Package: tblcheck (via r-universe)

June 5, 2026

Title Grade Tables in Learning Exercises

Version 0.3.1

Description 'tblcheck' extends 'gradethis' with functions that inspect data frame or tibble objects to make it easier for teachers to check that student tables meet expectations.

License MIT + file LICENSE

URL https://pkgs.rstudio.com/tblcheck, https://github.com/rstudio/tblcheck

BugReports https://github.com/rstudio/tblcheck/issues

Imports checkmate, dplyr, ellipsis, glue, gradethis (>= 0.2.7.9000), knitr, lifecycle, magrittr, methods, purrr, rlang, tidyselect, utils, vctrs

Suggests learnr, lubridate, mockery, rmarkdown, testthat (>= 3.0.0), tibble

VignetteBuilder knitr

Remotes rstudio/gradethis

Config/Needs/learnr rstudio/learnr, rstudio/gradethis

Config/Needs/website pkgdown, tidyverse/tidytemplate, forcats, stringr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository https://rstudio.r-universe.dev

Date/Publication 2023-03-11 01:25:15 UTC

RemoteUrl https://github.com/rstudio/tblcheck

RemoteRef HEAD

RemoteSha 361e592d5d1a43e9d817417eb945c2443970ec58

Contents

`friendly_class` *Generate a human-readable description of an object's class*

Description

Generate a human-readable description of an object's class

Usage

```
friendly_class(object)

## S4 method for signature 'ANY'
friendly_class(object)

## S4 method for signature 'character'
friendly_class(object)

## S4 method for signature 'numeric'
friendly_class(object)

## S4 method for signature 'integer'
friendly_class(object)

## S4 method for signature 'logical'
```

```
friendly_class(object)

## S4 method for signature 'complex'
friendly_class(object)

## S4 method for signature 'raw'
friendly_class(object)

## S4 method for signature 'factor'
friendly_class(object)

## S4 method for signature 'Date'
friendly_class(object)

## S4 method for signature 'POSIXt'
friendly_class(object)

## S4 method for signature 'Period'
friendly_class(object)

## S4 method for signature 'data.frame'
friendly_class(object)

## S4 method for signature 'tbl_df'
friendly_class(object)

## S4 method for signature 'grouped_df'
friendly_class(object)

## S4 method for signature 'rowwise_df'
friendly_class(object)

## S4 method for signature 'list'
friendly_class(object)

## S4 method for signature 'matrix'
friendly_class(object)

## S4 method for signature 'array'
friendly_class(object)
```

Arguments

`object`	An object whose class will be described

Value

A character string of length 1, based on the class and length of `object`.

`grade_this_table` *Grade this table*

Description

Automatically grade a table resulting from student code using `gradethis::grade_this()` and `tbl_grade()` to compare the student's result with the author's solution.

Usage

```
grade_this_table(
  correct = NULL,
  pre_check = NULL,
  post_check = NULL,
  pass_if_equal = FALSE,
  ...,
  max_diffs = 3,
  cols = NULL,
  check_class = TRUE,
  ignore_class = NULL,
  check_names = TRUE,
  check_column_order = FALSE,
  check_dimensions = TRUE,
  check_groups = TRUE,
  check_columns = TRUE,
  check_column_class = check_columns,
  check_column_levels = check_columns,
  check_column_values = check_columns,
  tolerance = sqrt(.Machine$double.eps),
  check_row_order = check_columns,
  hint = getOption("gradethis.fail.hint", FALSE),
  encourage = getOption("gradethis.fail.encourage", FALSE),
  pass.praise = NULL
)
```

Arguments

`correct` `[character(1)]`
The message shown to the student when their `.result` matches the exercise `.solution`, if `pass_if_equal` is `TRUE`.

`pre_check`, `post_check` `[expression]`
Code to run before or after the table or vector grading is performed. The pre check runs before calling `gradethis::pass_if_equal()` so that you can modify or adjust the student's `.result` or the `.solution` if there are parts of either that need to be ignored. These arguments can also be used in conjunction with the `pass_if_equal` option when the grading requirements are more involved.

`pass_if_equal` `[logical(1)]`
When `TRUE` (default for `grade_this_vector()` but not `grade_this_table()`), the `.result` is compared to the `.solution` with `gradethis::pass_if_equal()` after the *pre check* and before calling the **tblcheck** grading function.

`...` Additional arguments passed to `graded()` or additional data to be included in the feedback object.

`max_diffs` `[numeric(1)]`
The maximum number of mismatched values to display in an informative failure message. Passed to `tbl_check_names()` to determine the number of mismatched column names to display and the `n_values` argument of `tbl_check_column()` to determine the number of mismatched column values to display. Defaults to 3.

`cols` [`tidy-select`]
A selection of columns to compare between `object` and `expected`. Differences in other columns will be ignored. If `NULL`, the default, all columns will be checked.

