Package ‘webmorphR’

October 12, 2022

Title Reproducible Stimuli
Date 2022-06-01
Version 0.1.1
Description Create reproducible image stimuli, specialised for face images with 'psychomorph' or 'webmorph' templates.
License CC BY 4.0
BugReports https://github.com/debruine/webmorphR/issues
Depends R (>= 4.1.0)
Imports tools, grDevices, stats, dplyr, jsonlite, magick, geomorph, htr, progress, rsvg, grid, ggplot2
Suggests shiny, shinyjs, shinydashboard, shinyWidgets, DT, gifski, testthat, usethis, rmarkdown, covr
Encoding UTF-8
RoxygenNote 7.2.0
NeedsCompilation no
Author Lisa DeBruine [aut, cre, cph] (<https://orcid.org/0000-0002-7523-5539>), Emily Cunningham [pfr, ctb]
Maintainer Lisa DeBruine <debruine@gmail.com>
Repository CRAN
Date/Publication 2022-06-02 09:30:02 UTC

R topics documented:

  add_info ................................................................. 3
  align ................................................................. 4
  animate .............................................................. 6
  as_ggplot ............................................................ 7
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_stimlist .................................................</td>
</tr>
<tr>
<td>auto_delin ..................................................</td>
</tr>
<tr>
<td>average_tem ..................................................</td>
</tr>
<tr>
<td>avg ..........................................................</td>
</tr>
<tr>
<td>blank ........................................................</td>
</tr>
<tr>
<td>bounds .........................................................</td>
</tr>
<tr>
<td>centroid .......................................................</td>
</tr>
<tr>
<td>change_lines ..................................................</td>
</tr>
<tr>
<td>color_conv ....................................................</td>
</tr>
<tr>
<td>compare .........................................................</td>
</tr>
<tr>
<td>continuum ......................................................</td>
</tr>
<tr>
<td>crop ...........................................................</td>
</tr>
<tr>
<td>crop_tem .......................................................</td>
</tr>
<tr>
<td>delin ..........................................................</td>
</tr>
<tr>
<td>demo_stim ......................................................</td>
</tr>
<tr>
<td>draw_tem .......................................................</td>
</tr>
<tr>
<td>features .......................................................</td>
</tr>
<tr>
<td>get_info .......................................................</td>
</tr>
<tr>
<td>get_point ......................................................</td>
</tr>
<tr>
<td>gglabel .........................................................</td>
</tr>
<tr>
<td>greyscale ......................................................</td>
</tr>
<tr>
<td>height ..........................................................</td>
</tr>
<tr>
<td>horiz_eyes .....................................................</td>
</tr>
<tr>
<td>image_func .....................................................</td>
</tr>
<tr>
<td>image_func_types ...............................................</td>
</tr>
<tr>
<td>label ...........................................................</td>
</tr>
<tr>
<td>loop ............................................................</td>
</tr>
<tr>
<td>mask ...........................................................</td>
</tr>
<tr>
<td>mask_oval .......................................................</td>
</tr>
<tr>
<td>metrics ........................................................</td>
</tr>
<tr>
<td>mirror ..........................................................</td>
</tr>
<tr>
<td>mlabel ..........................................................</td>
</tr>
<tr>
<td>pad ..............................................................</td>
</tr>
<tr>
<td>patch ...........................................................</td>
</tr>
<tr>
<td>plot_rows .......................................................</td>
</tr>
<tr>
<td>plot_stim ......................................................</td>
</tr>
<tr>
<td>read_stim ......................................................</td>
</tr>
<tr>
<td>remove_tem .....................................................</td>
</tr>
<tr>
<td>rename_stim ....................................................</td>
</tr>
<tr>
<td>require_tem ....................................................</td>
</tr>
<tr>
<td>resize ..........................................................</td>
</tr>
<tr>
<td>rotate ..........................................................</td>
</tr>
<tr>
<td>same_tem ........................................................</td>
</tr>
<tr>
<td>social_media_size ..............................................</td>
</tr>
<tr>
<td>squash_tem .....................................................</td>
</tr>
<tr>
<td>subset_tem .....................................................</td>
</tr>
<tr>
<td>symmetrize .....................................................</td>
</tr>
<tr>
<td>tems_to_array ..................................................</td>
</tr>
</tbody>
</table>
Description

Add info with a data table that contains the info in either the same order as the stimulus list, or matching the stimuli item name with the column specified by `.by`.

Usage

```
add_info(stimuli, ..., .by = NULL)
```

Arguments

- **stimuli**: list of stimuli
- **...**: data table or named vectors of info to add
- **.by**: the column to use to match info to stimuli names; leave NULL if the data are to be matched by order

Details

You can also add data as named vectors.

Value

list of stimuli with info added

See Also

Other info: `compare()`, `get_info()`, `get_point()`, `height()`, `metrics()`, `rename_stim()`, `width()`
Examples

```r
stimuli <- demo_stim() |> 
  add_info(project = "XXX", gender = c("F", "M"))

stimuli$f_multi$info |> str()
```

---

**align**

*Align templates and images*

**Description**

Align images so that template points line up. Defaults to two-point alignment of the first two points in your template (usually the eyes) to their mean coordinate position across the stimuli.

**Usage**

```r
align(
  stimuli,
  pt1 = 0,
  pt2 = 1,
  x1 = NULL,
  y1 = NULL,
  x2 = NULL,
  y2 = NULL,
  width = NULL,
  height = NULL,
  ref_img = NULL,
  fill = wm_opts("fill"),
  procrustes = FALSE
)
```

**Arguments**

- `stimuli`: list of stimuli
- `pt1`: The first point to align (defaults to 0)
- `pt2`: The second point to align (defaults to 1)
- `x1, y1, x2, y2`: The coordinates to align the first and second point to
- `width, height`: The dimensions of the aligned images
- `ref_img`: The reference image to get coordinates and dimensions from if they are NULL
- `fill`: background color if cropping goes outside the original image, see `color_conv()`
- `procrustes`: logical; whether to use procrustes alignment
align

Details

Setting pt1 the same as pt2 aligns 1 point, but does not resize or rotate images. Setting pt1 and pt2 aligns 2 points, resizing and rotating faces. Setting procrustes = TRUE uses Procrustes analysis to resize and rotate images to be as close as possible to a mean shape.

You can specify the x and y coordinates to align, and the width and height of the output images, or set them from a reference image. The reference image (ref_img) can be a stim, a 1-item stimlist, or the index or name of a stim in stimuli. It defaults to average of all stimuli if NULL.

Visualise the template points with draw_tem() to determine which to align, using pt.shape = "index".

Value

list of stimuli with aligned images and/or templates

See Also

Stimulus manipulation functions crop_tem(), crop(), greyscale(), horiz_eyes(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()

Examples

# align eye points to specific x and y coordinates
# in a 300x300 pixel image
demo_unstandard(1:3) |>
align(pt1 = 0, pt2 = 1,
     x1 = 100, x2 = 200, y1 = 100, y2 = 100,
     width = 300, height = 300)

orig <- demo_unstandard()

# align to bottom-centre of nose (average position)
align(orig, pt1 = 55, pt2 = 55, fill = "dodgerblue")

# align to pupils of second image
align(orig, ref_img = 2, fill = "dodgerblue")

## Not run:
# procrustes align to average position
# this requires XQuartz on mac and may not run on linux
align(orig, procrustes = TRUE, fill = "dodgerblue")

## End(Not run)
animate

Create an animated gif from a list of stimuli

Description

Create an animated gif from a list of stimuli

Usage

animate(stimuli, fps = 1, loop = 0, rev = FALSE)

Arguments

stimuli list of stimuli
fps frames per second
loop how many times to loop the animation (0 = infinite)
rev whether to loop back and forth (TRUE) or in one direction (FALSE)

Value

magick image

See Also

Stimulus creating functions as_stimlist(), blank(), new_stimlist(), new_stim(), read_img(), read_stim(), read_tem(), write_stim()

Examples

# slideshow of images (1/second)
demo_stim() |> animate()

# rotate a face
degrees <- seq(0, 350, 10)
demo_stim(1) |> mask() |> rep(length(degrees)) |> rotate(degrees) |> animate(fps = 10)
as_ggplot

Convert stimuli to a ggplot

Description

Convert a stimulus or list of stimuli into a ggplot, which can be further used with ggplot functions.

Usage

as_ggplot(stimuli, ...)

Arguments

stimuli    list of stimuli
...        Additional arguments to pass to plot_stim() if stimuli contains more than 1 image

Value

a ggplot object

See Also

Visualisation functions draw_tem(), gglabel(), label(), mlabel(), plot_stimlist(), plot_stim(), plot_rows(), plot_stim()

Examples

stimuli <- demo_stim()
gg <- as_ggplot(stimuli)

# add to ggplot object; coordinates are pixels
# (images are 500x500 each, plus 10px padding)

  gg +
    ggplot2::geom_vline(xintercept = 0, color = "red") +
    ggplot2::geom_vline(xintercept = 1030, color = "blue") +
    ggplot2::geom_hline(yintercept = 0, color = "green") +
    ggplot2::geom_hline(yintercept = 520, color = "purple") +
    ggplot2::annotate("point", x = 515, y = 260, size = 10) +
    ggplot2::labs(
      title = "This is a ggplot!",
      caption = "Made with webmorphR"
    )
as_stimlist  
*Convert list to stimlist*

**Description**

Checks if an object is a stimulus or list of stimuli and repairs common problems.

