Package 'lexsync'

September 25, 2026

Type Package

Title Lexical Optimisation and Hardware-Timed Experiment Generation

Version 0.1.1

Description A cross-platform toolkit that unifies many-language lexical-corpus access, parallel multidimensional stimulus matching, deterministic pseudoword generation, counterbalancing and the automated generation of experiments from a declarative trial-event model, for 'PsychoPy', 'OpenSesame' and the browser ('jsPsych'). The laboratory targets bind electroencephalography onset triggers to the stimulus flip. It is the R member of a dual-language pair; a structurally identical 'Python' package is also provided. Several paradigms (factorial word contrasts, lexical decision, priming, self-paced reading and cued categorisation) are supported, and each design is accompanied by a machine- and human-readable materials datasheet for reproducibility.

License MIT + file LICENSE

Copyright The MIT licence covers the source code only. The example lexica in inst/extdata are derived from third-party data and are distributed under CC BY-SA 4.0; the terms and the attribution they require are in the LICENSE.note file.

Encoding UTF-8

Language en-GB

Depends R (>= 4.0.0)

Imports readr, yaml, stringdist, stringi, jsonlite, digest, stats, tools, utils

Suggests testthat (>= 3.1.8), knitr, rmarkdown, spelling, clue, shiny, bslib, DT, zip

VignetteBuilder knitr

URL https://github.com/pablobernabeu/lexsync, https://pablobernabeu.github.io/lexsync/r/

BugReports https://github.com/pablobernabeu/lexsync/issues

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Pablo Bernabeu [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1083-2460>)

Maintainer Pablo Bernabeu <pcbernabeu@gmail.com>

Repository CRAN

Date/Publication 2026-09-25 09:30:02 UTC

Contents

add_bigram_frequency . . . 3
add_neighbourhood . . . 4
add_pair_overlap . . . 4
assign_triggers . . . 5
balance_check . . . 6
balance_lists . . . 6
build_datasheet . . . 7
build_lexdec_stimuli . . . 9
build_pool . . . 10
cohens_d . . . 10
cohens_d_ci . . . 11
counterbalance . . . 12
count_syllables . . . 12
describe_stimuli . . . 13
export_experiments . . . 13
export_jspsych . . . 14
export_opensesame . . . 15
export_psychopy . . . 15
fetch_corpus . . . 16
generate_pseudowords . . . 16
lexsync_cache_clear . . . 17
lexsync_cache_dir . . . 18
list_corpora . . . 19
load_items . . . 19
load_lexicon . . . 20
load_pool . . . 21
log_artefact . . . 22
log_step . . . 22
make_pseudoword . . . 23
match_report . . . 23
match_report_continuous . . . 24
match_stimuli . . . 25
merge_norms . . . 25
methods_paragraph . . . 26
new_run_log . . . 27
PARADIGMS . . . 27

participant_table 28
required_fields 28
resample_stimuli 29
resolve_events 29
resolve_trial_timing 30
run_all 30
run_pipeline 31
select_continuous_stimuli 32
tost_equiv 33
variance_ratio 34
write_datasheet 34
write_run_log 35

Index **36**

`add_bigram_frequency` *Mean bigram probability (type-based, non-positional), a phonotactic-probability proxy*

Description

For each word, the mean over its adjacent letter bigrams of the corpus bigram probability (count divided by the total bigram count). Computed from integer counts and rounded, so it is identical in the R and Python engines.

Usage

```
add_bigram_frequency(df, reference = NULL)
```

Arguments

`df`	A data frame with a `word` column.
`reference`	A character vector of reference words (defaults to `df$word`).

Value

`df` with an added numeric `bigram_freq` column.

add_neighbourhood *Compute orthographic-neighbourhood dimensions (Coltheart's N and OLD20)*

Description

`n_density` is Coltheart's N: the number of reference words of the same length differing by a single letter substitution (Hamming distance 1). `old20` is the mean Levenshtein distance to the 20 nearest reference words (Yarkoni et al., 2008). Both are computed against `reference`, which should be a large word list (typically the whole lexicon), not just the experimental pool.

Usage

```
add_neighbourhood(df, reference = df$word, n_old = 20L)
```

Arguments

`df`	A data frame with a `word` column.
`reference`	A character vector of reference words.
`n_old`	Neighbourhood size for OLD (default 20).

Value

`df` with added integer `n_density` and numeric `old20` columns.

add_pair_overlap *Orthographic overlap between the two members of each pair*

Description

Adds two columns. `pair.lev` is the Levenshtein distance between the pair's two orthographic forms, and `pair.overlap` is `1 - lev / max(nchar)`, the proportion of the longer form the two share. Overlap is the standard confound control in a priming design: a related pair that also shares letters confounds semantic relatedness with orthographic similarity.

Usage

```
add_pair_overlap(df, prime = "prime", target = "target")
```

Arguments

`df`	A pair table.
`prime`, `target`	Column names holding the two orthographic forms.

Details

Both engines return identical values, and the reasons are worth stating because they are the constraints on any future relational dimension. The core is an integer edit distance, and `stringdist(method = "lv")` and rapidfuzz's `Levenshtein.distance` agree exactly, including on decomposed Unicode and CJK, which is the same cross-library agreement `add_neighbourhood()` already stakes `old20` on. Length is counted in code points, `nchar()`'s default and Python's `len()`, never in bytes. The arithmetic uses only `-` and `/`, which IEEE-754 mandates be correctly rounded, and the result is rounded to nine decimal places, the constant used everywhere else in the package. A degenerate pair of two empty forms returns 0 rather than `0/0`, because a NaN would be sorted and compared and would then drop the row from one engine's control window but not the other's.