`check_class` `[logical(1)]`
Whether to check that `object` and `expected` have the same classes with `tbl_check_class()`.

`ignore_class` `[character()]`
A vector of classes to ignore when finding differences between `object` and `expected`.
If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa.
If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.

`check_names` `[logical(1)]`
Whether to check that `object` and `expected` have the same column names with `tbl_check_names()`.

`check_column_order`
`[logical(1)]`
Whether to check that the columns of `object` are in the same order as `expected` with `tbl_check_names()`. Defaults to `FALSE`.

`check_dimensions`
`[logical(1)]`
Whether to check that `object` and `expected` have the same number of rows and columns with `tbl_check_dimensions()`.

`check_groups` `[logical(1)]`
Whether to check that `object` and `expected` have the same groups with `dplyr::group_vars()`.

`check_columns` `[logical(1)]`
Whether to check that all columns have the same contents with `tbl_check_column()`.

`check_column_class`
`[logical(1)]`
Whether to check that each column has the same class in `object` and `expected`.

`check_column_levels`
: `[logical(1)]`
Whether to check that each column has the same factor levels in `object` and `expected`.

`check_column_values`
: `[logical(1)]`
Whether to check that each column has the same values in `object` and `expected`.

`tolerance`
: `[numeric(1)  0]`
`values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.

`check_row_order`
: `[logical(1)]`
Whether to check that the values in each column are in the same order in `object` and `expected`.

`hint`
: Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage`
: Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

`pass.praise`
: Logical `TRUE` or `FALSE` to determine whether a praising phrase should be automatically prepended to any `pass()` or `pass_if_equal()` messages. Sets the `gradethis.pass.praise` option.

Value

The returned feedback is equivalent to **gradethis** grading code using `grade_this()` with the following components:

1. First the `pre_check` code, if any, is evaluated. If this code calls `pass()`, `fail()`, or their equivalents, that feedback is provided immediately.
2. If `pass_if_equal` is `TRUE`, then `pass_if_equal()` is called to compare the `.result` to the `.solution`. The message in `correct` is used for the feedback.
3. The appropriate **tblcheck** grading function is called, returning any feedback:
   (a) `grade_this_table()` returns the results from `tbl_grade()`
   (b) `grade_this_vector()` returns the results from `vec_grade()`
4. The `post_check` code, if any, is evaluated and any feedback from a call to `pass()`, `fail()`, or their equivalents is returned.
5. Finally, if no other feedback is returned, the feedback from `gradethis::fail()` is provided to the student, using the options `fail.message`, `fail.hint` and `fail.encourage`.

See Also

`tbl_grade()`

Other graders: `grade_this_vector()`

Examples

```
ex <- gradethis::mock_this_exercise(
 .solution_code = tibble::tibble(x = 1:3, y = letters[x]),
 .user_code = tibble::tibble(x = 1:3, y = c("A", "b", "c"))
)

## Grading Tables ----
grade_this_table()(ex)

# Roughly equivalent to...
gradethis::grade_this({
 gradethis::pass_if_equal()
 tbl_grade()
 gradethis::fail()
})(ex)
```

---

grade_this_vector *Grade this vector*

Description

Automatically grade a vector resulting from student code using `gradethis::grade_this()` and `vec_grade()` to compare the student's result with the author's solution.

Usage

```
grade_this_vector(
  correct = NULL,
  pre_check = NULL,
  post_check = NULL,
  pass_if_equal = TRUE,
  ...,
  max_diffs = 3,
  check_class = TRUE,
  ignore_class = NULL,
  check_length = TRUE,
  check_levels = TRUE,
  check_values = TRUE,
  tolerance = sqrt(.Machine$double.eps),
  check_names = TRUE,
  hint = getOption("gradethis.fail.hint", FALSE),
  encourage = getOption("gradethis.fail.encourage", FALSE),
  pass.praise = NULL
)
```

Arguments

`correct`	`[character(1)]` The message shown to the student when their `.result` matches the exercise `.solution`, if `pass_if_equal` is `TRUE`.
`pre_check, post_check`	`[expression]` Code to run before or after the table or vector grading is performed. The pre check runs before calling `gradethis::pass_if_equal()` so that you can modify or adjust the student's `.result` or the `.solution` if there are parts of either that need to be ignored. These arguments can also be used in conjunction with the `pass_if_equal` option when the grading requirements are more involved.
`pass_if_equal`	`[logical(1)]` When `TRUE` (default for `grade_this_vector()` but not `grade_this_table()`), the `.result` is compared to the `.solution` with `gradethis::pass_if_equal()` after the *pre check* and before calling the **tblcheck** grading function.
`...`	Additional arguments passed to `graded()` or additional data to be included in the feedback object.
`max_diffs`	`[numeric(1)]` The maximum number of mismatched values to print. Defaults to 3.
`check_class`	`[logical(1)]` Whether to check that `object` and `expected` have the same classes.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa. If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.
`check_length`	`[logical(1)]` Whether to check that `object` and `expected` have the same length.
`check_levels`	`[logical(1)]` Whether to check that `object` and `expected` have the same factor levels.
`check_values`	`[logical(1)]` Whether to check that `object` and `expected` contain the same values.
`tolerance`	`[numeric(1)  0]` `values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.
`check_names`	`[logical(1)]` Whether to check that `object` and `expected` have the same names.
`hint`	Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

