

```
draw(logic(cof(square(), malta(), circle()), "AND"))
draw(logic(cof(square(), malta(), circle()), "AND"))
```

luck

Coordinates of a luck

Description

Define the coordinates for drawing a luck of the ellipse within which a luck can be inscribed.

Usage

```
luck(
  size.x = 10,
  size.y = 15,
  rot = pi/2,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  vis = 1,
  lty = 1,
  lwd = 3
)
luck4(size.x = 10, size.y = 7, lwd = 3, lty = 1)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 7
rot	define the rotation. Default is $\frac{\pi}{2}$
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing a luck
 Return the coordinates for drawing a luck composed of 4 lines

Functions

- `luck4()`: Coordinates of a luck composed of 4 lines
 Define the coordinates for drawing a luck composed of 4 lines

Examples

```
# return the default coordinates for drawing a luck
luck()
# change the coordinates for drawing a smaller luck
luck(size.x = 10, size.y = 15)
# default coordinates of an luck composed of 4 lines
luck4()
# draw a luck composed of 4 lines with different lty
draw(luck4(lty = 2))
```

malta

*Coordinates of a Malta cross***Description**

Define the coordinates for drawing a Malta cross

Usage

```
malta(size.x = 10, size.y = size.x, pos.x = 0, shd = NA, lwd = 3, lty = 1)

s_malta(size.x = 10, pos.x = 0, shd = NA, lwd = 3, lty = 1)
```

Arguments

<code>size.x</code>	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
<code>size.y</code>	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is <code>size.x</code>
<code>pos.x</code>	numeric, define the position on the x axis. Default is 0
<code>shd</code>	character, define the color of the figure. Default is NA, which results in a transparent figure
<code>lwd</code>	integer, define the line width of the figure. Default is 3
<code>lty</code>	integer, define the line type of the figure, default is 1 (solid line)

Value

Return the coordinates for drawing a Malta cross
 Return the coordinates for drawing a single Malta cross

Functions

- `s_malta()`: Coordinates of a single Malta cross
 Define the coordinates for drawing a single Malta cross, to be used in `shape()`

Examples

```
# return the default coordinates for drawing a Malta cross
malta()
# change the coordinates for drawing a smaller Malta cross
malta(size.x = 5)
# return the default coordinates for drawing a single Malta cross
s_malta()
# change the coordinates for drawing a smaller single Malta cross
s_malta(size.x = 5)
```

margin

*Margin rule (Method)***Description**

Apply a change in the margins of the figure

Usage

```
margin(fig, n, rule, ...)
## S3 method for class 'figure'
margin(fig, n, rule, ...)
```

Arguments

<code>fig</code>	The figure on which the rule is applied
<code>n</code>	integer, defines the linetype of the linewidth
<code>rule</code>	character, lty changes the linetype (1 = solid, 2 = dashed, 3 = dotted), lwd changes the linewidth
<code>...</code>	Other arguments

Value

A figure with changed margins
 A figure with changed margins

Methods (by class)

- `margin(figure)`: Change the margins rule
Apply a change in the margins of the figure

Examples

```
# draw default triangle
draw(triangle())

# change the linetype
draw(margin(triangle(), "lty", 2))
# draw default triangle
draw(triangle())

# change the linetype
draw(margin(triangle(),"lty", 2))
```

mat_apply*Apply rule to generate a matriks (method)*

Description

Apply a rule or a set of rules to a figure to create a matriks

Usage

```
mat_apply(Sq1, hrules = "identity", vrules = "identity", mat.type = 9)

## S3 method for class 'figure'
mat_apply(Sq1, hrules = "identity", vrules = "identity", mat.type = 9)
```

Arguments

<code>Sq1</code>	the figure(s) on which the rule should be applied for creating the matriks
<code>hrules</code>	character, the rule(s) to be applied horizontally. Default is identity
<code>vrules</code>	character, the rule(s) to be applied vertically. Default is identity
<code>mat.type</code>	integer, the type of matriks, either 4-cell matriks or 9-cell matriks (Default is 9)

Value

A list of length 7 (4-cell matriks) or of length 12 (9-cell matriks)

An object of class matriks of length 7 (4-cell matriks) or of length 12 (9-cell matriks)

Methods (by class)

- `mat_apply(figure)`: Apply rule to generate a matriks (method)
Apply a rule or a set of rules to a figure to create a matriks

Examples

```
# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat <- mat_apply(triangle(), mat.type = 9, hrule = "size")
# apply the size rule on a triangle for creating a matriks with 9 cell
my_mat <- mat_apply(triangle(), mat.type = 9, hrule = "size")
```

maxi

Coordinates of a maxi

Description

Define the coordinates for drawing a maxi (i.e., a cross composed of four lucks)

Usage

```
maxi(size.x = 8, size.y = 4, pos.x = 0, shd = NA, lty = 1, lwd = 3)

s_maxi(size.x = 8, size.y = 4, pos.x = 0, shd = NA, lty = 1, lwd = 3)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 8
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 4
pos.x	numeric, define the position on the x axis. Default is 0
shd	character, define the color of the figure. Default is NA, which results in a transparent figure
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing a maxi

Return the coordinates for drawing a maxi

Functions

- **s_maxi()**: Coordinates of a single maxi
Define the coordinates for drawing a single maxi (i.e., a cross composed of four lucks), to be used in shape()

Examples

```
# return the default coordinates for drawing a maxi
maxi()
# change the coordinates for drawing a smaller maxi
maxi(size.x = 5)
# return the default coordinates for drawing a single maxi
s_maxi()
# change the coordinates for drawing a smaller single maxi
s_maxi(size.x = 5)
```

miley

Define the coordinates of a miley

Description

Define the coordinates for drawing the petals composing a miley

Usage

```
miley(lwd = 3, lty = 1)

s_miley(lwd = 3, lty = 1)
```

Arguments

lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 1 (solid line)