Value

`df` with `pair.lev` and `pair.overlap` added.

`assign_triggers` *Assign EEG trigger codes to stimuli*

Description

Adds `condition_trigger` (101, 102, ... per condition) and `item_trigger` (40-239 per item/`set`). Events reference these by the tokens "condition" and "item", or carry their own integer codes. The item range holds 200 codes (an 8-bit-port constraint), so past 200 sets the codes wrap and repeat, and a runtime notice says so.

Usage

```
assign_triggers(stimuli, conditions = NULL)
```

Arguments

`stimuli`	A stimuli data frame.
`conditions`	Optional character vector of condition labels, in the order that assigns their codes from 101 upwards. `NULL` (the default) numbers the conditions in order of first appearance.

Details

The condition codes follow `conditions` when it is given: its first entry is 101, its second 102, and so on, whether or not every entry occurs in `stimuli`, so the same condition keeps the same code in a subset such as a practice block. A condition present in `stimuli` but missing from `conditions` takes the next free codes, in code-point order of its label. Either way the mapping depends only on the labels, never on the row order. Without `conditions`, the codes follow the order in which the conditions first appear in `stimuli`, as in lexsync 0.1.0. On a shuffled trial list that order comes from the shuffle, so a different seed can swap two conditions' codes. `export_experiments()` passes the design's order, and the pipeline passes the order of the design's `conditions`, followed by any other condition in the order the item source lists it.

Value

`stimuli` with trigger columns added.

Examples

```
stim <- data.frame(condition = c("low", "high", "low", "high"), set = c(1, 1, 2, 2))
# First appearance: low = 101, high = 102.
unique(assign_triggers(stim)[, c("condition", "condition_trigger")])
# The design's order: high = 101, low = 102, however the rows are shuffled.
unique(assign_triggers(stim, conditions = c("high", "low"))[
  , c("condition", "condition_trigger")])
```

---

balance_check *Check that the levels of given columns occur equally often*

Description

Check that the levels of given columns occur equally often

Usage

```
balance_check(stimuli, columns)
```

Arguments

`stimuli`	A stimuli data frame.
`columns`	Columns whose level counts should be equal.

Value

A character vector of human-readable balance warnings (empty if none).

---

balance_lists *Assign item sets to counterbalancing lists so the lists match on the item dimensions*

Description

The factorial recipe's default deal is by set rank, which balances nothing. This searches instead for an assignment whose lists have near-equal totals on each declared dimension, by steepest-descent pairwise swaps between lists. List sizes are preserved, because a swap exchanges one set for another.

Usage

```
balance_lists(stimuli, design, schema)
```

Arguments

stimuli	A stimuli data frame with a set column and the balance dimensions.
design	A parsed design configuration. Reads counterbalance.lists, counterbalance.balance_on (defaulting to match_on, then to the continuous predictor and controls) and counterbalance.max_passes.
schema	The parsed global schema (provides the seed).

Details

The search is deterministic and identical in the R and Python engines: the objective is all-integer (see the notes in this file), the descent takes the single best swap each pass, and ties are broken by the seeded keyed hash rather than by position, so no list is favoured by being numbered first. Because the cost is a non-negative integer that strictly decreases, the search terminates; max_passes bounds it anyway and the report says whether the bound was reached.

Five situations are refused rather than answered with an assignment that would mislead. A Latin-square design is refused because every item already appears in every list there, so there is nothing left to equate, and fewer than two lists leaves no pair of lists to exchange sets between. A design with no resolvable balance dimension is refused, as is one naming a dimension the stimuli do not carry, and the message names the columns. The last refusal is arithmetic: the search stops if the integer objective would leave the range a double represents exactly, since past that point the two engines could disagree.

Value

A list with list_of_set (a named integer vector mapping each set to a list) and report (the dimensions, the integer cost before and after, the number of swaps taken, and whether the pass bound was reached).

---

build_datasheet *Assemble the materials datasheet for one design*

Description

Assemble the materials datasheet for one design

Usage

```
build_datasheet(
  design,
  schema,
  report,
  stimuli,
  source_path,
  artifacts,
  seed,
  engine = "R",
```

```
  candidate_pool = NULL,
  norms = NULL,
  balance = NULL,
  blocks = NULL,
  design_path = NULL,
  schema_path = NULL,
  selection_audit = NULL,
  neighbourhood_reference = NULL
)
```

Arguments

design	A parsed design list.
schema	The parsed global schema (schema.yaml).
report	Match report data frame, from match_report() or match_report_continuous(), or NULL for generated or tabled items.
stimuli	The selected stimulus data frame.
source_path	Path of the lexicon or item table the stimuli came from.
artifacts	Named list of the artifact paths written for the design (stimuli, descriptives, comparisons, experiments).
seed	The integer seed recorded for the counterbalanced trial order.
engine	Engine label recorded in the record (default "R").
candidate_pool	Optional list of per-condition candidate-pool sizes (list(condition, n_candidates)) recording how many items satisfied each condition's window before matching; reported for selection transparency.
norms	Optional list of norm-table provenance records, from the design's norms: block (see the pipeline). Each names a file, its sha256, the join key and the per-column coverage. Recorded because a norm table can supply the very variable a design manipulates, so a record that omitted it would describe a selection over columns of unstated origin.
balance	Optional balance-optimiser report, from balance_lists(). Recorded because it decides which items each participant sees.
blocks	Optional practice/filler block report. Recorded because those trials are presented but not analysed, so the presented and analysed counts differ and the record must say why.
design_path, schema_path	Optional paths of the design and schema files the run read; when given, their sha256 checksums complete the reproducibility record, because those two files decide everything the seed does not.
selection_audit	Optional matcher audit record; its window_relaxations entries are recorded because a relaxed window changes what "matched" means for that condition.
neighbourhood_reference	Optional record of the lexicon the neighbourhood dimensions were computed against (list(source, n_words, sha256)), recorded verbatim.