`pass.praise` Logical `TRUE` or `FALSE` to determine whether a praising phrase should be automatically prepended to any `pass()` or `pass_if_equal()` messages. Sets the `gradethis.pass.praise` option.

Value

The returned feedback is equivalent to **gradethis** grading code using `grade_this()` with the following components:

1. First the `pre_check` code, if any, is evaluated. If this code calls `pass()`, `fail()`, or their equivalents, that feedback is provided immediately.
2. If `pass_if_equal` is `TRUE`, then `pass_if_equal()` is called to compare the `.result` to the `.solution`. The message in `correct` is used for the feedback.
3. The appropriate **tblcheck** grading function is called, returning any feedback:
   (a) `grade_this_table()` returns the results from `tbl_grade()`
   (b) `grade_this_vector()` returns the results from `vec_grade()`
4. The `post_check` code, if any, is evaluated and any feedback from a call to `pass()`, `fail()`, or their equivalents is returned.
5. Finally, if no other feedback is returned, the feedback from `gradethis::fail()` is provided to the student, using the options `fail.message`, `fail.hint` and `fail.encourage`.

See Also

`vec_grade()`

Other graders: `grade_this_table()`

Examples

```
ex <- gradethis::mock_this_exercise(
 .solution_code = tibble::tibble(x = 1:3, y = letters[x]),
 .user_code = tibble::tibble(x = 1:3, y = c("A", "b", "c"))
)

#' ## Grading Vectors ----
# Here we use `pre_check` to modify `.result` and
grade_this_vector(
 pre_check = {
  .result <- .result$y
  .solution <- .solution$y
 }
)(ex)

# Roughly equivalent to...
gradethis::grade_this({
 .result <- .result$y
 .solution <- .solution$y
```

```
 gradethis::pass_if_equal()
 vec_grade()
 gradethis::fail()
})(ex)
```

`hinted_class_message` *Generate a hint for how to convert one object type to another*

Description

Generate a hint for how to convert one object type to another

Usage

```
hinted_class_message(object, expected)

## S4 method for signature 'ANY,ANY'
hinted_class_message(object, expected)

## S4 method for signature 'rowwise_df,grouped_df'
hinted_class_message(object, expected)

## S4 method for signature 'data.frame,grouped_df'
hinted_class_message(object, expected)

## S4 method for signature 'grouped_df,data.frame'
hinted_class_message(object, expected)

## S4 method for signature 'data.frame,rowwise_df'
hinted_class_message(object, expected)

## S4 method for signature 'rowwise_df,data.frame'
hinted_class_message(object, expected)
```

Arguments

`object`	An object to be compared to `expected`
`expected`	An object of the expected class

Value

A character string of length 1

`problem` *Declare a problem*

Description

Useful for constructing a small list to communicate the problem that was discovered during checking.

Usage

```
problem(
  type,
  expected,
  actual,
  ...,
  .class = c(paste0(type, "_problem"), "tblcheck_problem")
)
```

Arguments

`type`	A character string, e.g. `column_values` or `table_rows`, that describes the problem that was discovered.
`expected`, `actual`	The expected and actual values. These should be included when the value is a summary, e.g. `nrow(expected)` or `length(actual)`. Be careful not to include large amounts of data.
`...`	Additional elements to be included in the `problem` object.
`.class`	The class of the problem. Typically, we expect the problem class to be `<type>_problem`, but if you are building custom classes you may set these classes as desired.

Value

Returns a problem with class `<type>_problem` and the base classes `tblcheck_problem` and `gradethis_problem`.

See Also

Other Problem functions: `problem_grade()`, `problem_message()`, `problem_type()`

Examples

```
problem(
 type = "class",
 expected = "character",
 actual = "numeric",
 expected_length = 1,
 actual_length = 2
```

```
)
```

`problem_grade` *Apply automatic grading to a problem object*

Description

Automatically converts a `problem()` object into a **gradethis** grade. `problem_grade()` is an S4 generic and **tblcheck** provides an internal method for problems with class `"tblcheck_problem"`. In **tblcheck**, or for problems with this class, any problems are automatically turned into failing grades with `gradethis::fail()` and using the message provided by `problem_message()`.