**Usage**

```r
as_stimlist(x)
```

**Arguments**

- `x` The object

**Details**

Some webmorpHR functions, like `plot()` and `print()` require objects to have a "stimlist" class. If you've processed a list of stimuli with iterator functions like `lapply()` or `purrr::map()` and the resulting object prints or plots oddly, it is probably unclassed, and this function will fix that.

**Value**

A stimlist

**See Also**

Stimulus creating functions `animate()`, `blank()`, `new_stimlist()`, `new_stim()`, `read_img()`, `read_stim()`, `read_tem()`, `write_stim()`

**Examples**

```r
stimuli <- demo_stim() |> 
  lapply(function(stim) {
    # remove template lines
    stim$lines <- NULL
    return(stim)
  })

class(stimuli)

## Not run:
plot(stimuli) # error

## End(Not run)

s <- as_stimlist(stimuli)
class(s)
plot(s)
```
Auto-Delineation

Description

Automatically delineate faces using Face++ (an external service). Since each delineation counts against a daily limit, you need to set up your own Face++ account (see details below).

Usage

```r
auto_delin(stimuli, model = c("fpp106", "fpp83"), replace = FALSE, face = 1)
```

Arguments

- `stimuli`: list of stimuli
- `model`: Which model (fpp106, fpp83)
- `replace`: logical; whether to replace original templates - if FALSE, only gets templates for images with no template
- `face`: which face to delineate in each image if there is more than 1

Details

To use Face++ auto-delineation, you need to get your own free API key from https://www.faceplusplus.com.
After signing up for an account, go to https://console.faceplusplus.com/app/apikey/list and request a free API key. Add the key and secret to your .Renviron file as follows:

```r
FACEPLUSPLUS_KEY="1234567890abcdefghijk"
FACEPLUSPLUS_SECRET="1234567890abcdefghijk"
```

Value

list of stimuli with templates

See Also

Template functions `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `requiretems()`, `same_temps()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

Examples

```r
## Not run:
# requires an API key in .Renviron
demo_stim() |>
  auto_fpp106 <- auto_delin(model = "fpp106", replace = TRUE)

## End(Not run)
```
average_tem | Average templates

Description

This function just averages the templates. An average image is returned, but it is just all the images superimposed. To create a template-aware average, see `avg()`.

Usage

```r
average_tem(stimuli, name = "average")
```

Arguments

- `stimuli` list of stimuli
- `name` Name for the average

Value

list of stimuli consisting of just the average

See Also

Template functions `auto_delin()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

Examples

```r
tem_only_avg <- demo_stim() |> average_tem()

# view the average template
draw_tem(tem_only_avg, bg = "white")

# view the superimposed image
tem_only_avg
```

Description

Create an average from a list of delineated stimuli.
avg

Usage

avg(
  stimuli,
  texture = TRUE,
  norm = c("none", "twopoint", "rigid"),
  normpoint = 0:1
)

Arguments

stimuli list of stimuli to average
texture logical; whether textured should be averaged
norm how to normalise; see Details
normpoint points for twopoint normalisation

Details

Normalisation options:

- none: averages will have all coordinates as the mathematical average of the coordinates in
  the component templates
- twopoint: all images are first aligned to the 2 alignment points designated in normpoint.
  Their position is set to their position in the first image in stimuli
- rigid: procrustes aligns all images to the position of the first image in stimuli

Texture:
This applies a representative texture to the average, resulting in composite images with more
realistic texture instead of the very smooth, bland texture most other averaging programs create.
See the papers below for methodological details.

Value

list of stimuli with the average image and template

See Also

WebMorph.org functions continuum(), loop(), symmetrize(), trans(), webmorph_up()

Examples

if (webmorph_up()) {
  demo_stim() |> avg()
}
blank Make blank images

Description
Make blank images

Usage
blank(n = 1, width = 100, height = 100, color = "white", names = "img")

Arguments
- n: the number of images to return
- width, height: image dimensions
- color: background color, see color_conv()
- names: names of the images (appended with index if n)

Value
list of stimuli

See Also
Stimulus creating functions animate(), as_stimlist(), new_stimlist(), new_stim(), read_img(), read_stim(), read_tem(), write_stim()

Examples
stimuli <- blank(5, 100, 250, color = rainbow(5))
label(stimuli, size = 20)

bounds Get template bounds

Description
Get template bounds

Usage
bounds(stimuli, each = FALSE)
### centroid

**Description**

Get center coordinates

**Usage**

```r
centroid(stimuli, points = NULL)
```

**Arguments**

- `stimuli`: list of stimuli
- `points`: which points to include (0-based); if NULL, all points will be used

**Value**

named matrix of centroid x and y coordinates

**See Also**

Template functions: `auto_delin()`, `average_tem()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

**Examples**

```r
demo_stim() |> centroid()
```

# get the centre of the eye points
demo_stim() |> centroid(0:1)
change_lines

Change template lines

Description

Alter, add or remove lines in a template

Usage

change_lines(stimuli, line_id = 1, pts = NULL)

Arguments

stimuli 
list of stimuli
line_id 
index of the line to change
pts 
vector of points to change the line_idx to (deletes line if NULL)

Value

stimlist with altered templates

See Also

Template functions auto_delin(), average_tem(), centroid(), delin(), draw_tem(), features(), get_point(), remove_tem(), require_tem(), same_tem(), squash_tem(), subset_tem(), tem_def(), viz_tem_def()

Examples

# get image with dlib70 template and view lines
s <- demo_tems("dlib70")
s[[1]]$lines

# remove all lines
s2 <- change_lines(s, line_id = 1:13, pts = NULL)
s2[[1]]$lines

# visualise point indices
draw_tem(s2, pt.shape = "index", pt.size = 15)

# add a new line
s3 <- change_lines(s2, line_id = "face_outline",
                    pts = c(2:18, 28:19, 2))
s3[[1]]$lines
draw_tem(s3)
**color_conv**

*Convert colors*

**Description**

Convert from common color inputs to specified output type, adding alpha transparency for output formats that support it (hexa, rgba).

**Usage**

```r
color_conv(
  color,
  alpha = 1,
  from = c("guess", "col", "hex", "hexa", "hex3", "rgb", "rgba", "lab"),
  to = c("hexa", "hex", "rgba", "rgb", "lab", "hsv")
)
```

**Arguments**

- `color` A color in one of the input formats (see Details)
- `alpha` Alpha transparency (values <=1 converted to 0-255); ignored if color has alpha already
- `from`, `to` Input and output color spaces, see Details below.

**Details**

- color: one of the R colours listed in `grDevices::colors()`, e.g., "red"
- hex: hexadecimal string, e.g., "#FF0000"
- hexa: hexadecimal string with alpha, e.g., "#FF0000FF"
- hex3: abbreviated hexadecimal string, e.g., "#F00"
- rgb: vector of red, green and blue values 0-255, e.g., c(255, 0, 0)
- rgba: vector of red, green, blue and alpha values 0-255, e.g., c(255, 0, 0, 255)
- lab: CIE-Lab color
- hsv: vector of hue, saturation and value values (0-1), e.g., c(h=0, s = 1, v = 1)

**Value**

- color in to format

**See Also**

Other color: `col2lab()`, `lab2rgb()`
Examples

```r
# different ways to input red
color_conv("red")
color_conv("#FF0000")
color_conv("#FF0000FF")
color_conv(c(255,0,0))
color_conv("rgb(255,0,0)")  # you can use CSS-style text
color_conv(c(255,0,0,255))

# Lab must have names or use text format to be guessed
color_conv(c(l = 53.2, a = 80.1, b = 67.2))
color_conv("lab(53.2,80.1,67.2)")

# else, it will be guessed as rgb; fix by setting from explicitly
color_conv(c(53.2, 80.1, 67.2))
color_conv(c(53.2, 80.1, 67.2), from = "lab")

# add 50% alpha transparency to dodgerblue
color_conv("dodgerblue", alpha = 0.5, to = "rgba")
```

---

### Image Comparison

<table>
<thead>
<tr>
<th>compare</th>
<th>Image Comparison</th>
</tr>
</thead>
</table>

### Description

This is just a convenient way to use magick::compareare with webmorph stimuli. It defaults to the "MSE" metric, which gives a linearly increasing score to images along a morph continuum.