Value

The datasheet as a nested list, ready for `write_datasheet()`.

`build_lexdec_stimuli` *Assemble a word-vs-pseudoword lexical-decision set from a candidate pool*

Description

Real words are drawn by an even spread across the byte-ordered pool, then a length-matched pseudoword is generated for each. The pool is first filtered to lower-case a-z forms, the only ones the pseudoword generators are defined for, so the eligible pool can be smaller than the request; the pipeline's shortfall policy then decides whether that errors. `reference_words` (the full lexicon) supplies the bigram statistics and the real-word list a pseudoword must avoid. The presented string is the `target` column; conditions are `word` and `pseudoword` and `set` pairs them.

Usage

```
build_lexdec_stimuli(
  pool,
  n,
  reference_words = NULL,
  method = "letter_substitution"
)
```

Arguments

`pool`	Data frame of candidate words, e.g. from `build_pool()`.
`n`	Number of real words to select.
`reference_words`	Character vector supplying the bigram statistics and the real word forms a pseudoword must avoid; defaults to the pool's words.
`method`	Pseudoword generator: `"letter_substitution"` (default) or `"subsyllabic"` (Wuggy-style; Keuleers & Brysbaert, 2010).

Value

A stimulus data frame with `target`, `condition` and `set` columns.

build_pool *Build an experimental candidate pool by filtering a lexicon*

Description

A filter naming a column the frame does not have is silently skipped, because the same function filters lexica, supplied pools and pair tables, and those carry different columns. The cost is that a misspelt key silently widens a selection, so every caller that takes its filters from a design checks the names against the frame first: `run_pipeline()` for `pool_filters`, `match_stimuli()` for a condition's `define_by`, and the pair selector for both.

Usage

```
build_pool(lexicon, filters = NULL)
```

Arguments

`lexicon`	A lexicon data frame.
`filters`	A named list mapping columns to either a numeric `c(min, max)` range or a vector of permitted values.

Value

The filtered lexicon, with row names dropped. A row missing the filtered column is dropped under either kind of filter, and a range with a reversed or non-finite bound is an error rather than an empty pool.

Examples

```
schema <- yaml::read_yaml(system.file("extdata", "schema.yaml", package = "lexsync"))
lex <- load_lexicon(system.file("extdata", "en_example.csv", package = "lexsync"),
                    schema)
nrow(build_pool(lex, list(length = c(4, 6), frequency = c(4, 6))))
```

cohens_d *Cohen's d (pooled-SD standardised mean difference)*

Description

Cohen's d (pooled-SD standardised mean difference)

Usage

```
cohens_d(x, y)
```

Arguments

x, y	Numeric vectors.

Value

The standardised mean difference; 0 when either sample is too small or both share one constant, NA when the pooled SD is zero but the means differ (the standardised difference is then unbounded, not zero).

Examples

```
cohens_d(c(5, 6, 7, 8), c(5, 6, 7, 9))
```

---

cohens_d_ci *Cohen's d with a confidence interval, complementing the TOST verdict*

Description

The interval is the (1 - 2 * alpha) confidence interval for the standardised mean difference, so alpha = 0.05 gives the 90% interval that goes with a TOST decision at the .05 level (Lakens, 2017). Reporting the interval, rather than only a binary verdict, makes the realised imbalance and its sampling uncertainty explicit, and keeps the dependence on the number of items visible. With few items the interval is wide, so a small point estimate cannot be over-read as evidence of a small true difference (Sassenhagen & Alday, 2016).

Usage

```
cohens_d_ci(x, y, alpha = 0.05)
```

Arguments

x, y	Numeric vectors.
alpha	Significance level matching the TOST (default 0.05).

Details

The limits are d +/- t * SE(d), where t is the Student t quantile on nx + ny - 2 degrees of freedom and SE(d) is the large-sample standard error of d, sqrt(1/nx + 1/ny + d^2 / (2 * (nx + ny))) (Hedges & Olkin, 1985; Borenstein et al., 2009, eq. 4.20). The d^2 term carries the sampling error of the pooled standard deviation. It is negligible for a well-matched control, where d is near zero, but it dominates for a manipulated dimension and can more than double the width of that interval. TOST tests the raw mean difference, so the interval is slightly wider than the one its decision implies, and when a control's limit lies close to the bound, tost_p gives the verdict.

Value

A list with d, ci_low and ci_high.

counterbalance *Assign stimuli to lists and a randomised, reproducible trial order*

Description

Dispatches on the design's paradigm: the factorial recipe for matched word lists, or a Latin square over conditions for paired/sentence paradigms.

Usage

```
counterbalance(stimuli, design, schema, list_of_set = NULL)
```

Arguments

stimuli	A stimuli data frame (matched set or loaded item table).
design	A parsed design configuration.
schema	The parsed global schema (provides the seed).
list_of_set	Optional named integer vector mapping each set to a list, from balance_lists(). Supplied by the pipeline when counterbalance.optimise is on; when NULL the factorial recipe deals sets to lists by rank as before.

Value

stimuli with added list and trial columns.

count_syllables *Orthographic syllable estimate: the number of maximal vowel runs*

Description

Orthographic syllable estimate: the number of maximal vowel runs

Usage

```
count_syllables(word)
```

Arguments

word	Character vector of word forms.

Value

Integer vector: the estimated syllable count of each word.