Usage

```
problem_grade(problem, max_diffs = 3, env = parent.frame(), ...)

## Default S3 method:
problem_grade(problem, max_diffs = 3, env = parent.frame(), ...)

## S3 method for class 'list'
problem_grade(problem, max_diffs = 3, env = parent.frame(), ...)

## S3 method for class 'gradethis_problem'
problem_grade(problem, max_diffs = 3, env = parent.frame(), ...)

## S3 method for class 'tblcheck_problem'
problem_grade(problem, max_diffs = 3, env = parent.frame(), ...)
```

Arguments

`problem`	A problem generated by `tbl_check()`, `vec_check()` or their related helper functions.
`max_diffs`	`[numeric(1)]` The maximum number of mismatched values to display in an informative failure message. Passed to `tbl_check_names()` to determine the number of mismatched column names to display and the `n_values` argument of `tbl_check_column()` to determine the number of mismatched column values to display. Defaults to 3.
`env`	The environment used for grading.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

A `gradethis::fail()` message or `NULL` invisibly.

See Also

Other Problem functions: `problem_message()`, `problem_type()`, `problem()`

Examples

```
.result <- 1:10
.solution <- letters[1:10]
problem <- vec_check()
problem_grade(problem)
```

---

problem_message *Create a message from a problem object*

Description

`problem_message()` is an S3 generic that powers the conversion of problems detected by `tbl_check()`, `vec_check()`, and their related helper functions into a human-readable message.

Usage

```
problem_message(problem, ...)
```

Arguments

`problem`	An object with base class `gradethis_problem`. Problems identified by **tblcheck** also include `tblcheck_problem`, plus additional classes that more specifically identify the problem type.
`...`	Additional arguments passed to the underlying methods.

Value

A length-1 character string with a message describing the problem.

See Also

Other Problem functions: `problem_grade()`, `problem_type()`, `problem()`

Examples

```
problem <- problem(
 type = "class",
 expected = "character",
 actual = "numeric",
 expected_length = 1,
 actual_length = 2
)

problem_message(problem)
```

`problem_type` *Problem helper functions*

Description

- `problem_type()` returns a problem's type, or `NULL` if the input is not a problem.
- `is_problem()` tests whether an object is a `gradethis` problem.
- `is_tblcheck_problem()` tests whether an object is a problem created by `tblcheck`.
- `as_problem()` converts a list to a `tblcheck_problem`.

Usage

```
problem_type(x)

is_problem(x, type = NULL)

is_tblcheck_problem(x, type = NULL)

as_problem(x)
```

Arguments

`x`	An object
`type`	`[character(1)]` A `problem` type

Details

If `type` is specified, `is_problem()` and `is_tblcheck_problem()` test whether an object is a problem of the specified type.

Value

`is_problem()` and `is_tblcheck_problem()` return a logical of length 1. `problem_type()` returns a character of length 1. `as_problem()` returns a `tblcheck_problem`.

See Also

Other Problem functions: `problem_grade()`, `problem_message()`, `problem()`

Examples

```
problem_type(vec_check(1, "1"))
is_problem(vec_check(1, "1"), "vector_class")
is_tblcheck_problem(vec_check(1, "1"), "class")
```

`tbl_check` *Check that the rows and columns of two tables are the same*

Description

Checks for differences between `object` and `expected` in the following order:

1. Check table class with `tbl_check_class()`
2. Check column names with `tbl_check_names()`
3. Check number of rows and columns with `tbl_check_dimensions()`
4. Check groups with `tbl_check_groups()`
5. Check that each column is the same with `tbl_check_column()`

If the tables differ

- `tbl_check()` returns a list describing the problem
- `tbl_grade()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check(
  object = .result,
  expected = .solution,
  cols = NULL,
  check_class = TRUE,
  ignore_class = NULL,
  check_names = TRUE,
  check_column_order = FALSE,
  check_dimensions = TRUE,
  check_groups = TRUE,
  check_columns = TRUE,
  check_column_class = check_columns,
  check_column_levels = check_columns,
  check_column_values = check_columns,
  tolerance = sqrt(.Machine$double.eps),
  check_row_order = check_columns,
  env = parent.frame()
```

```
)

tbl_grade(
  object = .result,
  expected = .solution,
  cols = NULL,
  max_diffs = 3,
  check_class = TRUE,
  ignore_class = NULL,
  check_names = TRUE,
  check_column_order = FALSE,
  check_dimensions = TRUE,
  check_groups = TRUE,
  check_columns = TRUE,
  check_column_class = check_columns,
  check_column_levels = check_columns,
  check_column_values = check_columns,
  tolerance = sqrt(.Machine$double.eps),
  check_row_order = check_columns,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	A data frame to be compared to `expected`.
`expected`	A data frame containing the expected result.
`cols`	[tidy-select] A selection of columns to compare between `object` and `expected`. Differences in other columns will be ignored. If `NULL`, the default, all columns will be checked.
`check_class`	`[logical(1)]` Whether to check that `object` and `expected` have the same classes with `tbl_check_class()`.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa. If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.
`check_names`	`[logical(1)]` Whether to check that `object` and `expected` have the same column names with `tbl_check_names()`.