### Usage

```r
compare(stimuli, ref_stim, metric = "MSE", fuzz = 0, scale = FALSE)
```

### Arguments

- **stimuli**: Stimuli to compare to the ref_stim
- **ref_stim**: A stim, 1-item stimlist, or the name or index of the comparison item in stim
- **metric**: string with a metric from magick::metric_types(): "Undefined", "AE", "Fuzz", "MAE", "MEPP", "MSE", "NCC", "PAE", "PHASH", "PSNR", "RMSE"
- **fuzz**: relative color distance (value between 0 and 100) to be considered similar in the filling algorithm (only useful for AE)
- **scale**: whether to scale the values so that the maximum value is 1 and the minimum is 0 (only useful when stim is more than 1 image and includes ref_stim)
Details

Metric Types

- Undefined: ?
- AE: Absolute Error
- Fuzz: ?
- MAE: Mean Absolute Error
- MEPP: Mean Error Per Pixel
- MSE: Mean Squared Error
- NCC: Normalized Cross Correlation
- PAE: Peak Absolute Error
- PHASH: Perceptual Hash
- PSNR: Peak Signal-to-Noise Ratio
- RMSE: Root Mean Squared Error

How these metrics behave when comparing a morph continuum to its first image.

Increases with morph distance:

- very strong negative exponential decay at 0 fuzz; more linear with higher fuzz: AE
- strong negative exponential decay: PAE
- slight negative exponential decay: Fuzz, RMSE
- linear: MAE, MEPP, MSE
- no idea: PHASH

Decreases with morph distance:

- linear: NCC, Undefined
- slight exponential decay: PSNR

Value

Difference metric

See Also

Other info: add_info(), get_info(), get_point(), height(), metrics(), rename_stim(), width()

Examples

```r
stimuli <- demo_stim()
compare(stimuli, stimuli$m_multi)
compare(stimuli, stimuli$m_multi, "AE")
compare(stimuli, stimuli$m_multi, "AE", fuzz = 5)
```
continuum  Morph between two images

Description

Morph from one image to another in the specified steps.

Usage

continuum(from_img, to_img, from = 0, to = 1, by = 0.1, ...)

Arguments

- from_img: image to start at
- to_img: image to end at
- from: starting percentage
- to: ending percentage
- by: step size
- ...: arguments to pass to trans()

Value

a list of stimuli containing each step of the continuum

See Also

WebMorph.org functions avg(), loop(), symmetrize(), trans(), webmorph_up()

Examples

if (webmorph_up()) {
    stimuli <- demo_stim()
    cont <- continuum(stimuli$f_multi, stimuli$m_multi)

    # create an animated gif
    animate(cont, fps = 10, rev = TRUE)
}
crop

Crop images and templates

Description

Remove or add margins to images and templates.

Usage

crop(
  stimuli,
  width = 1,
  height = 1,
  x_off = NULL,
  y_off = NULL,
  fill = wm_opts("fill")
)

Arguments

  stimuli       list of stimuli
  width, height dimensions of cropped image in pixels or proportion (<2)
  x_off, y_off  offset in pixels or proportion (<1) (NULL centers cropped image)
  fill          background color if cropping goes outside the original image, see color_conv()

Details

Dimensions and offsets can be set in pixels or proportions. For width and height, values less than 2 will be interpreted as proportions, otherwise pixels. For x_off and y_off, values between -1 and 1 are interpreted as proportions, otherwise pixels.

Cropping is anchored at the image center (or calculated template centroid if there is no image) unless x_off or y_off are set.

Value

  list of stimuli with cropped tems and/or images

See Also

  Stimulus manipulation functions align(), crop_tem(), greyscale(), horiz_eyes(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()
Examples

stimuli <- demo_stim()

# crop to 60% width and 80% height (centered)
crop(stimuli, width = .60, height = .80)

# crop to upper right quadrant
crop(stimuli, .5, .5, x_off = .5, y_off = 0)

# negative offset with fill
crop(stimuli, 260, 260,
   x_off = -10, y_off = -10,
   fill = c("red", "dodgerblue"))

crop_tem

Crop to template boundaries and pad

Description

Calculate the maximum and minimum x and y coordinates across the stimuli (or for each stimulus) and crop all image to this plus padding.

Usage

crop_tem(
   stimuli,
   top = 10,
   right = top,
   bottom = top,
   left = right,
   each = FALSE,
   ...
)

Arguments

stimuli | list of stimuli
top, right, bottom, left | numeric; number of pixels or proportion (<1) to pad each side
each | logical; Whether to calculate bounds for the full set (default) or each image separately
...

Value

list of stimuli
See Also

Stimulus manipulation functions align(), crop(), greyscale(), horiz_eyes(), image_func(),
mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()

Examples

stimuli <- demo_stim()
ctem <- crop_tem(stimuli, each = TRUE)
draw_tem(ctem)

# demo with different templates
stimuli <- demo_tems()

# default 10 pixels around maximum template
crop_tem(stimuli)

# crop specific to each image
crop_tem(stimuli, each = TRUE)

---

delin

Manually delineate images

Description

Adjust the templates in a shiny interface. This will overwrite existing templates.

Usage

delin(stimuli)

Arguments

stimuli list of stimuli

Value

list of stimuli with new templates

See Also

Template functions auto_delin(), average_tem(), centroid(), change_lines(), draw_tem(),
features(), get_point(), remove_tem(), require_tem(), same_tem(), squash_tem(), subset_tem(),
tem_def(), viz_tem_def()
demo_stim

Examples
if (interactive()) {
  # adjust existing delineations
  stimuli <- demo_stim() |> delin()

  # create new delineations from scratch
  stimuli <- demo_stim() |> remove_tems() |> delin()
}

demo_stim  Demo Stimuli

Description
A convenience function to get demo stimuli. See the Details below for citation and license info.

Usage
demo_stim(pattern = NULL)
demo_tems(pattern = NULL)
demo_unstandard(pattern = NULL)

Arguments
pattern  Vector of patterns to use to search for files, or a vector of image indices (e.g.,
         1:4 selects the first 4 images and their templates)

Details
- demo_stim(): two composite faces with frl delineations; 500x500 pixels
- demo_tems(): an image with 5 different delineations; 675x900 pixels
- demo_unstandard(): a set of 10 composite faces with frl delineations; rotated, resized, and
cropped so face position is not standard and each image is a different size (444 to 645 pixels)

Citation:
The images from demo_stim() and demo_unstandard() are usable on a CC-BY license, citing:
The image from demo_tems() is Lisa DeBruine (the author of webmorphR) and available on a
CC-O license (no attribution needed).

Value
  list of stimuli
Examples

demo_stim() |> label()

# visualise templates
demo_tems() |> 
draw_tem(pt.size = 10) |> 
  label() |> 
  plot(maxwidth = 1000)

# visualise keeping relative sizes
demo_unstandard() |> 
to_size(keep_rels = TRUE) |> 
  pad(80, 0, 0, 0) |> 
  label() |> 
  plot(nrow = 2, maxwidth = 1000)

draw_tem

Draw template

Description

Visualise a template on an image.

Usage

draw_tem(
  stimuli,
  pt.color = wm_opts("pt.color"),
  pt.alpha = 0.75,
  pt.size = NULL,
  pt.shape = c("circle", "cross", "index"),
  line.color = wm_opts("line.color"),
  line.alpha = 0.5,
  line.size = NULL,
  bg = "image"
)

Arguments

stimuli : list of stimuli
pt.color, line.color : line or point color, see color_conv()
pt.alpha, line.alpha : transparency (0-1), ignored if color is a hex value with transparency. Set alpha to 0 to omit lines or points.
draw_tem

- **pt.size**, **line.size**: size in pixels (scales to image size if NULL)
- **pt.shape**: the shape of the points ("circle", "cross", "index")
- **bg**: background color ("image" uses the original image)

**Details**

Visualising the index of each point isn’t great yet and will overlay

**Value**

list of stimuli with template images

**See Also**

Template functions: `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

Visualisation functions: `as_ggplot()`, `gglabel()`, `label()`, `mlabel()`, `plot.stimlist()`, `plot.stim()`, `plot_rows()`, `plot_stim()`

**Examples**

```r
# get an image with 2 different templates
stimuli <- demo_tems("frl|fpp106")

# default template
draw_tem(stimuli)

# custom template
draw_tem(stimuli,
    pt.shape = "cross",
    pt.color = "red",
    pt.alpha = 1,
    pt.size = 15,
    line.color = rgb(0, 0, 0),
    line.alpha = 0.5,
    line.size = 5)

# indexed template
draw_tem(stimuli,
    pt.shape = "index",
    pt.size = 15,
    pt.alpha = 1,
    line.alpha = 0)
```
**features**

*Feature Points*

---

**Description**

Get point indices for features, usually for use with `subset_tem`.

**Usage**

```r
features(..., tem_id = c("frl", "dlib70"))
```

**Arguments**

- `...`: a vector of feature names (see Details)
- `tem_id`: template ID (currently only works for frl and dlib70)

**Details**

Available features for the frl template are: "gmm", "oval", "face", "mouth", "nose", "eyes", "brows", "left_eye", "right_eye", "left_brow", "right_brow", "ears", "undereyes", "teeth", "smile_lines", "cheek-bones", "philtrum", "chin", "neck", "halo".

Available features for the dlib70 template are: "teeth", "left_eye", "right_eye", "left_brow", "right_brow", "nose", "mouth", "face".