Examples

```
count_syllables(c("cat", "table", "beautiful"))
```

`describe_stimuli` *Per-group descriptive statistics for several dimensions*

Description

Per-group descriptive statistics for several dimensions

Usage

```
describe_stimuli(stimuli, dims, by = "condition")
```

Arguments

`stimuli`	A stimuli data frame.
`dims`	Character vector of dimension columns.
`by`	Grouping column (default `"condition"`).

Value

A long data frame with n, mean, sd, min, median and max per group.

`export_experiments` *Export all presentation targets (PsychoPy, OpenSesame, jsPsych)*

Description

Assigns the EEG trigger codes with `assign_triggers()` and then writes each target. The condition codes follow `conditions`, which defaults to the order of the design's `conditions` entries, so in a design that declares its conditions the first is 101, the second 102, and so on, whatever order the counterbalanced trials put them in. A design with no `conditions` block (a generated lexical decision, an item table) falls back to the order of first appearance, unless `conditions` is given.

Usage

```
export_experiments(
  stimuli,
  design,
  schema,
  outdir,
  base = NULL,
  conditions = NULL
)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema (trigger and presentation settings).
outdir	Output directory.
base	Optional file-name stem.
conditions	Optional character vector of condition labels in the order that assigns their trigger codes (see assign_triggers()). NULL (the default) takes the order of the design's conditions.

Value

A named list of generated file paths.

---

export_jspsych *Export a browser-runnable jsPsych experiment*

Description

The rendered events and the trial data are embedded in one HTML file, so anyone can reproduce the procedure online from the same materials. The jsPsych library and stylesheet are loaded from a CDN, so the machine running the file needs an internet connection; the trial data are embedded and the responses are saved locally, so no server is required either to run it or to collect them. Onset triggers are recorded in each trial's data (a browser cannot drive a parallel port).

Usage

```
export_jspsych(stimuli, design, schema, outdir, base = NULL)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema (trigger and presentation settings).
outdir	Output directory.
base	Optional file-name stem.

Value

The path to the generated .html, invisibly.

export_opensesame *Export a complete plain-text OpenSesame experiment*

Description

Export a complete plain-text OpenSesame experiment

Usage

```
export_opensesame(stimuli, design, schema, outdir, base = NULL)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema (trigger and presentation settings).
outdir	Output directory.
base	Optional file-name stem.

Value

The path to the generated .osexp, invisibly.

export_psychopy *Export a runnable PsychoPy script that interprets the event sequence*

Description

Export a runnable PsychoPy script that interprets the event sequence

Usage

```
export_psychopy(stimuli, design, schema, outdir, base = NULL)
```

Arguments

stimuli	Stimuli with trigger columns (see assign_triggers()).
design	A parsed design configuration.
schema	The parsed global schema (trigger and presentation settings).
outdir	Output directory.
base	Optional file-name stem.

Value

The path to the generated .py, invisibly.

`fetch_corpus` *Download a CSV-format registered corpus into the cache*

Description

Suitable for Connector A corpora that expose a delimited file. The URL's scheme is checked first; the transfer then lands in a sidecar file that is renamed into place only after the size cap, the markup sniff and any `sha256` the registry entry carries have all passed. The download is recorded so it can be cited; consult `list_corpora()` for the citation.

Usage

```
fetch_corpus(name, registry_path = NULL, dest = NULL)
```

Arguments

`name`	A corpus name present in the registry.
`registry_path`	Optional path to `registry.yaml`.
`dest`	Optional destination file; defaults to `<name>.csv` in `lexsync_cache_dir()`.

Details

The file lands in `lexsync_cache_dir()` unless `dest` names somewhere else. The cache directory is created only once the registry entry and its URL have been accepted, so a refused call writes nothing. A download to the default destination begins by deleting any `.part` sidecar in the cache that an interrupted transfer left there more than a day earlier. The cache persists between sessions, and one corpus may reach the 200 MB download cap, so several of them add up. `lexsync_cache_clear()` removes one corpus or the whole cache, and the next call downloads the corpus again. A `dest` the caller names, inside the cache or not, is used as given. Its directory must already exist, no sidecar is swept, and the only file beside it that lexsync writes or deletes is the `<dest>.part` sidecar of the download itself.

Value

The path to the downloaded file, invisibly.

`generate_pseudowords` *A length-matched pseudoword for each base word (byte-order processing)*

Description

A length-matched pseudoword for each base word (byte-order processing)

Usage

```
generate_pseudowords(base_words, reference_words)
```

Arguments

base_words	Character vector of words to derive pseudowords from.
reference_words	Character vector (typically the full lexicon) that supplies the bigram statistics and the real word forms to avoid.

Value

A data frame with columns base_word and pseudoword.

---

lexsync_cache_clear *Remove fetched corpora from the cache*

Description

Deletes what fetch_corpus() put in lexsync_cache_dir(). Given a name, that is the corpus's <name>.csv and any <name>.csv.part that an interrupted download left behind. With name left NULL it is the whole cache directory. A corpus needed again is downloaded afresh by the next fetch_corpus() call. A file that fetch_corpus() wrote to a dest of the caller's choosing is not in the cache, and is never touched.

Usage

```
lexsync_cache_clear(name = NULL)
```

Arguments

name	A corpus name as given to fetch_corpus(), or NULL (the default) to remove the whole cache.

Value

The paths of the files removed, invisibly. The vector is empty when there was nothing to remove.

See Also

lexsync_cache_dir() for where the cache lives.

Examples