Value

Return the coordinates for drawing the petals composing a miley

Return the coordinates for drawing the petals composing a single miley

Functions

- **s_miley()**: Define the coordinates a single miley
Define the coordinates for drawing the petals composing a single miley, to be used in shape()

Examples

```
# return the default coordinates for drawing a right petal
miley()
# change the line type of the right petal
miley(lty = 3)
# return the default coordinates for drawing the petals composing a single miley
s_miley()
# change the line type of the single miley
s_miley(lty = 3)
```

ninja*Coordinates of a ninja star*

Description

Define the coordinates for drawing a ninja star (composed of two lucks)

Usage

```
ninja(size.x = 10, size.y = 15, shd = "black", lwd = 3, lty = 0)  
s_ninja(size.x = 10, size.y = 15, shd = "black", lwd = 3, lty = 0)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 15
shd	character, define the shading of the figure. Default is black
lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 0

Value

Return the coordinates for drawing a ninja star

Return the coordinates for drawing a single ninja

Functions

- **s_ninja()**: Coordinates of a single ninja

Define the coordinates for drawing a single ninja star (composed of two lucks), to be used in `shape()`

Examples

```
# return the default coordinates for drawing a ninja
ninja()
# change the shade of the ninja
ninja(shd = "grey", lty = 0)
# return the default coordinates for drawing a single ninja
s_ninja()
# change the shade of the single ninja
s_ninja(shd = "grey", lty = 0)
```

pacman	<i>Coordinates of a pacman</i>
--------	--------------------------------

Description

Define the coordinates for drawing the circle sections for drawing a pacman

Usage

```
pacman(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = pi/4,
  theta2 = 7 * pi/4,
  lty = 1,
  lwd = 3,
  shd = NA,
  vis = 1
)
```

Arguments

size.x	integer, length of the semi-major axis of the ellipse within which the figure is inscribed. Default is $\sqrt{\text{square}() \$ \text{size.x}[[1]]^2 / 2}$
size.y	integer, length of the semi-minor axis of the ellipse within which the figure is inscribed. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
theta1	Starting angle of the circle section. Default is $\frac{\pi}{4}$
theta2	Ending angle of the circle section. Default is $\frac{7\pi}{4}$
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0

Value

Return the coordinates for drawing a pacman

Examples

```
# default coordinates of pacman
pacman()
# draw an actual pacman
draw(cof(pacman(shd = "yellow"), dot(pos.y = 6)))
```

pentagon

Coordinates of a pentagon

Description

Define the coordinates for drawing a pentagon

Usage

```
pentagon(
  size.x = 15,
  size.y = size.x,
  rot = pi/2,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  vis = 1,
  lty = 1,
  lwd = 3
)
```

Arguments

size.x	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 15
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x
rot	define the rotation. Default is $\frac{\pi}{2}$
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing a pentagon

Examples

```
# return the default coordinates for drawing a pentagon
pentagon()
# change the coordinates for drawing a smaller pentagon
pentagon(size.x = 10)
```

phantom

Coordinates of a panthom figure

Description

Draw an empty figure

Usage

```
phantom()
```

Value

An empty figure (nothing is plotted in draw)

Examples

```
# empty figure
phantom()
# draw an empty figure
draw(phantom())
```

pizza_4

Coordinates of a pizza with four slices

Description

Define the coordinates for drawing the circle sections composing a pizza with four slices

Usage

```
pizza_4(size.x = 15, shd = NA, lwd = 3, lty = 1)
s_pizza_4(size.x = 15, shd = NA, lwd = 3, lty = 1)

pizza_2(
  size.x = 15,
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
```

```

      shd = NA,
      lty = 1,
      lwd = 3
    )

s_pizza_2(
  size.x = 15,
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  lty = 1,
  lwd = 3
)

pizza_2_inv(
  size.x = 15,
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  lty = 1,
  lwd = 3
)

s_pizza_2_inv(
  size.x = 15,
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  lty = 1,
  lwd = 3
)

```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 15
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 1 (solid line)
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0

Value

Return the coordinates for drawing four circle sections composing a pizza with four slices

Return the coordinates for drawing four circle sections composing a singledocu pizza with four slices

Return the coordinates for drawing two circle sections composing a pizza with two slices

Return the coordinates for drawing two circle sections composing a single pizza with two slices

The coordinates of two circle sections composing an inverse pizza with two slices

The coordinates of two circle sections composing a single pizza with two slices

Functions

- `s_pizza_4()`: Coordinates of a single pizza with four slices

Define the coordinates for drawing the circle section composing a single pizza with four slices, to be used in `shape()`

- `pizza_2()`: Coordinates of a pizza with two slices

Define the coordinates for drawing the circle sections composing a pizza with two slices

- `s_pizza_2()`: Coordinates of a single pizza with two slices

Define the coordinates for drawing the circle section composing a single pizza with two slices, to be used in `shape()`

- `pizza_2_inv()`: Coordinates of an inverse pizza with two slices

Define the coordinates for drawing the circle sections composing an inverse pizza with two slices

- `s_pizza_2_inv()`: Coordinates of a single inverse pizza with two slices

Define the coordinates for drawing the circle sections composing an inverse pizza with two slices, to be used in `shape()`

Examples

```
# default coordinates of the pizza with four slices
pizza_4()
# default coordinates of the single pizza with four slices
s_pizza_4()
# default coordinates of the pizza with two slices
pizza_2()
# default coordinates of the single pizza with two slices
s_pizza_2()
# default coordinates of the inverse pizza with two slices
pizza_2_inv()
# default coordinates of the single inverse pizza with two slices
s_pizza_2_inv()
```

repetition	<i>Repetition distractors (Method)</i>
------------	--

Description

Generate repetition distractors from a matriks

Usage

```
repetition(obj, ...)