**Value**

vector of corresponding template indices

**See Also**

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `get_point()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

**Examples**

```r
features("mouth")
features("gmm")
features("nose", tem_id = "dlib70")
```
get_info  

Get Information

Description
Get Information

Usage
get_info(stimuli, ..., .rownames = "id")

Arguments
- stimuli: list of stimuli
- ...: column names to return
- .rownames: whether to return a table with no rownames (NULL), rownames from the list item names (NA), or as a new column (the column name as a string)

Value
a data frame or vector of the info

See Also
Other info: add_info(), compare(), get_point(), height(), metrics(), rename_stim(), width()

Examples
stimuli <- demo_stim() |>  
  add_info(project = "test", gender = c("F", "M"))

get_info(stimuli)
get_info(stimuli, "gender")

get_point  

Get Point Coordinates

Description
Get a data frame of the x and y coordinates of a template point

Usage
get_point(stimuli, pt = 0)
**Arguments**

- **stimuli**: list of stimuli with templates
- **pt**: point(s) to return

**Value**

data frame of x and y coordinates of the specified point(s) for each stimulus

**See Also**

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

Other info: `add_info()`, `compare()`, `get_info()`, `height()`, `metrics()`, `rename_stim()`, `width()`

**Examples**

demo_stim() |> get_point(0:1)
See Also

`label()` for a labeller using syntax like `magick::image_annotate`

Visualisation functions as `ggplot()`, `draw_tem()`, `label()`, `mlabel()`, `plot.stimlist()`, `plot.stim()`, `plot_rows()`, `plot_stim()`

Examples

```r
stimuli <- demo_stim()

# label with image names
# the default text size in ggplot is tiny
gglab(stimuli)

# add a watermark
gglab(stimuli,
  label = "watermark",
  x = 0.5,
  y = 0.5,
  geom = "text",
  size = 30,
  color = "black",
  angle = -30,
  alpha = 0.5)
```

---

**greyscale**

*Make images greyscale*

Description

Make images greyscale

Usage

```r
greyscale(stimuli)
```

Arguments

- `stimuli` list of class stimuli

Value

stimlist with new images
See Also

Stimulus manipulation functions align(), crop_tem(), crop(), horiz_eyes(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()

Stimulus manipulation functions align(), crop_tem(), crop(), horiz_eyes(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()

Examples

stimuli <- demo_stim()
grey_stim <- greyscale(stimuli)
plot(grey_stim)

<table>
<thead>
<tr>
<th>height</th>
<th>Image heights</th>
</tr>
</thead>
</table>

Description

Image heights

Usage

height(stimuli, type = c("all", "min", "max", "unique"))

Arguments

stimuli list of stimuli
type whether to return all heights, min, max, or only unique heights

Value

vector of heights

See Also

Other info: add_info(), compare(), get_info(), get_point(), metrics(), rename_stim(), width()

Examples

demo_stim() |> height()
horiz_eyes  

Make eyes horizontal

Description

Rotate each stimulus so the eye points are horizontal.

Usage

horiz_eyes(stimuli, left_eye = 0, right_eye = 1, fill = wm_opts("fill"))

Arguments

- stimuli: list of stimuli
- left_eye: The first point to align (defaults to 0)
- right_eye: The second point to align (defaults to 1)
- fill: background color to pass to rotate, see color_conv()

Value

list of stimuli with rotated tems and/or images

See Also

Stimulus manipulation functions align(), crop_tem(), crop(), greyscale(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate(), to_size()

Examples

stimuli <- demo_unstandard(1:3)
horiz_eyes(stimuli, fill = "red")

image_func  

Apply a magick function to each image

Description

This is a convenience function for applying magick functions that take an image as the first argument and return an image. It’s fully vectorised, so you can set separate argument values for each image.

Usage

image_func(stimuli, func, ...)
Arguments

- `stimuli` - list of stimuli
- `func` - the function or a string with the short name of the magick function (see `image_func_types()`)
- `...` - arguments to pass to the function

Details

These functions only affect the image, not the template. If a function changes the morphology of the image (e.g., "implode"), the template will not alter in the same way.

Value

List of stimuli with new images

See Also

Stimulus manipulation functions `align()`, `crop_tem()`, `crop()`, `greyscale()`, `horiz_eyes()`, `mask_oval()`, `mask()`, `mirror()`, `pad()`, `resize()`, `rotate()`, `to_size()`

Examples

```r
stimuli <- demo_stim() |> resize(0.5)

# make a photographic negative version
image_func(stimuli, "negate")

# set different argument values for each image
image_func(stimuli, "implode", factor = c(0.2, -0.2))

# other image functions
image_func(stimuli, "blur", 5, 3)
image_func(stimuli, "contrast", sharpen = 1)
image_func(stimuli, "oilpaint", radius = 5)
image_func(stimuli, "colorize", opacity = 50,
           color = c("hotpink", "dodgerblue"))

# load a logo image and superimpose it on each image
logo <- system.file("extdata/logo.png", package = "webmorphR") |>
      magick::image_read() |>
      magick::image_resize(70)
image_func(stimuli, "composite", logo, offset = "+5+10")

# use a self-defined function
testfunc <- function(image) {
  rot <- magick::image_rotate(image, 180)
  c(image, rot) |> magick::image_average()
}
image_func(stimuli, testfunc)
```
### image_func_types

**Possible functions**

**Description**

`image_func` can take a named function from the magick package, but only functions that return an image that is compatible with the current template (e.g., doesn’t change size or shape).

**Usage**

```r
image_func_types()
```

**Value**

list of compatible function names

**Examples**

```r
image_func_types()
```

---

### label

**Label images**

**Description**

Defaults to `mlabel()` unless you use arguments specific to `gglabel()`. All arguments are vectorised over the stimuli and values are recycled or truncated if there are fewer or more than stimuli.

**Usage**

```r
label(stimuli, ...)
```

**Arguments**

- `stimuli` list of stimuli
- `...` arguments to pass on to `mlabel()` or `gglabel()`

**Value**

stimlist with labelled images

**See Also**

`mlabel()`, `gglabel()`

Visualisation functions `as_ggplot()`, `draw_tem()`, `gglabel()`, `mlabel()`, `plot.stimlist()`, `plot.stim()`, `plot_rows()`, `plot_stim()`
Examples

stimuli <- demo_stim()

# label with magick::image_annotate
label(stimuli,
    text = c("CHINWE", "GEORGE"),
    gravity = c("north", "south"),
    color = "red")

# label with ggplot2::annotate
label(stimuli,
    label = c("CHINWE", "GEORGE"),
    x = 0.5,
    y = c(0.99, 0.02),
    vjust = c(1, 0),
    size = 18,
    color = "red")

---

loop  

---

Description

Morph between each image in a list of stimuli, looping back to the start.

Usage

loop(stimuli, steps = 10, ...)

Arguments

stimuli  

steps  

...  

Value

list of stimuli containing each step of the loop

See Also

WebMorph.org functions `avg()`, `continuum()`, `symmetrize()`, `trans()`, `webmorph_up()`
Examples

```r
if (webmorph_up()) {
    # align and crop images
    stimuli <- demo_unstandard(1:5) |>
        align() |> crop_tem()

    loop <- loop(stimuli, 5)

    # create an animated gif
    animate(loop, fps = 10)
}
```

---

**mask**

*Mask Images with templates*

**Description**

Use template points to define the borders of a mask to apply to the images. The image outside of the mask (or inside, if `reverse = TRUE`) is replaced by the fill color.

**Usage**

```r
mask(
    stimuli, 
    mask = "face",  
    fill = wm_opts("fill"), 
    reverse = FALSE, 
    expand = 1, 
    tem_id = "frl"
)
```

**Arguments**

- **stimuli**: list of stimuli
- **mask**: vector of masks or a custom list of template points
- **fill**: color to make the mask, see `color_conv()`
- **reverse**: logical; whether the mask covers the areas outside (FALSE) or inside (TRUE) the mask
- **expand**: how many pixels to expand the mask (negative numbers contract the mask)
- **tem_id**: template ID to pass on to `tem_def()` to get built-in mask definitions
Details

For FRL templates, the argument `mask` can be a vector with one or more of the following: oval, face, neck, ears (left_ear, right_ear), eyes (left_eye, right_eye), brows (left_brow, right_brow), mouth, teeth, nose.

For Face++ templates (fpp83 or fpp106), the argument `mask` can be a vector with one or more of the following: face, eyes (left_eye, right_eye), brows (left_brow, right_brow), mouth, teeth, nose. Because these templates have no forehead points, "face" is usually disappointing.

Set custom masks using the template points (0-based). View an image with labelled templates using `plot(stim, pt.plot = TRUE, pt.shape="index")`. Separate points along a line with commas, line segments with semicolons, and mask areas with colons. For example, this would be the custom mask for the eyes in the fpp83 template:

"44,4,56,51,79;79,58,11,25,44:61,67,38,34,40;40,41,17,47,61"

If you set `expand = 0`, there is sometimes a thin visible line where multiple components of the mask touch.