```
# A throwaway cache stands in for yours, so running this leaves your own alone.
local({
  tmp <- tempfile("lexsync-")
  old <- Sys.getenv("R_USER_CACHE_DIR", unset = NA)
  on.exit({
    if (is.na(old)) Sys.unsetenv("R_USER_CACHE_DIR") else
      Sys.setenv(R_USER_CACHE_DIR = old)
    unlink(tmp, recursive = TRUE)
  })
  Sys.setenv(R_USER_CACHE_DIR = tmp)
  # Two corpora and an interrupted download
  dir.create(lexsync_cache_dir(), recursive = TRUE)
  invisible(file.create(file.path(lexsync_cache_dir(),
    c("subtlex_nl.csv", "subtlex_nl.csv.part", "lexique_fr.csv"))))
  # One corpus, with its partial download
  print(basename(lexsync_cache_clear("subtlex_nl")))
  # Everything else, and the directory itself
  print(basename(lexsync_cache_clear()))
  dir.exists(lexsync_cache_dir())
})
```

`lexsync_cache_dir` *Per-user cache directory for fetched corpora*

Description

The directory `tools::R_user_dir("lexsync", "cache")` names for this package. It is where `fetch_corpus()` puts a download unless told otherwise, and it is the only place the package writes to without being handed a path. This function only reports the path. The directory is created by the first download into it, so until then it does not exist.

Usage

```
lexsync_cache_dir()
```

Details

The cache persists between sessions. A registered corpus is a delimited word list, and a download is refused above 200 MB, so a cache holding several large corpora can reach a few hundred megabytes. Each download into the cache first deletes any partial download that an interrupted transfer left there more than a day earlier, and fetching a corpus again replaces the earlier copy. Downloaded corpora otherwise stay until `lexsync_cache_clear()` removes them, one corpus at a time or all together. Nothing kept there is irreplaceable, so the next call downloads afresh.

Value

The cache directory path, which need not exist yet.

See Also

`lexsync_cache_clear()` to empty the cache.

Examples

```
lexsync_cache_dir()
```

`list_corpora` *List the corpora known to the registry*

Description

List the corpora known to the registry

Usage

```
list_corpora(registry_path = NULL)
```

Arguments

`registry_path`	Optional path to `registry.yaml`.

Value

A data frame describing each registered corpus.

`load_items` *Load a paradigm item table (prime-target pairs, sentences, ...)*

Description

The table must carry an `item` identifier, a `condition` label and the paradigm's presented fields. Field values are validated (no control characters; bounded length) so a crafted item cannot corrupt the generated loop table or scripts. Items are mapped to a deterministic integer `set` id (byte order), so counterbalancing matches the corpus path and the two engines.

Usage

```
load_items(path, required_fields)
```

Arguments

`path`	Path to a UTF-8 CSV item table.
`required_fields`	Character vector of presented fields the paradigm needs.

Value

A data frame with `set`, `condition` and the item fields.

`load_lexicon` *Load a lexicon from a CSV file*

Description

Reads a derived lexicon, validates the column contract, lower-cases the orthographic form, removes duplicates and attaches a stable integer `id` plus the inexpensive dimensions `length` and `frequency`. The orthographic neighbourhood dimensions are added later, on the experimental pool, by `add_neighbourhood()`, because they are quadratic in the size of the reference set.

Usage

```
load_lexicon(path, schema, language = NULL)
```

Arguments

`path`	Path to a derived lexicon CSV.
`schema`	The parsed schema (see `config/schema.yaml`).
`language`	Optional language label to record in a `language` column.

Value

A data frame with at least `word`, `length`, `n_syllables`, `frequency` and `id`, plus the frequency column the schema names (by default `freq_zipf`) and every other column the file carried. Rows are in byte order of `word` and `id` numbers them from 1.

Examples

```
# Both inputs are bundled with the package, so this runs offline and touches
# nothing outside the installation.
schema <- yaml::read_yaml(system.file("extdata", "schema.yaml", package = "lexsync"))
lex <- load_lexicon(system.file("extdata", "en_example.csv", package = "lexsync"),
                    schema)
head(lex[, c("word", "frequency", "length", "n_syllables")])
```

load_pool *Load a supplied candidate pool of words and give it the matcher's dimensions*

Description

A researcher who already has a curated word list (from a previous study, a norming session, a colleague) should not have to dress it up as a corpus lexicon to get lexsync's matching, validation and datasheet. This reads such a list and returns something the matcher accepts.

Usage

```
load_pool(path, schema, lexicon = NULL, language = NULL)
```

Arguments

path	Path to a UTF-8 CSV with at least a word column.
schema	The parsed schema (used when a lexicon is loaded).
lexicon	Optional path to a derived lexicon to draw dimensions from.
language	Optional language label recorded in a language column.

Details

The list needs only a word column. Length and the syllable estimate are derived from the form. Everything else is either supplied on the list itself or looked up: with lexicon given, the corpus dimensions (frequency above all) are joined for those words, and a word the lexicon does not have is a hard error rather than an NA, because the tolerance windows drop NA rows silently and the pool would then be smaller than the user believes it is.

The returned reference matters as much as the pool. n_density and old20 are properties of a word in its *language*, not among the handful of words a study happens to use, so computing them against a 200-word supplied list would give numbers that mean nothing. When a lexicon is given, the reference is the lexicon's words; only without one does it fall back to the pool itself.

Value

A list with pool (the data frame, carrying word, id, length, n_syllables and any joined or supplied dimensions) and reference (the word vector the neighbourhood dimensions should be computed against).

log_artefact *Record a written artefact (path, rows, fingerprint) in the log*

Description

Record a written artefact (path, rows, fingerprint) in the log

Usage

```
log_artefact(log, path, rows = NA_integer_)
```

Arguments

log	A run-log object.
path	A file path that has just been written.
rows	Optional row count.

Value

The updated run-log object.

log_step *Append a step to a run log*

Description

Append a step to a run log

Usage