## S3 method for class 'matriks'
repetition(obj, ...)
```

Arguments

obj	matriks, The matriks for which the distractor is generated
...	other arguments

Value

An object of class responses of length 3, which contains the repetition distractors of a matriks (R-Left, R-Top, R-Diag). If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class responses of length 3, which contains the repetition distractors of a matriks (R-Left, R-Top, R-Diag). If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

Methods (by class)

- **repetition(matriks):** Repetition distractors (Method)

Generate repetition distractors from a matriks

Examples

```
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
draw(mat)
# draw the repetition distractors
draw(repetition(mat))
# create a matrix
m1 <- mat_apply(pacman(), hrules = "lty")
m2 <- mat_apply(dot(), "shade")
mat <- com(m1, m2)
# draw the matrix
```

```
draw(mat)
# draw the repetition distractors
draw(repetition(mat))
```

replace

Replace figures (Method)

Description

Replace a figure with another figure

Usage

```
replace(obj, index, replacement, visible)

## S3 method for class 'figure'
replace(obj, index, replacement, visible = FALSE)
```

Arguments

obj	A figure composed of different figures
index	integer, the index of the element to replace
replacement	The figure with which the original one is replaced
visible	logical, if TRUE it will replace only the visible figure. Default is FALSE

Value

An object with a changed figure
The starting object with a replaced figure
An object with a changed figure
The starting object with a replaced figure

Methods (by class)

- **replace(figure):** Replace figures
Replace a figure with another figure

Examples

```
# concanate three figures into an object
my_shapes <- cof(square(), triangle(), slice())
# draw object
draw(my_shapes)
# replace the square with a gray pacman
draw(replace(my_shapes, 1, pacman(shd = "grey")))
# concanate three figures into an object
my_shapes <- cof(square(), triangle(), slice())
```

```
# draw object
draw(my_shapes)
# replace the square with a gray pacman
draw(replace(my_shapes, 1, pacman(shd = "grey")))
```

response_list	<i>Response list (Method)</i>
---------------	-------------------------------

Description

Generate the response list from a matriks (correct response and distractors)

Usage

```
response_list(obj, seed, ...)
## S3 method for class 'matriks'
response_list(obj, seed = 666, ...)
```

Arguments

obj	matriks, The matriks for which the distractor is generated
seed	seed
...	other arguments

Value

An object of class responses of length 11, containing the correct response + 10 distractors (3 repetition, 1 difference, 2 wrong principles, 4 incomplete correlate)

An object of class responses of length 11, containing the correct response + 10 distractors (3 repetition, 1 difference, 2 wrong principles, 4 incomplete correlate)

Methods (by class)

- **response_list(matriks):** Response list
Generate the response list from a matriks (correct response and distractors)

Examples

```
# create a matrix
m1 <- mat_apply(hexagon(), hrules = "lty", vrules = "size")
# draw the matrix
draw(m1)
# draw the responses
draw(response_list(m1))

# change the difference distractor by changing the random seed
draw(response_list(m1, seed = 8))
```

```

# create a matrix
m1 <- mat_apply(hexagon(), hrules = "lty", vrules = "size")
# draw the matrix
draw(m1)
# draw the responses
draw(response_list(m1))

# change the difference distractors by changing the random seed
draw(response_list(m1, seed = 8))

```

rotate

Rotation rule (Method)

Description

Apply a rotation of a fixed angle to a figure

Usage

```

rotate(fig, n, rule, ...)
## S3 method for class 'figure'
rotate(fig, n = 4, rule = "rotation", ...)

```

Arguments

fig	The figure on which the rule is applied
n	integer, defines the angle of the rotation. Default is 4, which corresponds to a rotation of 4α
rule	character, defines the rotation rule. Default is counterclockwise. If the rule arguments contain the string "inv" forces a clockwise rotation. Each corresponds to an $\alpha = \frac{1}{k}\pi$. Default k is 4. To change the value of k is sufficient to add a number from 1 to 9 in the argument.
...	Other arguments

Value

A figure of class figure with different rotation coordinates
A figure of class figure with different rotation coordinates

Methods (by class)

- `rotate(figure)`: Rotate a figure
Apply a rotation of a fixed angle to a figure

Examples

```
# default luck
draw(luck())

# apply the default rotation on the default luck
draw(rotate(luck()))

# force clockwise rotation
draw(rotate(luck(), rule = "inv"))
# default luck
draw(luck())

# apply the default rotation on the default luck
draw(rotate(luck()))

# force clockwise rotation
draw(rotate(luck(), rule = "inv"))
```

semi_circle_bottom_inv

Coordinates of an upward-facing left semi-circle

Description

Define the coordinates for drawing an upward-facing left semi-circle

Usage

```
semi_circle_bottom_inv(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = 5 * pi/4,
  theta2 = pi/4,
  shd = NA,
  lty = 1,
  lwd = 3,
  vis = 1
)
semi_circle_bottom(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = 3 * pi/4,
  theta2 = 7 * pi/4,
```

```

  shd = NA,
  lty = 1,
  lwd = 3,
  vis = 1
)

```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is <code>sqrt(square()\$size.x[[1]]^2 / 2)</code>
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
theta1	Starting angle of the circle section. Default is <code>3*pi/4</code> .
theta2	Ending angle of the circle section (built counterclockwise). Default is <code>7*pi/4</code> .
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0

Value

The coordinates for drawing an upward-facing left semi-circle

The coordinates a upward-facing left semi-circle

Functions

- `semi_circle_bottom_inv()`: Coordinates of an upward-facing right semi-circle
Define the coordinates fr drawing an upward-facing right semi-circle

Examples

```

# default coordinates of the upward-facing right semi-circle
semi_circle_bottom_inv()
# change the rotation of the upward-facing right semi-circle
semi_circle_bottom_inv(theta1 = pi, theta2 = 2*pi)
# default coordinates of the upward-facing left semi-circle
semi_circle_bottom()
# change the rotation of the upward-facing left semi-circle
semi_circle_bottom(theta1 = pi, theta2 = 2*pi)

```

semi_circle_top	<i>Coordinates of a downward-facing left semi-circle</i>
-----------------	--

Description

Define the coordinates for drawing a downward-facing left semi-circle

Usage