`check_column_order`
: `[logical(1)]`
Whether to check that the columns of `object` are in the same order as `expected` with `tbl_check_names()`. Defaults to `FALSE`.

`check_dimensions`
: `[logical(1)]`
Whether to check that `object` and `expected` have the same number of rows and columns with `tbl_check_dimensions()`.

`check_groups` `[logical(1)]`
Whether to check that `object` and `expected` have the same groups with `dplyr::group_vars()`.

`check_columns` `[logical(1)]`
Whether to check that all columns have the same contents with `tbl_check_column()`.

`check_column_class`
: `[logical(1)]`
Whether to check that each column has the same class in `object` and `expected`.

`check_column_levels`
: `[logical(1)]`
Whether to check that each column has the same factor levels in `object` and `expected`.

`check_column_values`
: `[logical(1)]`
Whether to check that each column has the same values in `object` and `expected`.

`tolerance` `[numeric(1)  0]`
`values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.

`check_row_order`
: `[logical(1)]`
Whether to check that the values in each column are in the same order in `object` and `expected`.

`env` The environment in which to find `.result` and `.solution`.

`max_diffs` `[numeric(1)]`
The maximum number of mismatched values to display in an informative failure message. Passed to `tbl_check_names()` to determine the number of mismatched column names to display and the `n_values` argument of `tbl_check_column()` to determine the number of mismatched column values to display. Defaults to 3.

`...` Arguments passed on to `gradethis::fail`

`hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check()` or a `gradethis::fail()` message from `tbl_grade()`. Otherwise, invisibly returns `NULL`.

Problems

1. `class`: The table does not have the expected classes.
2. `not_table`: `object` does not inherit the data.frame class.
3. `names`: The table has column names that are not expected, or is missing names that are expected.
4. `names_order`: The table has the same column names as expected, but in a different order.
5. `ncol`: The table doesn't have the expected number of columns.
6. `nrow`: The table doesn't have the expected number of rows.
7. `groups`: The table has groups that are not expected, or is missing groups that are expected.

Additional problems may be produced by `tbl_check_column()`.

Examples

```
.result <- data.frame(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = 1:10, b = 11:20)
tbl_check()
tbl_grade()

.result <- tibble::tibble(a = 1:10, b = a, c = a, d = a, e = a, f = a)
.solution <- tibble::tibble(z = 1:10, y = z, x = z, w = z, v = z, u = z)
tbl_check()
tbl_grade()
tbl_grade(max_diffs = 5)
tbl_grade(max_diffs = Inf)

.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = 1:11, b = 12:22)
tbl_check()
tbl_grade()

.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = letters[1:10], b = letters[11:20])
tbl_check()
tbl_grade()

.result <- tibble::tibble(a = 1:10, intermediate = 6:15, b = 11:20)
.solution <- tibble::tibble(a = 1:10, b = 11:20)
tbl_check(cols = any_of(names(.solution)))
tbl_grade(cols = any_of(names(.solution)))

.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = 11:20, b = 1:10)
```

```
tbl_check()
tbl_grade()
tbl_grade(max_diffs = 5)
tbl_grade(max_diffs = Inf)

.result <- tibble::tibble(a = 1:10, b = rep(1:2, 5))
.solution <- dplyr::group_by(tibble::tibble(a = 1:10, b = rep(1:2, 5)), b)
tbl_check()
tbl_grade()
tbl_grade(check_groups = FALSE)
```

`tbl_check_class` *Checks that two objects have the same classes*

Description

Checks if `object` and `expected` have the same class. If the classes differ

- `tbl_check_class()` and `vec_check_class()` return a list describing the problem
- `tbl_grade_class()` and `vec_grade_class()` return a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_class(
  object = .result,
  expected = .solution,
  ignore_class = NULL,
  env = parent.frame()
)

vec_check_class(
  object = .result,
  expected = .solution,
  ignore_class = NULL,
  env = parent.frame()
)

tbl_grade_class(
  object = .result,
  expected = .solution,
  ignore_class = NULL,
  env = parent.frame(),
  ...
)

vec_grade_class(
  object = .result,
```

```
  expected = .solution,
  ignore_class = NULL,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	An object to be compared to `expected`.
`expected`	An object containing the expected result.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa. If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.
`env`	The environment in which to find `.result` and `.solution`.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_class()` and `vec_check_class()` or a `gradethis::fail()` message from `tbl_grade_class()` and `vec_grade_class()`. Otherwise, invisibly returns `NULL`.