```
log_step(log, message, data = NULL)
```

Arguments

log	A run-log object.
message	A short description of the step.
data	An optional named list of step details.

Value

The updated run-log object.

make_pseudoword *The most bigram-plausible legal non-word at the smallest edit distance*

Description

Searches single-letter substitutions first, then two-letter substitutions; candidates are ranked by summed bigram frequency with a byte-order tie-break, so the choice is deterministic and identical across engines.

Usage

```
make_pseudoword(word, bigrams, lexset, usedset)
```

Arguments

word	The base word to derive the pseudoword from.
bigrams	Named integer vector of bigram counts, from bigram_counts().
lexset	Environment used as a set of the real word forms a pseudoword must avoid.
usedset	Environment used as a set of the pseudowords already taken.

Value

A single pseudoword string, or NULL if no legal candidate exists.

match_report *Build the full match-quality report*

Description

Build the full match-quality report

Usage

```
match_report(stimuli, dims, schema)
```

Arguments

stimuli	A matched-stimuli data frame (must contain condition).
dims	Dimensions to summarise and compare.
schema	The parsed global schema (equivalence settings).

Value

A list with `descriptives` and `comparisons` data frames. Every comparison is against the first condition in order of appearance, so a design with a single condition has nothing to compare and `comparisons` comes back with its columns and no rows.

Each row of `comparisons` names that first condition in `reference` and the condition compared with it in `condition`. The signed statistics run from the reference to the other condition: `cohens_d`, `d_ci_low` and `d_ci_high` are the reference mean minus the `condition` mean, in pooled standard deviations, so a positive d means the reference scores higher. `var_ratio` is the other way up, the `condition` variance over the reference variance.

`match_report_continuous` *Realised-control report for a continuous design*

Description

Returns the same list shape as `match_report()` (`descriptives` + `comparisons`), but the comparisons describe a continuous predictor: its realised span and, for each control, the Pearson correlation with the predictor (near zero when the control is held constant). Mirrors match_report_continuous in validation.py.

Usage

```
match_report_continuous(stimuli, predictor, controls, schema)
```

Arguments

`stimuli`	A stimuli data frame (a single "continuous" group).
`predictor`	The spanned predictor dimension.
`controls`	Character vector of control dimensions.
`schema`	The parsed global schema.

Value

A list with `descriptives` and `comparisons` data frames.

match_stimuli *Match stimuli across conditions on several lexical dimensions*

Description

The first condition is the anchor; its items are chosen by an even spread across the sorted candidate subpool. Every other condition is then matched to the anchor item by item, on the `match_on` dimensions, using standardised Euclidean distance under a tolerance window derived from the anchor.

Usage

```
match_stimuli(pool, design, schema, verbose = FALSE)
```

Arguments

`pool`	A lexicon/pool with all `match_on` dimensions present (see `add_neighbourhood()`).
`design`	A parsed design configuration (conditions, `match_on`, `n_per_condition`/`n_per_cell`).
`schema`	The parsed global schema (tolerances live here).
`verbose`	Logical; report tolerance relaxations and a shrunk anchor.

Details

Two policies govern degraded selections, each read from the design's `matching` block with the schema as fallback: `shortfall` (`"error"`, the default, refuses to return fewer sets than requested; `"allow"` accepts the shrink) and `on_insufficient_tolerance` (`"relax"`, the default, widens an undersupplied tolerance window to the full condition subpool and records the relaxation in an `"audit"` attribute; `"error"` refuses instead).

Value

A data frame of selected stimuli with a `condition` label and a `set` index pairing matched items across conditions.

merge_norms *Left-join a norm table (e.g. concreteness, age of acquisition, valence)*

Description

The connector for semantic dimensions: the norm data themselves are fetched separately (licensing varies), then merged here so the matcher can equate on them. The join is deterministic and identical across engines.

Usage

```
merge_norms(lexicon, norms, on = "word", columns = NULL)
```

Arguments

lexicon	A lexicon data frame.
norms	A data frame or the path to a CSV with a word column and norms.
on	The join column (default "word").
columns	Optional norm columns to keep.

Details

The result is the lexicon itself with the norm columns appended, and the key is looked up positionally rather than through `merge()`. That is what makes the two engines agree by construction, with nothing to repair afterwards, because `merge()` and `pandas.merge` were measured to diverge in three ways, each of them silent: R hoists the by column to position 1 while pandas keeps the left frame's order, so the column order differed whenever on was not already first; R disambiguates a colliding column name with `.x/.y` and pandas with `_x/_y`, and either way a dimension the design matches on disappears under a name nothing looks for; and `merge(sort = FALSE)` leaves the row order unspecified, so x's order is not carried through. A positional lookup has none of those degrees of freedom: the output is the input plus columns, in both engines. A colliding name is now an error instead.

The key is trimmed and case-folded on *both* sides. Only the norm table's side was normalised before, so a lexicon holding `Dog` matched nothing and the design carried on with an all-`NA` dimension. Because both engines agreed on that wrong answer, no parity test could have caught it. The lexicon's own spelling is preserved rather than folded in place: `word` is the byte-order tie-break behind every selection, so the join must not rewrite it.

Value

`lexicon` with the norm columns appended, in the lexicon's own row and column order. Rows with no matching norm get `NA`.

`methods_paragraph` *A ready-to-adapt methods paragraph rendered from a datasheet*

Description

A ready-to-adapt methods paragraph rendered from a datasheet

Usage

```
methods_paragraph(ds)
```

Arguments

ds	A datasheet list, from `build_datasheet()`.

Value

A single character string describing the materials procedure.

new_run_log *Start a new run log*

Description

Start a new run log

Usage