```
semi_circle_top(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = pi/4,
  theta2 = 5 * pi/4,
  lty = 1,
  lwd = 3,
  shd = NA,
  vis = 1
)

semi_circle_top_inv(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = 7 * pi/4,
  theta2 = 3 * pi/4,
  shd = NA,
  lty = 1,
  lwd = 3,
  vis = 1
)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is $\sqrt{\text{square}() \cdot \text{size.x}[[1]]^2 / 2}$
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
theta1	Starting angle of the circle section. Default is $\frac{7\pi}{4}$
theta2	Ending angle of the circle section (built counterclockwise). Default is $\frac{3\pi}{4}$.

lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0

Value

Return the coordinates for drawing downward-facing left semi-circle

Return the coordinates for drawing a downward-facing right semi-circle

Functions

- `semi_circle_top_inv()`: Coordinates of a downward-facing right semi-circle
Define the coordinates for drawing a downward-facing right semi-circle

Examples

```
# default coordinates of the downward-facing left semi-circle
semi_circle_top()
# change the rotation of the downward-facing left semi-circle
semi_circle_top(theta1 = pi/2, theta2 = 3*pi/2)
# default coordinates of the downward-facing right semi-circle
semi_circle_top_inv()
# change the rotation of the downward-facing right semi-circle
semi_circle_top_inv(theta1 = 0, theta2 = pi/2)
```

shade

Shade rule (Method)

Description

Apply a change in the shading of the figure

Usage

```
shade(fig, n, rule, ...)
## S3 method for class 'figure'
shade(fig, n = 1, rule = "shade", ...)
```

Arguments

fig	The figure on which the rule is applied
n	integer, defines the color of the shading. Default is 1 (white). Other options are 2 (grey) and 3 (black)
rule	character, defines the rule for shading the figure
...	Other arguments

Value

An object of class figure with different shading characteristics
 An object of class figure with different shading characteristics

Methods (by class)

- `shade(figure)`: Change the shade of a figure
 Apply a change in the shading of the figure

Examples

```
# draw default triangle
draw(triangle())

# make it grey
draw(shade(triangle(), 2))
# draw default triangle
draw(triangle())

# make it grey
draw(shade(triangle(), 2))
```

shape*Shape rule (Method)*

Description

Apply a change in figures rule by change the visibility of the shapes in a figure

Usage

```
shape(fig, n, rule, ...)
## S3 method for class 'figure'
shape(fig, n = 1, rule = "shape", ...)
```

Arguments

<code>fig</code>	A vector of figures obtained with the concatenation of figures function (cof()). Three figures are needed
<code>n</code>	integer, the index of the element to see. Default is 1 (the first figure in cof() is shown). To see the other figures, change <code>n</code> to index the figure you want to show
<code>rule</code>	character, defines the rule for shading the figure
<code>...</code>	Other arguments

Value

An object of class figures, only the first figure is visible

Methods (by class)

- `shape(figure)`: Change the visible shapes

Examples

```
# Three figures, only the first is shown
draw(shape(cof(s_lily(), square(), s_star())))

# Show the third figure (star)
draw(shape(cof(s_lily(), square(), s_star()), n = 3))

# Show the first and the second figures
draw(shape(cof(s_lily(), square(), s_star()), n = c(1,2)))
```

show

Show figures (Method)

Description

Change the visibility of a figure from 0 to 1

Usage

```
show(obj, index)

## S3 method for class 'figure'
show(obj, index = "Full")
```

Arguments

<code>obj</code>	A figure composed of different figures
<code>index</code>	integer, the index of the element to hide

Value

The starting object with one more visible figure

The starting object with one more visible figure

Methods (by class)

- `show(figure)`: Show figures

Change the visibility of a figure from 0 to 1

Examples

```
# concatenate three figures into an object. The first figure is not visible
my_shapes <- cof(square(vis = 0), triangle(), slice())
# draw object
draw(my_shapes)
# show the square
draw(show(my_shapes, 1))
# concatenate three figures into an object. The first figure is not visible
my_shapes <- cof(square(vis = 0), triangle(), slice())
# draw object
draw(my_shapes)
# show the square
draw(show(my_shapes, 1))
```

size

Sizing rule (Method)

Description

Apply a resizing to a figure

Usage

```
size(fig, n, rule, ...)
## S3 method for class 'figure'
size(fig, n = 2, rule = "size", ...)
```

Arguments

fig	The figure on which the rule is applied
n	A number defining the dimension of the sizing. Default is 2.
rule	Define the sizing rule. Default is to reduce the dimension. rule = "inv" forces to increase the dimension.
...	Other arguments

Value

A figure of class figure with different size.x and size.y

Methods (by class)

- `size(figure)`: Resize a figure

Examples

```
# default square
draw(square())

# apply the default resizing to the default square
draw(size(square()))

# make the square bigger
draw(size(square(), rule = "inv"))
```

slice	<i>Coordinates of a pizza slice</i>
-------	-------------------------------------

Description

Define the coordinates for drawing a circle section

Usage

```
slice(
  size.x = 15,
  size.y = 0,
  pos.x = 0,
  pos.y = 0,
  theta1 = pi/4,
  theta2 = 3 * pi/4,
  lty = 1,
  lwd = 3,
  vis = 1,
  shd = NA
)
```

Arguments

size.x	integer, length of the semi-major axis of the ellipse within which the figure is inscribed. Default is 15
size.y	integer, length of the semi-major axis of the ellipse within which the figure is inscribed. Default is 0
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
theta1	Starting angle of the circle section. Default is $\frac{\pi}{4}$
theta2	Ending angle of the circle section (built counterclockwise). Default is $\frac{3\pi}{4}$
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3

vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
shd	character, define the shading of the figure. Default is NA which results in a transparent figure

Value

Return the coordinates for drawing a circle section

Examples

```
# default coordinates of the pizza slice
slice()
# change the rotation of the pizza slice
slice(theta1 = 3*pi/4, theta2 = 5*pi/4)
```

split_mat

Split the correct response (Method)