Problems

1. `class`: The object does not have the expected classes

Examples

```
.result <- 1:10
.solution <- as.character(1:10)
vec_check_class()
vec_grade_class()

.result <- data.frame(a = 1:10)
```

```
.solution <- tibble::tibble(a = 1:10)
tbl_check_class()
tbl_grade_class()

.result <- tibble::tibble(a = 1:10, b = a %% 2 == 0)
.solution <- dplyr::group_by(tibble::tibble(a = 1:10, b = a %% 2 == 0), b)
tbl_check_class()
tbl_grade_class()

# Ignore the difference between tibble and data frame
.result <- data.frame(a = 1:10)
.solution <- tibble::tibble(a = 1:10)
tbl_check_class(ignore_class = c("tbl_df", "tbl"))
tbl_grade_class(ignore_class = c("tbl_df", "tbl"))

# Ignore the difference between integer and double
.result <- 1L
.solution <- 1
vec_check_class(ignore_class = c("integer" = "numeric"))
vec_grade_class(ignore_class = c("integer" = "numeric"))
```

`tbl_check_column` *Checks that a column is identical across two tables*

Description

Checks for differences between the `name` column in `object` and in `expected` in the following order:

1. Check that the `name` column exists in `object`
2. Check class with `vec_check_class()`
3. Check length with `vec_check_dimensions()`
4. If the column is a factor, check factor levels with `vec_check_levels()`
5. Check column values with `vec_check_values()`

If the columns differ

- `tbl_check_column()` returns a list describing the problem
- `tbl_grade_column()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_column(
  column,
  object = .result,
  expected = .solution,
  check_class = TRUE,
```

```
  ignore_class = NULL,
  check_length = TRUE,
  check_levels = TRUE,
  check_values = TRUE,
  tolerance = sqrt(.Machine$double.eps),
  check_names = FALSE,
  env = parent.frame()
)

tbl_grade_column(
  column,
  object = .result,
  expected = .solution,
  max_diffs = 3,
  check_class = TRUE,
  ignore_class = NULL,
  check_length = TRUE,
  check_levels = TRUE,
  check_values = TRUE,
  tolerance = sqrt(.Machine$double.eps),
  check_names = FALSE,
  env = parent.frame(),
  ...
)
```

Arguments

`column`	`[character(1)]` The name of the column to check.
`object`	A data frame to be compared to `expected`.
`expected`	A data frame containing the expected result.
`check_class`	`[logical(1)]` Whether to check that `column` has the same class in `object` and `expected`.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa. If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.
`check_length`	`[logical(1)]` Whether to check that `column` has the same length in `object` and `expected`.
`check_levels`	`[logical(1)]` Whether to check that `column` and has the same factor levels in `object` and `expected`.

`check_values` `[logical(1)]`
Whether to check that `column` has the same values in `object` and `expected`.

`tolerance` `[numeric(1)  0]`
`values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.

`check_names` `[logical(1)]`
Whether to check that `column` has the same names in `object` and `expected`. Defaults to `FALSE`.

`env` The environment in which to find `.result` and `.solution`.

`max_diffs` `[numeric(1)]`
The maximum number of mismatched values to print. Defaults to 3.

`...` Arguments passed on to `gradethis::fail`

`hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_column()` or a `gradethis::fail()` message from `tbl_grade_column()`. Otherwise, invisibly returns `NULL`.

Problems

1. `names` (`table_problem`): `object` doesn't contain a column named column.
2. `class`: Any mismatch in the classes of the `column`.
3. `length`: The `column` doesn't have the expected length.
4. `levels_n`, `levels`, `levels_reversed`, `levels_order`: See `vec_check_levels()`.
5. `values`: The `column` doesn't have the expected values.
6. `names` (`column_problem`): The `column` has different names than expected.
7. `names_order`: The `column` has the same names as expected, but in a different order.

Examples

```
.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = letters[1:10], b = letters[11:20])
tbl_check_column("a")
tbl_grade_column("a")

.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = 1:11, b = 12:22)
tbl_check_column("a")
tbl_grade_column("a")
```