Value

list of stimuli with masked images

See Also

Stimulus manipulation functions `align()`, `crop_tem()`, `crop()`, `greyscale()`, `horiz_eyes()`, `image_func()`, `mask_oval()`, `mirror()`, `pad()`, `resize()`, `rotate()`, `to_size()`

Examples

```r
stimuli <- demo_stim()

mask(stimuli, 
     mask = c("face", "neck", "ears"),
     fill = "dodgerblue")

mask(stimuli, "face", expand = 30)

# reverse masking masks over the features
stimuli |> 
    mask("eyes", "#FFFF00", TRUE) |> 
    mask("brows", rgb(0.2, 0.5, 0.5), TRUE) |> 
    mask("mouth", "#FF000066", TRUE)

# custom mask (list style)
fpp83_eyes <- list(
    left_eye = list( 
        c(44,4,56,51,79),
        c(79,58,11,25,44) 
    ),
    right_eye = list( 
        c(61,67,38,34,40),
```
mask_oval

Apply an oval mask to images

Description

Superimpose an oval mask on a set of images.

Usage

mask_oval(stimuli, bounds = NULL, fill = wm_opts("fill"), each = TRUE)

Arguments

- **stimuli**: list of stimuli
- **bounds**: bounds (t, r, b, l) of oval, calculated from templates if NULL
- **fill**: background color for mask, see `color_conv()`
- **each**: logical; whether to calculate a mask for each image (default) or just one

Details

If the images have templates and `bounds = NULL`, the maximum and minimum x and y coordinates for each image will be calculated (or the overall max and min if each = FALSE) and an oval with those dimensions and position will be placed over the face.

If `bounds` are set to a list of top, right, bottom and left boundaries, these will be used instead of the boundaries derived from templates.

Value

list of stimuli with cropped tems and/or images

See Also

Stimulus manipulation functions `align()`, `crop_tem()`, `crop()`, `greyscale()`, `horiz_eyes()`, `image_func()`, `mask()`, `mirror()`, `pad()`, `resize()`, `rotate()`, `to_size()`
Examples

```r
# remove external template points and crop
stimuli <- demo_stim() |> subset_tem(features("face")) |> crop_tem(25)

# three styles of mask
omask1 <- mask_oval(stimuli) |> label("default")
omask2 <- mask_oval(stimuli, each = FALSE) |> label("each = FALSE")
omask3 <- mask_oval(stimuli, bounds = list(t= 50, r = 30, b = 40, l = 30)) |> label("manual bounds")

# visualise masks
c(omask1, omask2, omask3) |> plot(nrow = 2, byrow = FALSE)
```

---

### metrics

*Image shape metrics*

#### Description

Get metrics defined by template points.

#### Usage

```r
metrics(stimuli, formula = c(0, 1))
```

#### Arguments

- **stimuli**: list of stimuli with tems
- **formula**: a vector of two points to measure the distance apart, or a string of the formula for the metric

#### Details

Reference x and y coordinates by point number like `x[0]` or `y[188]`. Use any R functions to process the numbers, as well as `pow()` (same as `^()`, for consistency with webmorph.org). Remember that 0,0 is the top left for images; e.g., `min(y[0], y[1])` gives your the *higher* of the two pupil y-coordinates.

#### Value

named vector of the metric

#### See Also

Other info: `add_info()`, `compare()`, `get_info()`, `get_point()`, `height()`, `rename_stim()`, `width()`
Examples

stimuli <- demo_stim()

metrics(stimuli, c(0, 1)) # eye-spacing

# face width-to-height ratio
fwh <- "abs(max(x[113],x[112],x[114])-min(x[110],x[111],x[109]))/abs(y[90]-min(y[20],y[25]))"
metrics(stimuli, fwh)

Description

Use tem_id to get the symmetry map for your template. If tem_id is omitted, images and templates
will be fully reversed (e.g., if point 1 is the left eye in the original image, it will be the right eye in
the mirrored image).

Usage

mirror(stimuli, tem_id = NULL, axis = "vertical")

Arguments

stimuli list of stimuli
tem_id template ID to be passed to tem_def (usually "frl" or "fpp106") or NULL
axis vertical or horizontal axis of mirroring

Value

list of stimuli with mirrored images and templates

See Also

Stimulus manipulation functions align(), crop_tem(), crop(), greyscale(), horiz_eyes(),
image_func(), mask_oval(), mask(), pad(), resize(), rotate(), to_size()

Examples

# load an image and mirror it
o <- demo_tems("frl") |> resize(0.5)
m <- mirror(o, "frl")

# visualise the face outline points

c(o, m) |> subset_tem(features("face")) |> draw_tem(pt.shape = "index", pt.size = 15) |> label(c("original", "mirrored"))
mlabel  Label with magick annotations

Description

Label image using magick::image_annotate. All arguments are vectorised over the stimuli and values are recycled or truncated if there are fewer or more than stimuli. Setting a font, weight, style only works if your imagemagick is compiled with fontconfig support.

Usage

mlabel(
  stimuli,
  text = TRUE,
  gravity = "north",
  location = "+0+0",
  degrees = 0,
  size = 0.1,
  font = "sans",
  style = "normal",
  weight = 400,
  kerning = 0,
  decoration = NULL,
  color = "black",
  strokecolor = NULL,
  boxcolor = NULL
)

Arguments

stimuli  list of stimuli

  a vector of the label text(s) or TRUE to use stimlist names

text  string with gravity value from magick::gravity_types.

  geometry string with location relative to gravity

  rotates text around center point

  font size in pixels or proportion of image width (if < 1.0)

  string with font family such as "sans", "mono", "serif", "Times", "Helvetica", "Trebuchet", "Georgia", "Palatino" or "Comic Sans".

  value of magick::style_types(): "Undefined", "Any", "Italic", "Normal", "Oblique"

  thickness of the font, 400 is normal and 700 is bold.

  increases or decreases whitespace between letters

  value of magick::decoration_types(): "LineThrough" "None", "Overline", "Underline"
color: a vector of the label colour(s)
strokecolor: adds a stroke (border around the text)
boxcolor: adds a background color

Value
stimlist with labelled images

See Also

`gglabel()` for a labeller using syntax like `ggplot2::annotate()`
Visualisation functions `as_ggplot()`, `draw_tem()`, `gglabell()`, `label()`, `plot.stimlist()`, `plot.stim()`, `plot_rows()`, `plot_stim()`

Examples

```r
stimuli <- demo_stim()
mlabel(stimuli,
  text = c("CHINWE", "GEORGE"),
  gravity = c("north", "south"),
  color = "red")
```

Description

Add padding to sides of stimuli. This is a convenience function to calculate offsets for `crop()`.

Usage

```r
pad(stimuli, top = 10, right = top, bottom = top, left = right, ...)
```

Arguments

- `stimuli`: list of stimuli
- `top`, `right`, `bottom`, `left`: number of pixels or proportion (<1) to pad each side
- `...`: additional arguments to pass to `crop()`

Details

The value for top is copied to bottom and right, and the value for right is copied to left, so setting only top produces a consistent border, while setting just top and right sets different borders for top-bottom and right-left. (This convention will be familiar if you use CSS.)

Padding size values are interpreted as a proportion of width or height if less than 1.
Value

list of stimuli

See Also

Stimulus manipulation functions `align()`, `crop_tem()`, `crop()`, `greyscale()`, `horiz_eyes()`, `image_func()`, `mask_oval()`, `mask()`, `mirror()`, `resize()`, `rotate()`, `to_size()`

Examples

```r
stimuli <- demo_stim()

# default 10-pixel padding
pad(stimuli, fill = "dodgerblue")

# change pad width and set fill
pad(stimuli, 2, fill = "dodgerblue")

# set top border to 10% height
# different colour for each image
pad(stimuli, 0.1, 1, 1, 1,
   fill = c("hotpink", "dodgerblue"))
```

---

<table>
<thead>
<tr>
<th>patch</th>
<th>Patch colour</th>
</tr>
</thead>
</table>

Description

Get the median (or mean or user-defined function) colour value of a specified patch of pixels on an image. This is useful for matching background colours.

Usage

```r
patch(
  stimuli,
  width = 10,
  height = 10,
  x_off = 0,
  y_off = 0,
  color = c("hex", "rgb"),
  func = stats::median
)
```
Arguments

stimuli  list of stimuli
width, height  dimensions of the patch in pixels, if <=1, interpreted as proportions of width or height
x_off, y_off  offset in pixels or proportion (<1)
color  The type of color to return (hex, rgb)
func  The function to apply to an array of L*ab color values to determine the central colour (defaults to median, but mean, min, or max can also be useful)

Details

The colour values of each pixel in the patch are converted to CIE-Lab values before using the func to calculate the central tendency of the L (lightness), a (red-green axis) and b (blue-yellow axis); see \texttt{col2lab()} and \texttt{lab2rgb()} for more details.

This excludes transparent pixels, and returns "transparent" if all pixels in the patch are transparent.