Description

Split all the visible figures composing a cell of the matrix or of a concatenation of figures

Usage

```
split_mat(obj, vis = TRUE, cell = NULL)

## S3 method for class 'figure'
split_mat(obj, vis = TRUE, cell = NULL)

## S3 method for class 'matriks'
split_mat(obj, vis = TRUE, cell = NULL)
```

Arguments

obj	The complex figure or the matrix to split
vis	logical, split only the visible figures. Default is TRUE
cell	integer, The index of the cell to be split. Default is the correct response

Value

A list of figures of length equal to the number of figures visible in the correct response (vis = TRUE) or to all the figures composing the complex figure (vis = FALSE)

A list of figures of length equal to the number of figures visible in the correct response (vis = TRUE) or to all the figures composing the complex figure (vis = FALSE)

A list of figures of length equal to the number of figures visible in the correct response (vis = TRUE) or to all the figures composing the complex figure (vis = FALSE)

Methods (by class)

- **split_mat(figure)**: Split the correct response
Split all the visible figures composing a cell of the matrix or of a concatenation of figures
- **split_mat(matriks)**: Split all the visible figures composing a cell of the matrix or a concatenation of figures

Examples

```
m1 <- mat_apply(hexagon(), hrules = "lty")
# split the elements in the correct response and assign to an object
split_m1 <- split_mat(m1$Sq1)
m1 <- mat_apply(hexagon(), hrules = "lty")
# split the elements in the correct response and assign to an object
split_m1 <- split_mat(m1$Sq1)
m1 <- mat_apply(hexagon(), hrules = "lty")
# split the elements in the correct response and assign to an object
split_m1 <- split_mat(m1)
```

square	<i>Coordinates of a square</i>
--------	--------------------------------

Description

Define the coordinates for drawing a square

Usage

```
square(
  size.x = 15,
  size.y = size.x,
  rot = pi/4,
  pos.x = 0,
  pos.y = 0,
  shd = NA,
  vis = 1,
  lty = 1,
  lwd = 3
)
square4(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = size.x,
  pos.x = size.x,
  pos.y = size.x,
  lwd = 3,
  lty = 1
)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is $\sqrt{\text{size.x}[1]^2 / 2}$
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x.
rot	define the rotation. Default is $\frac{\pi}{4}$
pos.x	numeric, position on the x axis. Default is 0.
pos.y	numeric, position the y axis, Default is 0.
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3.

Value

Return the coordinates for drawing a square

Return the coordinates for drawing a square composed of 4 lines

Functions

- `square4()`: Coordinates of a square composed of 4 lines
Define the coordinates for drawing a square composed of 4 lines

Examples

```
# return the default coordinates for drawing a square
square()
# change the coordinates for drawing a smaller square
square(size.x = 5)
# default coordinates of square composed of 4 lines
square4()
# draw square composed of 4 lines with different lty
draw(square4(lty = 2))
```

Description

Define the coordinates for drawing a star (composed of 4 luck)

Usage

```
star(size.x = 10, size.y = 15, shd = "black", lwd = 3, lty = 0)

s_star(size.x = 10, size.y = 15, shd = "black", lwd = 3, lty = 0)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is 15
shd	character, define the shading of the figure. Default is black
lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 0

Value

Return the coordinates for drawing star composed of four lucks
 Return the coordinates for drawing a single star composed of four lucks

Functions

- `s_star()`: Coordinates of a single star
 Define the coordinates for drawing a single star (composed of 4 luck), to be used in `shape()`

Examples

```
# get the coordinates of a star composed of four luck
star()

# change the color of the star
draw(star(shd = "grey", lty = 0))

# get the coordinates of a single star composed of four luck
s_star()

# change the color of the star
draw(s_star(shd = "grey", lty = 0))
```

Description

Define the coordinates for drawing a triangle

Usage

```
triangle(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  rot = pi/2,
  shd = NA,
  vis = 1,
  lty = 1,
  lwd = 3
)
```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
rot	define the rotation. Default is $\frac{\pi}{2}$
shd	character, define the shading of the figure. Default is NA which results in a transparent figure
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing a triangle

Examples

```
# return the default coordinates for drawing a triangle
triangle()
# change the coordinates for drawing a smaller triangle
triangle(size.x = 5)
```

up_petal	<i>Define the coordinates of petals</i>
----------	---

Description

Define the coordinates for drawing the circle arches composing some petals

Usage

```
up_petal(lwd = 3, lty = 1)  
  
down_petal(lwd = 3, lty = 1)  
  
left_petal(lwd = 3, lty = 1)  
  
right_petal(lwd = 3, lty = 1)
```

Arguments

lwd	integer, define the line width of the figure. Default is 3
lty	integer, define the line type of the figure, default is 1 (solid line)

Value

Return the coordinates for drawing the circle arches composing an up petal

Return the coordinates for drawing the circle arches composing a down petal

Return the coordinates for drawing the circle arches composing a left petal

Return the coordinates for drawing the circle arches composing a right petal

Functions

- `down_petal()`: Define the coordinates of a down petal
Define the coordinates for drawing the circle arches composing a down petal
- `left_petal()`: Define the coordinates of a left petal
Define the coordinates for drawing the circle arches composing a left petal
- `right_petal()`: Define the coordinates of a right petal
Define the coordinates for drawing the circle arches composing a right petal

Examples

```
# return the default coordinates for drawing the circle arches composing an up petal  
up_petal()  
# change the line type of the up petal  
up_petal(lty = 3)  
# return the default coordinates for drawing a down petal  
down_petal()
```

```

# change the line type of the down petal
down_petal(lty = 3)
# return the default coordinates for drawing a left petal
left_petal()
# change the line type of the left petal
left_petal(lty = 3)
# return the default coordinates for drawing a right petal
right_petal()
# change the line type of the right petal
right_petal(lty = 3)

```

vertical_eight *Eight-shaped figures*

Description

Define the coordinates for drawing eight-shaped figures vertical_eight defines the coordinates for drawing a vertical eight-shaped figures.