```
new_run_log(name, meta = list())
```

Arguments

name	A label for the run.
meta	A named list of run-level metadata (seed, versions, ...).

Value

A run-log object (a list).

PARADIGMS *The paradigm registry: default event sequences and required fields*

Description

Each event is a list with type (fixation | text | mask | blank | region_by_region | response | question | feedback), content (a literal or a {field} reference), an optional trigger (an integer EEG code or the token "condition"/"item"), onset_locked, response keys/timeout_ms, and an optional blocks restricting the event to named blocks.

Usage

```
PARADIGMS
```

Format

A named list with one entry per paradigm (factorial, lexical_decision, priming, categorisation, self_paced_reading), each holding stimulus_fields, a counterbalance recipe and an events list.

Value

A plain named list (class "list"; no package-specific class) of built-in paradigm specifications. Each element contains stimulus_fields, a character vector naming required item-table fields; counterbalance, a character scalar naming the counterbalancing recipe; and events, an ordered list of event-specification lists. The selected entry supplies the default trial sequence, required item fields, and counterbalancing rule inherited by a design that names that paradigm.

participant_table *Build a participant counterbalancing table*

Description

Crosses the supplied counterbalancing factors and replicates the cells to cover `n_participants`, generalising the `expand.grid()` + replication pattern of the original workflow's `participant_parameters.R`.

Usage

```
participant_table(factors, n_participants)
```

Arguments

`factors`	A named list of factors, each a vector of levels.
`n_participants`	Number of participants to allocate.

Value

A data frame with one row per participant.

Examples

```
participant_table(list(list = 1:2, order = c("a", "b")), 6)
```

required_fields *Trial fields a design needs present in its items (paradigm + events)*

Description

Trial fields a design needs present in its items (paradigm + events)

Usage

```
required_fields(design)
```

Arguments

`design`	A parsed design list.

Value

Character vector of the item fields the design's trials reference.

Examples

```
required_fields(list(paradigm = "categorisation"))
```

resample_stimuli *Produce several disjoint matched item sets (items as a random factor)*

Description

Each replicate is an independent, fully matched set drawn from the pool with the items of earlier replicates removed, so no item is reused. This lets a study treat its items as a random factor (running different item samples across participant groups, or showing an effect holds across samples) instead of treating them as a fixed set (Clark, 1973; Yarkoni, 2022). Deterministic and identical to the Python engine.

Usage

```
resample_stimuli(pool, design, schema, n_sets, verbose = FALSE)
```

Arguments

pool	A candidate pool with the match_on dimensions present.
design	A parsed design configuration.
schema	The parsed global schema.
n_sets	Number of disjoint matched sets to draw.
verbose	Logical; passed to match_stimuli().

Value

A data frame of matched stimuli with an added replicate column. The replicates are bound together, which drops the "audit" attribute match_stimuli() uses to report a relaxed tolerance window, so a relaxation inside a replicate reaches the console under verbose but not the run log or the datasheet.

resolve_events *The design's trial event list: its own* events*, else its paradigm's*

Description

The design's trial event list: its own events, else its paradigm's

Usage

```
resolve_events(design)
```

Arguments

design	A parsed design list.

Value

The list of trial events the design presents.

Examples

```
vapply(resolve_events(list(paradigm = "lexical_decision")),
       function(e) e$type, character(1))
```

`resolve_trial_timing` *Realise per-trial event durations onto the stimuli table*

Description

An event may declare a duration that varies from trial to trial, either read from an item column or drawn from a range. A drawn value is a pure function of the keyed hash, so both engines realise the same milliseconds, and it is written into the stimuli table as well as the generated script, because timing that varies is a variable the analysis needs, not presentation detail.

Usage

```
resolve_trial_timing(stimuli, design, schema)
```

Arguments

`stimuli`	A counterbalanced stimuli data frame.
`design`	A parsed design configuration.
`schema`	The parsed global schema (provides the seed).

Value

`stimuli` with one integer column per jittered event.

`run_all` *Run the lexsync pipeline for every design configuration*

Description

Run the lexsync pipeline for every design configuration

Usage

```
run_all(
  config_dir = "config",
  schema_path = file.path(config_dir, "schema.yaml"),
  outdir = NULL,
  verbose = TRUE
)
```

Arguments

config_dir	Directory of design_*.yaml configurations.
schema_path	Path to the global schema.
outdir	Output directory supplied by the caller. It must be supplied; lexsync does not choose a default output location.
verbose	Logical; print progress.

Value

A named list of per-design results, invisibly.

---

run_pipeline *Run the lexsync pipeline for one design*

Description

Run the lexsync pipeline for one design

Usage

```
run_pipeline(
  design_path,
  schema_path = "config/schema.yaml",
  outdir = NULL,
  reference_words = NULL,
  verbose = TRUE
)
```

Arguments

design_path	Path to a design configuration (YAML).
schema_path	Path to the global schema (YAML).
outdir	Output directory supplied by the caller (subdirectories stimuli, reports, experiments are created). It must be supplied; lexsync does not choose a default output location.
reference_words	Optional reference word list for neighbourhood computation; defaults to the whole lexicon.
verbose	Logical; print progress.

Value

A named list of output paths, invisibly.

`select_continuous_stimuli` *Select a set spanning a continuous predictor, holding controls constant*

Description

Instead of dichotomising the predictor into conditions and matching, items are chosen to cover the predictor's range evenly while the control dimensions are held within a tolerance band, so they stay near-constant and near-uncorrelated with the predictor. The set is analysed by regression / mixed models rather than between-condition contrasts (Kuperman, 2015; Liben-Nowell et al., 2019). Two deterministic even-spread passes make the R and Python engines select byte-identical stimuli. Mirrors select_continuous_stimuli in matching.py.

Usage