```
.result <- tibble::tibble(a = 1:10, b = 11:20)
.solution <- tibble::tibble(a = 11:20, b = 1:10)
tbl_check_column("a")
tbl_grade_column("a")
tbl_grade_column("a", max_diffs = 5)
tbl_grade_column("a", max_diffs = Inf)
```

`tbl_check_dimensions` *Check that the dimensions of two object are the same*

Description

Checks if `object` and `expected` have the same dimenisons. If the dimensions differ

- `tbl_check_dimensions()` returns a list describing the problem
- `tbl_grade_dimensions()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_dimensions(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame()
)

vec_check_dimensions(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame()
)

vec_check_length(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame()
)

tbl_grade_dimensions(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame(),
  ...
)
```

```
vec_grade_dimensions(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame(),
  ...
)

vec_grade_length(
  object = .result,
  expected = .solution,
  check_ncol = TRUE,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	An object to be compared to `expected`.
`expected`	An object containing the expected result.
`check_ncol`	`[logical(1)]` Whether to check that `object` and `expected` have the same number of columns.
`env`	The environment in which to find `.result` and `.solution`.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_dimensions()` or a `gradethis::fail()` message from `tbl_grade_dimensions()`. Otherwise, invisibly returns `NULL`.

Problems

1. `dimensions_n`: `object` and `expected` have a different number of dimensions
2. `length`: `object` and `expected` are one-dimensional vectors of different lengths
3. `ncol`: `object` and `expected` are two-dimensional objects with a different number of columns

4. `nrow`: `object` and `expected` are two-dimensional objects with a different number of rows
5. `dimensions`: `object` and `expected` are multi-dimensional arrays with different dimensions

Examples

```
.result <- 1:10
.solution <- 1:5
tbl_check_dimensions()
tbl_grade_dimensions()

.result <- tibble::tibble(a = 1:10, b = 1:10, c = 1:10)
.solution <- tibble::tibble(a = 1:10, b = 1:10)
tbl_check_dimensions()
tbl_grade_dimensions()

.result <- tibble::tibble(a = 1:10, b = 1:10)
.solution <- tibble::tibble(a = 1:5, b = 1:5)
tbl_check_dimensions()
tbl_grade_dimensions()

.result <- 1:12
.solution <- matrix(1:12, 3)
tbl_check_dimensions()
tbl_grade_dimensions()
```

`tbl_check_groups` *Check that the groups of two object are the same*

Description

Checks if `object` and `expected` have the same groups. If the groups differ

- `tbl_check_groups()` returns a list describing the problem
- `tbl_grade_groups()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_groups(object = .result, expected = .solution, env = parent.frame())

tbl_grade_groups(
  object = .result,
  expected = .solution,
  max_diffs = 3,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	An object to be compared to `expected`.
`expected`	An object containing the expected result.
`env`	The environment in which to find `.result` and `.solution`.
`max_diffs`	`[numeric(1)]` The maximum number of missing and/or unexpected names to include in an informative failure message. Defaults to 3.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_groups()` or a `gradethis::fail()` message from `tbl_grade_groups()`. Otherwise, invisibly returns `NULL`.

Problems

1. `groups`: The object has groups that are not expected, or is missing groups that are expected.

Examples

```
.result <- dplyr::group_by(tibble::tibble(a = 1:10, b = 11:20), a)
.solution <- dplyr::group_by(tibble::tibble(a = 1:10, b = 11:20), b)
tbl_check_groups()
tbl_grade_groups()
```

`tbl_check_is_table` *Checks that an object is a table*

Description

Checks if `object` inherits the data.frame class. If the not

- `tbl_check_is_table()` returns a list describing the problem
- `tbl_grade_is_table()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_is_table(object = .result, env = parent.frame())

tbl_grade_is_table(object = .result, env = parent.frame(), ...)
```

Arguments

`object`	An object to be compared to `expected`.
`env`	The environment in which to find `.result` and `.solution`.
`...`	Arguments passed on to `gradethis::fail`

`hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_is_table()` or a `gradethis::fail()` message from `tbl_grade_is_table()`. Otherwise, invisibly returns `NULL`.

Problems

1. `not_table`: The object is not a table

Examples

```
.result <- data.frame(a = 1:10)
tbl_check_is_table()
tbl_grade_is_table()

.result <- tibble::tibble(a = 1:10)
tbl_check_is_table()
tbl_grade_is_table()

.result <- list(a = 1:10)
tbl_check_is_table()
tbl_grade_is_table()
```

`tbl_check_names` *Check that the names of two object are the same*

Description

Checks if `object` and `expected` have the same names. If the names differ

- `tbl_check_names()` and `vec_check_names()` returns a list describing the problem
- `tbl_grade_names()` and `vec_grade_names()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
tbl_check_names(
  object = .result,
  expected = .solution,
  check_order = TRUE,
  env = parent.frame()
)

vec_check_names(
  object = .result,
  expected = .solution,
  check_order = TRUE,
  env = parent.frame()
)

tbl_grade_names(
  object = .result,
  expected = .solution,
  max_diffs = 3,
  check_order = TRUE,
  env = parent.frame(),
  ...
)

vec_grade_names(
  object = .result,
  expected = .solution,
  max_diffs = 3,
  check_order = TRUE,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	An object to be compared to `expected`.
`expected`	An object containing the expected result.
`check_order`	`[logical(1)]` Whether to check that the names of `object` and `expected` are in the same order.
`env`	The environment in which to find `.result` and `.solution`.
`max_diffs`	`[numeric(1)]` The maximum number of missing and/or unexpected names to include in an informative failure message. Defaults to 3.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `tbl_check_names()` and `vec_check_names()` or a `gradethis::fail()` message from `tbl_grade_names()` and `vec_grade_names()`. Otherwise, invisibly returns `NULL`.