Usage

```

vertical_eight(lwd = 3, lty = 1)

horizontal_eight(lwd = 3, lty = 1)

s_vertical_eight(lwd = 3, lty = 1)

s_horizontal_eight(lwd = 3, lty = 1)

```

Arguments

lwd	integer, define the line width of the figure. Default is 3.
lty	integer, define the line type of the figure, default is 1 (solid line).

Value

Return the coordinates for drawing a vertical eight-shaped figure
 Return the coordinates for drawing an horizontal eight-shaped figure
 Return the coordinates for drawing a single vertical eight-shaped figure to be used in shape()
 Return the coordinates for drawing a single horizontal eight-shaped figure to be used in shape()

Functions

- **horizontal_eight()**: Coordinates of an horizontal eight
 Define the coordinates for drawing an horizontal eight-shaped figure
- **s_vertical_eight()**: Coordinates of a single vertical eight
 Define the coordinates for drawing a single vertical eight-shaped figure, to be used in shape()

- `s_horizontal_eight()`: Coordinates of a single horizontal eight
Define the coordinates for drawing a single vertical eight-shaped figure, to be used in `shape()`

Examples

```
# default coordinates of the vertical eight-shaped figure
vertical_eight()
# change the line type
vertical_eight(lty = 2)
# default coordinates of the horizontal eight-shaped figure
horizontal_eight()
# change the line type
horizontal_eight(lty = 2)
# default coordinates of the single vertical eight-shaped figure
s_vertical_eight()
# change the line type
s_vertical_eight(lty = 2)
# default coordinates of a single horizontal eight-shaped figure
s_horizontal_eight()
# change the line type
s_horizontal_eight(lty = 2)
```

vertical_s

Coordinates of S-shaped figures

Description

Define the coordinates for drawing S-shaped figures

Usage

```
vertical_s(lty = 1, lwd = 3)

vertical_s_inv(lty = 1, lwd = 3)

horizontal_s(lty = 1, lwd = 3)

horizontal_s_inv(lty = 1, lwd = 3)

s_vertical_s(lty = 1, lwd = 3)

s_vertical_s_inv(lty = 1, lwd = 3)

s_horizontal_s(lty = 1, lwd = 3)

s_horizontal_s_inv(lty = 1, lwd = 3)
```

Arguments

lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3.

Details

Define the coordinates of a vertical S-shaped figure

Value

- Return the coordinates for drawing a vertical S-shaped figure
- Return the coordinates for drawing an inverted vertical S-shaped figure
- Return the coordinates for drawing an horizontal S-shaped figure
- Return the coordinates for drawing an horizontal S-shaped figure
- Return the coordinates for drawing a vertical S-shaped figure
- Return the coordinates for drawing a single vertical S-shaped figure
- Return the coordinates for drawing a single horizontal S-shaped figure
- Return the coordinates for drawing a single inverted horizontal S-shaped figure

Functions

- **vertical_s_inv()**: Coordinates of an inverted vertical S-shaped figure
Define the coordinates of an inverted vertical S-shaped figure
- **horizontal_s()**: Coordinates of an horizontal S-shaped figure
Define the coordinates of an horizontal S-shaped figure
- **horizontal_s_inv()**: Coordinates of an inverted horizontal S-shaped figure
Define the coordinates of an inverted horizontal S-shaped figure
- **s_vertical_s()**: Coordinates of a single vertical S-shaped figure
Define the coordinates for drawing a single vertical S-shaped figure composed of two arches, which is forced to be a single figure (to be used in shape())
- **s_vertical_s_inv()**: Coordinates of a single inverted vertical S-shaped figure
Define the coordinates for drawing a single inverted vertical S-shaped figure composed of two arches, which is forced to be a single figure (to be used in shape())
- **s_horizontal_s()**: Coordinates of a single horizontal S-shaped figure
Define the coordinates for drawing a single horizontal S-shaped figure composed of two arches, which is forced to be a single figure (to be used in shape())
- **s_horizontal_s_inv()**: Coordinates of a single inverted horizontal S-shaped figure
Define the coordinates for drawing a single inverted horizontal S-shaped figure composed of two arches, which is forced to be a single figure (to be used in shape())

Examples

```
# default coordinates of the vertical S-shaped figure
vertical_s()
# change the line type
vertical_s(lty = 2)
# default coordinates of the inverted vertical S-shaped figure
vertical_s_inv()
# change the line type
vertical_s_inv(lty = 2)
# default coordinates of the horizontal S
horizontal_s()
# change the line type
horizontal_s(lty = 2)
# default coordinates of the horizontal S-shaped figure
horizontal_s_inv()
# change the line type
horizontal_s_inv(lty = 2)
# default coordinates of the vertical S-shaped figure
s_vertical_s()
# change the line type
s_vertical_s(lty = 2)
# default coordinates of the single inverted vertical S-shaped figure
s_vertical_s_inv()
# change the line type
s_vertical_s_inv(lty = 2)
# default coordinates of the single horizontal S-shaped figure
s_horizontal_s()
# change the line type
s_horizontal_s(lty = 2)
# default coordinates of the single inverted horizontal S-shaped figure
s_horizontal_s_inv()
# change the line type
s_horizontal_s_inv(lty = 2)
```

vert_bow_tie

Coordinates of bow ties

Description

Define the coordinates for drawing bow ties composed of two triangles

Usage

```
vert_bow_tie(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  shd = NA,
  lty = 1,
```

```

  lwd = 3
)

s_vert_bow_tie(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  shd = NA,
  lty = 1,
  lwd = 3
)

hor_bow_tie(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  shd = NA,
  lwd = 3,
  lty = 1
)

s_hor_bow_tie(
  size.x = 10,
  size.y = size.x,
  pos.x = 0,
  shd = NA,
  lwd = 3,
  lty = 1
)

```

Arguments

size.x	numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is 10
size.y	numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is size.x
pos.x	numeric, define the position on the x axis. Default is 0
shd	character, define the color of the figure. Default is NA, which results in a transparent figure
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3