Value

a vector of hex or rgba color values

Examples

```r
stimuli <- demo_stim()

# get colour from the upper left corner
patch(stimuli)

# get median colour from centre .1 width pixels
patch(stimuli, width = .1, height = .1,
      x_off = .45, y_off = .45)

# get mean rgb colour from full image
patch(stimuli, width = 1, height = 1,
      color = "rgb", func = mean)
```

Description

Plot in rows
plot_stim

Usage

plot_rows(
  ...,  # ... stimlists (optionally named) and any arguments to pass on to label
  top_label = NULL,  # logical; whether to plot row labels above the row (TRUE) or inside (FALSE), if NULL, then TRUE if stimlists are named
  maxwidth = wm_opts("plot.maxwidth"),  # maximum width and height of each row in pixels
  maxheight = wm_opts("plot.maxheight")
)

Arguments

...  # stimlists (optionally named) and any arguments to pass on to label
top_label  # logical; whether to plot row labels above the row (TRUE) or inside (FALSE), if NULL, then TRUE if stimlists are named
maxwidth, maxheight  # maximum width and height of each row in pixels

Value

stimlist with plot

See Also

Visualisation functions as_ggplot(), draw_tem(), gglab(), label(), mlabel(), plot.stimlist(), plot.stim(), plot_stim()

Examples

s <- demo_unstandard()
plot_rows(
  female = s[1:3],  # Show all the stimuli in a grid. You can use plot() as an alias.
  male = s[6:8],
  maxwidth = 600
)

plot_stim  # Plot stimuli

Description

Show all the stimuli in a grid. You can use plot() as an alias.
Usage

```r
plot_stim(
  stimuli,
  nrow = NULL,
  ncol = NULL,
  byrow = TRUE,
  padding = 10,
  external_pad = TRUE,
  fill = wm_opts("fill"),
  maxwidth = wm_opts("plot.maxwidth"),
  maxheight = wm_opts("plot.maxheight")
)
```

Arguments

- **stimuli**: list of class stimlist
- **nrow**: number of rows
- **ncol**: number of columns
- **byrow**: fill grid by rows (first ncol images in the first row); if FALSE, fills by columns (first nrow images in the first column)
- **padding**: around each image in pixels
- **external_pad**: whether to include external padding
- **fill**: background color, see `color_conv()`
- **maxwidth, maxheight**: maximum width and height of grid in pixels

Value

- stimlist with the plot image (no templates)

See Also

- Visualisation functions `as_ggplot()`, `draw_tem()`, `gglabel()`, `label()`, `mlabel()`, `plot.stimlist()`, `plot.stim()`, `plot_rows()`

Examples

```r
stimuli <- demo_stim() |> resize(0.5)
plot_stim(stimuli)
```

# default padding is 10px internal and external
plot(stimuli, fill = "dodgerblue")
plot(stimuli, external_pad = 0, fill = "dodgerblue")
plot(stimuli, padding = 0, fill = "dodgerblue")

# make 8 numbered images
n <- blank(8, color = grDevices::cm.colors(8)) |>
read_stim

label(1:8, gravity = "center", size = 50)

# 2 rows, allocating by row
plot(n, nrow = 2)

# 2 rows, allocating by column
plot(n, nrow = 2, byrow = FALSE)

---

**Description**

Read images and templates from a directory.

**Usage**

```r
read_stim(path, pattern = NULL, breaks = "/")
```

**Arguments**

- `path` Path to directory containing image and/or template files (or a single file path)
- `pattern` Vector of patterns to use to search for files, or a vector of image indices (e.g., 1:4 selects the first 4 images and their templates if they exist)
- `breaks` a vector of characters used to determine the stimulus names from the file names

**Value**

a list of stimuli

**See Also**

Stimulus creating functions `animate()`, `as_stimlist()`, `blank()`, `new_stimlist()`, `new_stim()`, `read_img()`, `read_tem()`, `write_stim()`

**Examples**

```r
path <- system.file("extdata/test", package = "webmorphR")

# read in all images and templates in a directory
stimuli <- read_stim(path)

# read in just images and templates with "m_
mm_stimuli <- read_stim(path, "m_")
```
rename_stim

---

### remove_tem

**Remove templates**

**Description**

Remove templates

**Usage**

```r
remove_tem(stimuli)
```

**Arguments**

- **stimuli**
  - list of stimuli

**Value**

- list of stimuli

**See Also**

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

**Examples**

```r
demo_stim() |> remove_tem()
```

---

### rename_stim

**Set stimulus names in a stimlist**

**Description**

Set stimulus names in a stimlist

**Usage**

```r
rename_stim(
  stimuli,
  new_names = NULL,
  prefix = "",
  suffix = "",
  pattern = NULL,
  replacement = NULL,
  ...)
```
**require_tems**

**Description**

Checks a list of stimuli for templates and omits images without a template. If `all_same = TRUE`, checks that all the templates are the same type. Errors if no images have a template or not all templates are the same (when `all_same == TRUE`).

**Usage**

```r
require_tems(stimuli, all_same = FALSE)
```

**Arguments**

- `stimuli`: list of stimuli
- `all_same`: logical; whether all images should have the same template

**Value**

list of stimuli with tems
See Also

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `same_tem()`, `squizha_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

Examples

```r
stimuli <- demo_stim()
have_temps <- require_temps(stimuli)

## Not run:
# produces an error because no tems
no_temps <- stimuli |> remove_tem()
require_temps(no_temps)

# warns that some images were removed
mix_temps <- c(stimuli, no_temps)
have_temps <- require_temps(mix_temps)

# produces an error because tems are different
demo_temps() |> require_temps(all_same = TRUE)

## End(Not run)
```

---

**resize**

**Resize stimuli**

Description

Resize images and templates to the specified width and height.

Usage

```r
resize(stimuli, width = NULL, height = NULL)
```

Arguments

- **stimuli**: list of stimuli
- **width, height**: new dimensions (in pixels or percent if < 10)

Value

list of stimuli with resized tems and/or images

See Also

Stimulus manipulation functions `align()`, `crop_tem()`, `crop()`, `greyscale()`, `horiz_eyes()`, `image_func()`, `mask_oval()`, `mask()`, `mirror()`, `pad()`, `rotate()`, `to_size()`
rotate

Examples

stimuli <- demo_stim()

# set width to proportion, height proportional
resize(stimuli, .5) |> draw_tem()

# set width and height separately by pixels
resize(stimuli, 400, 250) |> draw_tem()

rotate

Rotate templates and images

Description

Rotate templates and images

Usage

rotate(
  stimuli,
  degrees = 0,
  fill = wm_opts("fill"),
  keep_size = TRUE,
  origin = "image"
)

Arguments

stimuli list of stimuli
degrees degrees to rotate
fill background color, see color_conv()
keep_size whether to keep the original size or expand images to the new rotated size
origin The origin of the rotation. Options are: "image" will rotate around the image center.
            "tem" will rotate around the average of all template coordinates.
            A vector of 1 or more point indices (0-based) will rotate around their average position.

Value

list of stimuli with rotated tems and/or images

See Also

Stimulus manipulation functions align(), crop_tem(), crop(), greyscale(), horiz_eyes(),
image_func(), mask_oval(), mask(), mirror(), pad(), resize(), to_size()
### Examples

```r
stimuli <- demo_stim() |> resize(0.5)
rotate(stimuli, 45, fill = "dodgerblue")
rotate(stimuli, 45, fill = "dodgerblue", keep_size = FALSE)

# if images are not in the centre of the image,
# try setting the origin to tem or specific point(s)
offset <- stimuli[1] |> draw_tem() |> pad(0, 250, 0, 0, fill = "dodgerblue")
rotate(offset, 45, origin = "image", fill = "pink")
rotate(offset, 45, origin = "tem", fill = "pink")

# rotate around point 0 (left eye)
offset |> crop_tem() |> rep(8) |> rotate(seq(0, 325, 45), origin = 0, fill = "pink") |> animate(fps = 5)
```

---

The function `same_tems` checks if all templates are the same.

### Description

Check All Templates are the Same

### Usage

```r
same_tems(stimuli)
```

### Arguments

- **stimuli**: list of stimuli

### Value

logical

### See Also

Template functions: `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()`, `squash_tem()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`
social_media_size

Examples

```r
stim <- demo_stim()
stim2 <- subset_tem(stim, features("gmm"))

same_temstems(stim)

c(stim, stim2) |> same_temstems()
```

---

**social_media_size**

*Social Media Image Sizes*

**Description**

A convenience function for getting recommended dimensions for images on social media sites.

**Usage**

```r
social_media_size(platform = c("twitter", "instagram"), type = "default")
```

**Arguments**

- `platform`: currently only "twitter"
- `type`: which type of image

**Details**

Twitter:

- link: Image from a Tweet with shared link
- one: Tweet sharing a single image (default)
- two: Tweet sharing two images
- three_left: Tweet sharing three images, Left image
- three_tight: Tweet sharing three images, Right images
- four: Tweet sharing four images

Instagram:

- feed_large: (default)
- feed_small: stories_large: stories_small:

**Value**

named vector of width and height in pixels

**Examples**

```r
social_media_size("twitter", "link")
social_media_size("twitter", "one")
social_media_size("twitter", "two")
```
**Squash Template Points**

**Description**

Move template points that are outside the image boundaries (e.g., negative values or larger than image width or height) to the borders of the image.