```
select_continuous_stimuli(
  pool,
  design,
  schema,
  verbose = FALSE,
  key = "word",
  label = "continuous",
  renumber_sets = TRUE
)
```

Arguments

`pool`	A candidate pool with the predictor and control dimensions present.
`design`	A parsed design configuration carrying a `continuous` block.
`schema`	The parsed global schema (tolerance windows).
`verbose`	Logical; report a window relaxation.
`key`	Column used as the selection unit and the byte-order tie-break, by default `"word"`. The pair-keyed path passes `"set"`: after a pair table is collapsed to one row per item set there is no `word` column, and `set` is unique per row, integer, and already derived deterministically.
`label`	Value written into the result's `condition` column, or `NULL` to leave the existing conditions alone. The pair path passes `NULL`, because its rows already carry the design's own conditions and overwriting them would destroy the contrast the design exists to test.
`renumber_sets`	Logical; renumber the selected rows `1..n`. The pair path passes `FALSE`, because its `set` ids have to survive selection for the result to be re-expanded back to the full pair table.

Details

The design is checked before anything is selected, so a design that cannot be honoured is refused outright. `continuous.controls` must be non-empty and must not name the predictor, `match_on` must name exactly the same dimensions as `continuous.controls`, every dimension named and the `key` column must be present in the pool, no `tolerance_k` may be negative, and the pool must not be empty.

Value

A data frame of the selected stimuli. Unless `label` is `NULL` the `condition` column is set to it, `"continuous"` by default, and unless `renumber_sets` is `FALSE` the `set` column is renumbered `1..n`.

---

tost_equiv *Two one-sided tests (TOST) of equivalence on a Cohen's d bound*

Description

Reports the larger of the two one-sided p-values; a value below `alpha` supports equivalence within +/- `bound_d` standard deviations. A non-significant difference test is not itself evidence of equivalence, hence TOST is reported alongside the standardised mean difference (Lakens, 2017).

Usage

```
tost_equiv(x, y, bound_d = 0.5, alpha = 0.05)
```

Arguments

`x, y`	Numeric vectors.
`bound_d`	Smallest effect size of interest (Cohen's d); defaults to the schema value of 0.5 (Lakens, 2017).
`alpha`	Significance level.

Value

A list with `p` and logical `equivalent`.

variance_ratio *Variance ratio: a distributional balance check*

Description

The ratio of a condition's variance to the reference's, complementing the mean-based Cohen's d and TOST. Two conditions can share a mean yet differ in spread and still confound, which a mean-based statistic misses (Armstrong et al., 2012; Austin, 2009). A ratio near 1 is balanced; a common heuristic flags ratios outside roughly 0.5 to 2.

Usage

```
variance_ratio(cond, ref)
```

Arguments

cond, ref	Numeric vectors (condition and reference).

Value

The variance ratio, or NA when a variance is undefined.

Examples

```
variance_ratio(c(1, 2, 3, 4), c(1, 2, 3, 8))
```

write_datasheet *Write a datasheet to a JSON record and a Markdown rendering*

Description

Write a datasheet to a JSON record and a Markdown rendering

Usage

```
write_datasheet(ds, json_path, md_path)
```

Arguments

ds	A datasheet list, from build_datasheet().
json_path	Output path for the machine-readable JSON record.
md_path	Output path for the human-readable Markdown rendering.

Value

Invisibly, the two paths written.

`write_run_log` *Write the run log to Markdown (and optionally JSON Lines)*

Description

Write the run log to Markdown (and optionally JSON Lines)

Usage

```
write_run_log(log, md_path, jsonl_path = NULL)
```

Arguments

`log`	A run-log object.
`md_path`	Output path for the Markdown log.
`jsonl_path`	Optional output path for the JSON Lines log.

Value

`md_path`, invisibly.

Index

add_bigram_frequency, 3
add_neighbourhood, 4
add_neighbourhood(), *20*, *25*
add_pair_overlap, 4
assign_triggers, 5
assign_triggers(), *13–15*

balance_check, 6
balance_lists, 6
balance_lists(), *8*, *12*
bigram_counts(), *23*
build_datasheet, 7
build_datasheet(), *26*, *34*
build_lexdec_stimuli, 9
build_pool, 10
build_pool(), *9*

cohens_d, 10
cohens_d_ci, 11
count_syllables, 12
counterbalance, 12

describe_stimuli, 13

export_experiments, 13
export_experiments(), *5*
export_jspsych, 14
export_opensesame, 15
export_psychopy, 15

fetch_corpus, 16
fetch_corpus(), *17, 18*

generate_pseudowords, 16

lexsync_cache_clear, 17
lexsync_cache_clear(), *16*, *18, 19*
lexsync_cache_dir, 18
lexsync_cache_dir(), *16, 17*
list_corpora, 19
list_corpora(), *16*
load_items, 19
load_lexicon, 20
load_pool, 21
log_artefact, 22
log_step, 22

make_pseudoword, 23
match_report, 23
match_report(), *8*, *24*
match_report_continuous, 24
match_report_continuous(), *8*
match_stimuli, 25
match_stimuli(), *10*, *29*
merge_norms, 25
methods_paragraph, 26

new_run_log, 27

PARADIGMS, 27
participant_table, 28

required_fields, 28
resample_stimuli, 29
resolve_events, 29
resolve_trial_timing, 30
run_all, 30
run_pipeline, 31
run_pipeline(), *10*

select_continuous_stimuli, 32

tost_equiv, 33

variance_ratio, 34

write_datasheet, 34
write_datasheet(), *9*
write_run_log, 35