Details

vert_bow_tie() Define the coordinates for drawing a vertical bow tie composed of two triangles

Value

Return the coordinates for drawing a vertical bow tie
Return the coordinates for drawing a single vertical bow tie
Return the coordinates for drawing a vertical bow tie
Return the coordinates for drawing a single horizontal bow tie

Functions

- `s_vert_bow_tie()`: Coordinates of a single vertical bow tie
Define the coordinates for drawing a single vertical bow tie composed of two triangles, to be used in `shape()`
- `hor_bow_tie()`: Coordinates of an horizontal bow tie
Define the coordinates for drawing an horizontal bow tie composed of two triangles
- `s_hor_bow_tie()`: Coordinates of a single horizontal bow tie
Define the coordinates for drawing a single horizontal bow tie composed of two triangles, to be used in `shape()`

Examples

```
# return the default coordinates for drawing a vertical bow tie
vert_bow_tie()
# change the coordinates for drawing a smaller bow tie
vert_bow_tie(size.x = 5)
# return the default coordinates for drawing a bow tie
s_vert_bow_tie()
# change the coordinates for drawing a smaller bow tie
s_vert_bow_tie(size.x = 5)
# return the default coordinates for drawing a vertical bow tie
hor_bow_tie()
# change the coordinates for drawing a smaller bow tie
hor_bow_tie(size.x = 5)
# return the default coordinates for drawing a single horizontal bow tie
s_hor_bow_tie()
# change the coordinates for drawing a smaller bow tie
s_hor_bow_tie(size.x = 5)
```

Description

Define the coordinates for drawing lines

Usage

```

vline(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  vis = 1
)

hline(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  vis = 1
)

diagline(
  size.x = list(sqrt(square()$size.x[[1]]^2/2)),
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  rotation = pi - pi/4,
  vis = 1
)

diagline_inv(
  size.x = sqrt(square()$size.x[[1]]^2/2),
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  rotation = pi + pi/4,
  vis = 1
)

```

Arguments

size.x numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is $\sqrt{\text{square}() \$ \text{size.x}[[1]]^2 / 2}$

size.y numeric, define the semi-minor axis of the ellipse within which the figure is

	inscribed. Default is size.x.
pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
lty	integer, define the line type of the figure, default is 1 (solid line).
lwd	integer, define the line width of the figure. Default is 3.
vis	integer, define the visibility of the figure (default is 1, visible)
rotation	define the rotation of the line

Details

vline() Define the coordinates for drawing a vertical line

Value

- Return the coordinates for drawing a vertical line
- Return the coordinates for drawing an horizontal line
- Return the coordinates for drawing the main diagonal line
- Return the coordinates for drawing the inverse diagonal line

Functions

- hline(): description Coordinates of an horizontal line
Define the coordinates for drawing an horizontal line
- diagline(): Coordinates of the main diagonal line
Define the coordinates for drawing the main diagonal line
- diagline_inv(): Coordinates of the inverse diagonal line
Define the coordinates for drawing the inverse diagonal line

Examples

```
# default coordinates of a vertical line
vline()
# draw a vertical line with different lty
draw(vline(lty = 2))
# default coordinates of an horizontal line
hline()
# draw a vertical line with different lty
draw(hline(lty = 2))
# default coordinates of the main diagonal line
diagline()
# draw the main diagonal line with different lty
draw(diagline(lty = 2))
# default coordinates of the inverse diagonal line
diagline_inv()
# draw the inverse diagonal line with different lty
draw(diagline_inv(lty = 2))
```

v_arc_left_up	<i>Coordinates for drawing circle arches</i>
---------------	--

Description

Define the coordinates for drawing different circle arches

Usage

```
v_arc_left_up(
  size.x = square()$size.x[[1]]/2,
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  vis = 1,
  lty = 1,
  lwd = 3
)

v_arc_right_up(
  size.x = square()$size.x[[1]]/2,
  size.y = size.x,
  pos.x = 0,
  pos.y = 0,
  lty = 1,
  lwd = 3,
  vis = 1
)

v_arc_left_down(
  size.x = square()$size.x[[1]]/2,
  size.y = size.x,
  lty = 1,
  lwd = 3,
  vis = 1,
  pos.x = 0,
  pos.y = 0
)

v_arc_right_down(
  size.x = square()$size.x[[1]]/2,
  size.y = size.x,
  lty = 1,
  lwd = 3,
  vis = 1,
  pos.x = 0,
  pos.y = 0
)
```

```
)  
  
h_arc_left_up(  
  size.x = square()$size.x[[1]]/2,  
  size.y = size.x,  
  lty = 1,  
  lwd = 3,  
  vis = 1,  
  pos.x = 0,  
  pos.y = 0  
)  
  
h_arc_right_up(  
  size.x = square()$size.x[[1]]/2,  
  size.y = size.x,  
  lty = 1,  
  lwd = 3,  
  vis = 1,  
  pos.x = 0,  
  pos.y = 0  
)  
  
h_arc_left_down(  
  size.x = square()$size.x[[1]]/2,  
  size.y = size.x,  
  lty = 1,  
  lwd = 3,  
  vis = 1,  
  pos.x = 0,  
  pos.y = 0  
)  
  
h_arc_right_down(  
  size.x = square()$size.x[[1]]/2,  
  size.y = size.x,  
  lty = 1,  
  lwd = 3,  
  vis = 1,  
  pos.x = 0,  
  pos.y = 0  
)
```

Arguments

`size.x` numeric, define the semi-major axis of the ellipse within which the figure is inscribed. Default is `square()$size.x[[1]]/2`

`size.y` numeric, define the semi-minor axis of the ellipse within which the figure is inscribed. Default is `size.x`

pos.x	numeric, position on the x axis. Default is 0
pos.y	numeric, position the y axis, Default is 0
vis	Visibility of the figure. Default is 1, making the figure visible. To hide the figure, change it to 0
lty	integer, define the line type of the figure, default is 1 (solid line)
lwd	integer, define the line width of the figure. Default is 3