**Usage**

```r
squash_tem(stimuli)
```

**Arguments**

- `stimuli` list of stimuli

**Value**

list of stimuli

**See Also**

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem(), same_tems()`, `subset_tem()`, `tem_def()`, `viz_tem_def()`

**Examples**

```r
nosquash <- demo_stim(1) |> crop(0.4, 0.5)
squash <- demo_stim(1) |> crop(0.4, 0.5) |> squash_tem()

# add padding and visualise templates
c(nosquash, squash) |> pad(50) |> draw_tem(pt.size = 5)
```
subset_tem  Subsetting template points

Description
Keep or delete specified template points. Points will be renumbered and line definitions will be updated. If all points in a line are deleted, the line will be removed. Point indexing is 0-based, so the first two points (usually the pupils) are 0 and 1.

Usage
subset_tem(stimuli, ..., keep = TRUE)

Arguments
- stimuli: list of stimuli
- ...: vectors of points to keep or delete
- keep: logical; whether to keep or delete the points

Value
stimlist with altered templates

See Also
Template functions auto_delin(), average_tem(), centroid(), change_lines(), delin(), draw_tem(), features(), get_point(), remove_tem(), require_tem(), same_tem(), squash_tem(), tem_def(), viz_tem_def()

Examples
# keep just the first two points
demo_stim(1) |> 
subset_tem(0:1) |> 
draw_tem(pt.size = 10)

# remove the last 10 points
# (produces the 179-point Perception Lab template)
demo_stim(1) |> 
subset_tem(179:188, keep = FALSE) |> 
draw_tem()

# use features() to keep only points from a pre-defined set
# "gmm" is points used for geometric morphometrics
demo_stim(1) |> 
subset_tem(features("gmm")) |> 
draw_tem()
symmetrize Symmetrize Images

Description
Use webmorph.org to make faces symmetric in shape and/or colour.

Usage
symmetrize(stimuli, shape = 1, color = 1, tem_id = "frl", ...)
symmetrise(stimuli, shape = 1, color = 1, tem_id = "frl", ...)

Arguments
- **stimuli**: list of stimuli
- **shape, color**: amount of symmetry (0 for none, 1.0 for perfect)
- **tem_id**: template ID to be passed to `tem_def()` (usually "frl" or "fpp106")
- **...**: Additional arguments to pass to `trans()`

Value
list of stimuli with symmetrised images and templates

See Also
WebMorph.org functions `avg()`, `continuum()`, `loop()`, `trans()`, `webmorph_up()`

Examples
```r
if (webmorph_up()) {
  stimuli <- demo_stim(1)

  sym_both <- symmetrize(stimuli)
  sym_shape <- symmetrize(stimuli, color = 0)
  sym_color <- symmetrize(stimuli, shape = 0)
  sym_anti <- symmetrize(stimuli, shape = -1.0, color = 0)
}
```
tems_to_array

Convert stimuli to array for geomorph

Description

Convert stimuli to array for geomorph

Usage

tems_to_array(stimuli)

Arguments

stimuli list of stimuli

Value

3D array

Examples

data <- demo_stim() |> tems_to_array() 
dim(data)

---

tem_def

Get template definition

Description

Template definitions are lists that contain information about templates that are needed to do things like symmetrising and masking images. This function is mostly used internally.

Usage

tem_def(tem_id = "frl", path = NULL)

Arguments

tem_id the name of a built-in template (frl, fpp106, fpp83, dlib70, or dlib7) or a numeric ID of a template to retrieve from webmorph.org

path path of local tem definition file
Details

If you have defined a custom template on webmorph.org, you can get its function definition by ID. You can see the ID numbers next to the templates available to you under the Template > Current Template menu.

Value

list with template definition

See Also

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()`, `same_tem()`, `squash_tem()`, `subset_tem()`, `viz_tem_def()`

Examples

```r
fpp106 <- tem_def("fpp106")
fpp106$lines |> str()

fpp83 <- tem_def("fpp83")
fpp83$mask |> str()

frl <- tem_def("frl")
frl$points[1:10, ]
viz_tem_def(frl, pt.size = 10, line.size = 5)
```

---

to_size

Resize and crop/pad images to a specified size

Description

Resize and crop/pad images to a specified size

Usage

```r
to_size(
  stimuli,
  width = NULL,
  height = NULL,
  fill = wm_opts("fill"),
  crop = FALSE,
  keep_rels = FALSE
)
```
trans

Transform Images

Description

Transform a base image in shape, color, and/or texture by the differences between two images.

Usage

trans(
  trans_img = NULL,
  from_img = NULL,
  to_img = NULL,
  shape = 0,
  color = 0,
  texture = 0,
  outname = NULL,
  norm = c("none", "twopoint", "rigid"),
  normpoint = 0:1,
)

Arguments

- stimuli: list of stimuli
- width: the target width (if null, the maximum stimulus width is used)
- height: the target height (if null, the maximum stimulus height is used)
- fill: background color if cropping goes outside the original image, see color_conv()
- crop: whether to crop or pad images to make them the specified size
- keep_rels: whether to keep the size relationships between images in the set, or make all the maximum size

Value

list of stimuli with cropped items and/or images

See Also

Stimulus manipulation functions align(), crop_tem(), crop(), greyscale(), horiz_eyes(), image_func(), mask_oval(), mask(), mirror(), pad(), resize(), rotate()

Examples

# images with different aspect ratios and sizes
stimuli <- demo_unstandard(c(1:4, 6:9))

to_size(stimuli, 200, 200, fill = "dodgerblue")
sample_contours = TRUE,
    warp = c("multiscale", "linear", "multiscalerb")
)

Arguments

trans_img      list of stimuli to transform
from_img       negative transform dimension endpoint (0% image)
to_img         positive transform dimension endpoint (100% image)
shape, color, texture
    amount to change along the vector defined by from_img and to_img (can range
    from -3 to +3)
outname        name to save each image as
norm           how to normalise the images; see Details
normpoint      points for twopoint normalisation
sample_contours
    whether to sample contours or just points
warp           warp type

Details

Normalisation options:

• none: averages will have all coordinates as the mathematical average of the coordinates in
  the component templates
• twopoint: all images are first aligned to the 2 alignment points designated in normpoint.
  Their position is set to their position in the first image in stimuli
• rigid: procrustes aligns all images to the position of the first image in stimuli

Sample contours:

This interpolates more control points along the lines. This can improve the accuracy of averages
and transforms. If you see a “feathery” appearance along lines that have many, close-together
points, try turning this off.

Warp types:

• multiscale: Implements multi-scale affine interpolation for image warping. This is the de-
  fault, with a good balance between speed and accuracy
• linear: Implements triangulated linear interpolation for image warping. Linear warping is
  least accurate, often resulting in image artifacts, but is very fast.
• multiscalerb: Implements multi-scale rigid body interpolation for image warping. This de-
  creases image artifacts in some circumstances, but is much slower.

Value

list of stimuli with transformed images and templates
See Also

WebMorph.org functions `avg()`, `continuum()`, `loop()`, `symmetrize()`, `webmorph_up()`

Examples

```r
if (webmorph_up()) {
  stimuli <- demo_stim()
  sexdim <- trans(stimuli, stimuli$f_multi, stimuli$m_multi,
                   shape = c(fem = -0.5, masc = 0.5))

  sexdim |> draw_tem() |> label()
}
```

**viz_tem_def**

Visualise a template definition

Description

Visualise a template definition

Usage

```r
viz_tem_def(tem_def, ...)
```

Arguments

- `tem_def` the template definition; usually from `tem_def()`
- `...` further arguments to pass to `draw_tem()`: pt.size and line.size often need to be adjusted

Value

a stimlist with a blank image and the template drawn on it

See Also

Template functions `auto_delin()`, `average_tem()`, `centroid()`, `change_lines()`, `delin()`, `draw_tem()`, `features()`, `get_point()`, `remove_tem()`, `require_tem()``, `same_tem()``, `squash_tem()`, `subset_tem()``, `tem_def()`

Examples

```r
dlib70 <- tem_def("dlib70")
viz_tem_def(dlib70, pt.size = 5, line.size = 3)
```
webmorph_up  
*Check if webmorph.org is available*

**Description**
Check if webmorph.org is available

**Usage**
webmorph_up()

**See Also**
WebMorph.org functions `avg()`, `continuum()`, `loop()`, `symmetrize()`, `trans()`

**Examples**
webmorph_up()

---

**width**  
*Image widths*

**Description**
Image widths

**Usage**

width(stimuli, type = c("all", "min", "max", "unique"))

**Arguments**

- **stimuli**: list of stimuli
- **type**: whether to return all widths, min, max, or only unique widths

**Value**
vector of widths

**See Also**
Other info: `add_info()`, `compare()`, `get_info()`, `get_point()`, `height()`, `metrics()`, `rename_stim()`

**Examples**

demo_stim() |> width()
wm_opts

Set/get global webmorph options

Description

See `wm_opts_defaults()` for explanations of the default options.

Usage

wm_opts(...)  

Arguments

...  

One of four: (1) nothing, then returns all options as a list; (2) a name of an option element, then returns its value; (3) a name-value pair which sets the corresponding option to the new value (and returns nothing), (4) a list with option-value pairs which sets all the corresponding arguments.