Value

Return the coordinates for drawing the left up arch of a circle
 Return the coordinates for drawing the right up arch of a circle
 Return the coordinates for drawing the left down arch of a circle
 Return the coordinates for drawing the right down arch of a circle
 Return the coordinates for drawing the left up arch of a circle
 Return the coordinates for drawing the right up arch of a circle
 Return the coordinates for drawing the left down arch of a circle
 Return the coordinates for drawing the right down arch

Functions

- `v_arc_right_up()`: Coordinates of a vertical right up arch
 Define the coordinates for drawing the right up arch of a circle
- `v_arc_left_down()`: Coordinates of a vertical left down arch
 Define the coordinates for drawing the left down arch of a circle
- `v_arc_right_down()`: Coordinates of a vertical right down arch
 Define the coordinates for drawing the right down arch of a circle
- `h_arc_left_up()`: Coordinates of a horizontal left up arch
 Define the coordinates for drawing the left up arch of a circle
- `h_arc_right_up()`: Coordinates of a horizontal right up arch
 Define the coordinates for drawing the right up arch of a circle
- `h_arc_left_down()`: Coordinates of a horizontal left down arch
 Define the coordinates for drawing the left down arch of a circle
- `h_arc_right_down()`: Coordinates of a horizontal right down arch
 Define the coordinates for drawing the right down arch of a circle

Examples

```
# default coordinates of the left up arch
v_arc_left_up()
# default coordinates of the right up arch
v_arc_right_up()
# default coordinates of the left down arch
v_arc_left_down()
```

```
# default coordinates of the right down arch
v_arc_right_down()
# default coordinates of the left up arch
h_arc_left_up()
# default coordinates of the right up arch
h_arc_right_up()
# default coordinates of the left down arch
h_arc_left_down()
# default coordinates of the right down arch
h_arc_right_down()
```

wp

Wrong principle distractors (method)

Description

Generate the wrong principle distractors

Usage

```
wp(obj, ...)
## S3 method for class 'matriks'
wp(obj, ...)
```

Arguments

obj	The matriks
...	Other arguments

Value

An object of class responses that contains the wrong principle distractors of a matriks (WP-Matrix and WP-Copy). If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

An object of class responses that contains the wrong principle distractors of a matriks (WP-Matrix and WP-Copy). If the distractor could not be generated because of the constraints imposed by the matrix, it will be covered by a thick, black X and a warning is given.

Methods (by class)

- `wp(matriks)`: Wrong principle distractors
Generate the wrong principle distractors

Examples

```
m1 <- mat_apply(hexagon(), hrules = "lty")
# draw the matriks
draw(m1)
# draw the wp distractors with the title
draw(wp(m1), main = TRUE)
m1 <- mat_apply(hexagon(), hrules = "lty")
# draw the matriks
draw(m1)
# draw the wp distractors with the title
draw(wp(m1), main = TRUE)
```

Index

axe, 3
biscuit, 4
change_color, 5
circle, 6
cof, 7
com(cof), 7
concatenation(cof), 7
correct, 10
cross, 11
cross_dice(dice), 13
decof, 12
diagline(vline), 63
diagline_inv(vline), 63
dice, 13
difference, 14
dot, 15
down_petal(up_petal), 57
draw, 16
ellipse, 19
h_arc_left_down(v_arc_left_up), 66
h_arc_left_up(v_arc_left_up), 66
h_arc_right_down(v_arc_left_up), 66
h_arc_right_up(v_arc_left_up), 66
hexagon, 20
hide, 21
hide.figure, 21
hline(vline), 63
hor_bow_tie(vert_bow_tie), 61
horizontal_eight(vertical_eight), 58
horizontal_s(vertical_s), 59
horizontal_s_inv(vertical_s), 59
ic, 22
ic_flip(ic), 22
ic_inc(ic), 22
ic_neg(ic), 22
ic_size(ic), 22
identity, 25
left_petal(up_petal), 57
lily, 26
logic, 27
luck, 28
luck4(luck), 28
malta, 29
margin, 30
mat_apply, 31
maxi, 32
miley, 33
ninja, 34
pacman, 35
pentagon, 36
phantom, 37
pizza_2(pizza_4), 37
pizza_2_inv(pizza_4), 37
pizza_4, 37
repetition, 40
replace, 41
response_list, 42
right_petal(up_petal), 57
rotate, 43
s_axe(axe), 3
s_biscuit(biscuit), 4
s_hor_bow_tie(vert_bow_tie), 61
s_horizontal_eight(vertical_eight), 58
s_horizontal_s(vertical_s), 59
s_horizontal_s_inv(vertical_s), 59
s_lily(lily), 26
s_malta(malta), 29
s_maxi(maxi), 32
s_miley(miley), 33
s_ninja(ninja), 34

s_pizza_2 (pizza_4), 37
s_pizza_2_inv (pizza_4), 37
s_pizza_4 (pizza_4), 37
s_star (star), 54
s_vert_bow_tie (vert_bow_tie), 61
s_vertical_eight (vertical_eight), 58
s_vertical_s (vertical_s), 59
s_vertical_s_inv (vertical_s), 59
semi_circle_bottom
 (semi_circle_bottom_inv), 44
semi_circle_bottom_inv, 44
semi_circle_top, 46
semi_circle_top_inv (semi_circle_top),
 46
shade, 47
shape, 48
show, 49
size, 50
slice, 51
split_mat, 52
square, 53
square4 (square), 53
star, 54

triangle, 55

up_petal, 57

v_arc_left_down (v_arc_left_up), 66
v_arc_left_up, 66
v_arc_right_down (v_arc_left_up), 66
v_arc_right_up (v_arc_left_up), 66
vert_bow_tie, 61
vertical_eight, 58
vertical_s, 59
vertical_s_inv (vertical_s), 59
vline, 63

wp, 69

X (cross), 11