Value

a list of options, values of an option, or nothing

See Also

`wm_opts_defaults()`

Examples

wm_opts()  # see all options

wm_opts("verbose")  # see value of webmorph.verbose

## Not run:
# set value of webmorph.verbose
wm_opts(verbos = FALSE)

# set multiple options
opts <- list(fill = "black",
             pt.color = "white",
             line.color = "red")
wm_opts(opts)

## End(Not run)
**Description**

Options set on load (unless they were already set by `.Renviron`)

- **overwrite** ("ask"): Whether to overwrite images saved with `write_stim()` when in interactive mode; possible values are "ask" (ask if filenames exist), TRUE (always overwrite), and FALSE (never overwrite)
- **fill** ("white"): the colour to use to fill image backgrounds
- **pt.color** ("green"): the colour to use for points in `draw_tem()`
- **line.color** ("blue"): the colour to use for lines in `draw_tem()`
- **plot** ("inline"): whether to plot images inline in R markdown documents (set to any other value to just view them in the viewer)
- **plot.maxwidth** (2400): The maximum width of images created by `plot()`
- **plot.maxheight** (2400): The maximum height of images created by `plot()`
- **verbose** (TRUE): Whether to produce verbose output and progress bars for long functions like `auto_delin()`, `avg()` or `trans()`
- **server** ("https://webmorph.org"): The server to use for webmorph functions like `avg()` and `trans()`; do not change unless you've set up a local server
- **connection** (stdin()): use internally for testing interactive functions; do not change

**Usage**

```r
wm_opts_defaults()
```

**Value**

a list of default options

**See Also**

- `wm_opts()`

**Examples**

```r
wm_opts_defaults() |> str() # view defaults

## Not run:
# reset all options to default
wm_opts_defaults() |> wm_opts()

## End(Not run)
```
write_stim

Write images and templates to files

Description

Write images and templates to files

Usage

```r
write_stim(
  stimuli,
  dir = ".",
  names = NULL,
  format = "png",
  ..., 
  overwrite = wm_opts("overwrite")
)
```

Arguments

- `stimuli`: list of stimuli
- `dir`: Directory to save to
- `names`: A vector of stimulus names or NULL to use names from the stimuli list
- `format`: output format such as "png", "jpeg", "gif"; is overridden if names end in .png, .jpg, or .gif
- `...`: other arguments to pass to magick::image_write, such as quality (for jpegs)
- `overwrite`: whether to overwrite existing files (TRUE/FALSE) or "ask" (only in interactive mode)

Value

list of saved paths

See Also

Stimulus creating functions `animate()`, `as_stimlist()`, `blank()`, `new_stimlist()`, `new_stim()`, `read_img()`, `read_stim()`, `read_tem()`

Examples

```r
## Not run:
# write demo stim as jpegs to directory ./test_faces
demo_stim() |> 
  write_stim("test_faces", format = "jpg")

## End(Not run)
```
write_tps

Create a TPS file from a stimlist

Description

Create a TPS file from a stimlist

Usage

write_tps(stimuli, path_to_tps = NULL)

Arguments

stimuli list of stimuli
path_to_tps optional filename to save TPS file

Value
text of tps file

Examples

# set path_to_tps to save to a file
tps <- demo_stim() |> write_tps()
Index

* color
  * color_conv, 15

* info
  * add_info, 3
  * compare, 16
  * get_info, 26
  * get_point, 26
  * height, 29
  * metrics, 37
  * rename_stim, 46
  * width, 60

* manipulators
  * align, 4
  * crop, 19
  * crop_tem, 20
  * greyscale, 28
  * horiz_eyes, 30
  * image_func, 30
  * mask, 34
  * mask_oval, 36
  * mirror, 38
  * pad, 40
  * resize, 48
  * rotate, 49
  * to_size, 56

* stim
  * animate, 6
  * as_stimlist, 8
  * blank, 12
  * read_stim, 45
  * write_stim, 63

* tem
  * auto_delin, 9
  * average_tem, 10
  * centroid, 13
  * change_lines, 14
  * delin, 21
  * draw_tem, 23
  * features, 25
  * get_point, 26
  * remove_tem, 46
  * require_tem, 46
  * same_tem, 50
  * squash_tem, 52
  * subset_tem, 53
  * tem_def, 55
  * viz_tem_def, 59

* viz
  * as_ggplot, 7
  * draw_tem, 23
  * gglabel, 27
  * label, 32
  * mlabel, 39
  * plot_rows, 42
  * plot_stim, 43

* webmorph
  * avg, 10
  * continuum, 18
  * loop, 33
  * symmetrize, 54
  * trans, 57
  * webmorph_up, 60

add_info, 3, 17, 26, 27, 29, 37, 47, 60
align, 4, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57
animate, 6, 8, 12, 45, 63
as_ggplot, 7, 24, 28, 32, 40, 43, 44
as_stimlist, 6, 8, 12, 45, 63
auto_delin, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
auto_delin(), 62
average_tem, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
avg, 10, 18, 33, 54, 59, 60
avg(). 10, 62

blank, 6, 8, 12, 45, 63
bounds, 12
centroid, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
change_lines, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
col2lab, 15
col2lab(), 42
color_conv(), 4, 12, 19, 23, 30, 34, 36, 44, 49, 57
compare, 3, 16, 26, 27, 29, 37, 47, 60
continuum, 11, 18, 33, 54, 59, 60
crop, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57
crop(), 20, 40
crop_tem, 5, 19, 20, 29–31, 35, 36, 38, 41, 48, 49, 57
delin, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
demo_stim, 22
demo_stim(), 22
demo_tem, demo_stim, 22
demo_temr, 22
demo_unstandard(demo_stim), 22
demo_unstandard(), 22
draw_tem, 7, 9, 10, 13, 14, 21, 23, 25, 27, 28, 32, 40, 43, 44, 46, 48, 50, 52, 53, 56, 59
draw_tem(), 5, 59, 62
features, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
get_info, 3, 17, 26, 27, 29, 37, 47, 60
get_point, 3, 9, 10, 13, 14, 17, 21, 24–26, 26, 29, 37, 46–48, 50, 52, 53, 56, 59, 60
gglabel, 7, 24, 27, 32, 40, 43, 44
gglabel(), 32, 40
ggplot2::annotate, 27
ggplot2::annotate(), 27, 40
greyscale (greyscale), 28
grDevices::colors(), 15
greyscale, 5, 19, 21, 28, 30, 31, 35, 36, 38, 41, 48, 49, 57
height, 3, 17, 26, 27, 29, 37, 47, 60
horiz_eyes, 5, 19, 21, 29, 30, 31, 35, 36, 38, 41, 48, 49, 57
image_func, 5, 19, 21, 29, 30, 32, 35, 36, 38, 41, 48, 49, 57
image_func_types, 32
image_func_types(), 31
lab2rgb, 15
lab2rgb(), 42
label, 7, 24, 28, 32, 40, 43, 44
label(), 28
lapply(), 8
loop, 11, 18, 33, 54, 59, 60
magick::decoration_types(), 39
magick::image_anotate, 28, 39
magick::image_write, 63
magick::style_types(), 39
mask, 5, 19, 21, 29–31, 34, 36, 38, 41, 48, 49, 57
mask_oval, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57
metrics, 3, 17, 26, 27, 29, 37, 47, 60
mirror, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57
mlabel, 7, 24, 28, 32, 39, 43, 44
mlabel(), 32
new_stim, 6, 8, 12, 45, 63
new_stimlist, 6, 8, 12, 45, 63
pad, 5, 19, 21, 29–31, 35, 36, 38, 40, 48, 49, 57
patch, 41
plot(), 8, 43, 62
plot_stim, 7, 24, 28, 32, 40, 43, 44
plot_stimlist, 7, 24, 28, 32, 40, 43, 44
plot_rows, 7, 24, 28, 32, 40, 42, 44
plot_stim, 7, 24, 28, 32, 40, 43, 43
plot_stim(), 7
print(), 8
purrr::map(), 8
read_img, 6, 8, 12, 45, 63
read_stim, 6, 8, 12, 45, 63
read_tem, 6, 8, 12, 45, 63
remove_tem, 9, 10, 13, 14, 21, 24, 25, 27, 26, 48, 50, 52, 53, 56, 59
rename_stim, 3, 17, 26, 27, 29, 37, 46, 60
require_tem, 9, 10, 13, 14, 21, 24, 25, 27, 46, 47, 50, 52, 53, 56, 59
resize, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57
INDEX

rotate, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 57

same_tem, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
social_media_size, 51
squash_tem, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
subset_tem, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59
symmetrise (symmetrize), 54
symmetrize, 11, 18, 33, 54, 59, 60
tem_def, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 55, 59
tem_def(), 34, 54, 59
tems_to_array, 55
to_size, 5, 19, 21, 29–31, 35, 36, 38, 41, 48, 49, 56
trans, 11, 18, 33, 54, 57, 60
trans(), 18, 33, 54, 62

viz_tem_def, 9, 10, 13, 14, 21, 24, 25, 27, 46, 48, 50, 52, 53, 56, 59

webmorph_up, 11, 18, 33, 54, 59, 60
width, 3, 17, 26, 27, 29, 37, 47, 60
wm_opts, 61
wm_opts(), 62
wm_opts_defaults, 62
wm_opts_defaults(), 61
write_stim, 6, 8, 12, 45, 63
write_stim(), 62
write_tps, 64