Problems

1. `names`: The object has names that are not expected, or is missing names that are expected.
2. `names_order`: The object has the same names as expected, but in a different order.

Examples

```
.result <- c(1, 2, 3, 4, 5, 6, 7)
.solution <- c(a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, g = 7)
vec_check_names()
vec_grade_names()
vec_grade_names(max_diffs = 5)
vec_grade_names(max_diffs = Inf)

.result <- tibble::tibble(a = 1:5, b = 6:10, c = 11:15)
.solution <- tibble::tibble(a = 1:5, x = 6:10, y = 11:15)
tbl_check_names()
tbl_grade_names()
```

`tbl_equal` *Check that the rows and columns of two tables are the same*

Description

Test if two tables are equivalent using the same process as `tbl_check()`. Unlike `tbl_check()`, which returns either a problem object or `NULL`, `tbl_equal()` returns either `TRUE` or `FALSE`.

Usage

```
tbl_equal(
  object = .result,
  expected = .solution,
  cols = NULL,
  check_class = TRUE,
  ignore_class = NULL,
  check_names = TRUE,
  check_column_order = FALSE,
  check_dimensions = TRUE,
  check_groups = TRUE,
  check_columns = TRUE,
  check_column_class = check_columns,
  check_column_levels = check_columns,
  check_column_values = check_columns,
  tolerance = sqrt(.Machine$double.eps),
  check_row_order = check_columns,
  env = parent.frame()
)
```

Arguments

`object`	A data frame to be compared to `expected`.
`expected`	A data frame containing the expected result.
`cols`	[tidy-select] A selection of columns to compare between `object` and `expected`. Differences in other columns will be ignored. If `NULL`, the default, all columns will be checked.
`check_class`	`[logical(1)]` Whether to check that `object` and `expected` have the same classes with `tbl_check_class()`.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer"`

= "numeric") will ignore class differences only if object has class integer and expected has class numeric, or vice versa.

If all the classes of expected are included in ignore_class, a class problem will never be returned.

check_names [logical(1)]
Whether to check that object and expected have the same column names with tbl_check_names().

check_column_order
[logical(1)]
Whether to check that the columns of object are in the same order as expected with tbl_check_names(). Defaults to FALSE.

check_dimensions
[logical(1)]
Whether to check that object and expected have the same number of rows and columns with tbl_check_dimensions().

check_groups [logical(1)]
Whether to check that object and expected have the same groups with dplyr::group_vars().

check_columns [logical(1)]
Whether to check that all columns have the same contents with tbl_check_column().

check_column_class
[logical(1)]
Whether to check that each column has the same class in object and expected.

check_column_levels
[logical(1)]
Whether to check that each column has the same factor levels in object and expected.

check_column_values
[logical(1)]
Whether to check that each column has the same values in object and expected.

tolerance [numeric(1) 0]
values differences smaller than tolerance are ignored. The default value is close to 1.5e-8.

check_row_order
[logical(1)]
Whether to check that the values in each column are in the same order in object and expected.

env The environment in which to find .result and .solution.

Value

A TRUE or FALSE value.

Examples

```
tbl_equal(
```

```
 data.frame(a = 1:10, b = 11:20),
 data.frame(b = 11:20, a = 1:10)
)
```

`vec_check` *Checks that two vectors are the same*

Description

Checks for differences between `object` and `expected` in the following order:

1. Check class with `vec_check_class()`
2. Check length with `vec_check_dimensions()`
3. If the vector is a factor, check factor levels are the same with `vec_check_levels()`
4. Check vector values are the same with `vec_check_values()`
5. Check names with `vec_check_names()`

If the vectors differ

- `vec_check()` returns a list describing the problem
- `vec_grade()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
vec_check(
  object = .result,
  expected = .solution,
  check_class = TRUE,
  ignore_class = NULL,
  check_length = TRUE,
  check_levels = TRUE,
  check_values = TRUE,
  tolerance = sqrt(.Machine$double.eps),
  check_names = TRUE,
  env = parent.frame()
)

vec_grade(
  object = .result,
  expected = .solution,
  max_diffs = 3,
  check_class = TRUE,
  ignore_class = NULL,
  check_length = TRUE,
  check_levels = TRUE,
  check_values = TRUE,
  tolerance = sqrt(.Machine$double.eps),
```

```
  check_names = TRUE,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	A vector to be compared to `expected`.
`expected`	A vector containing the expected result.
`check_class`	`[logical(1)]` Whether to check that `object` and `expected` have the same classes.
`ignore_class`	`[character()]` A vector of classes to ignore when finding differences between `object` and `expected`. If an element is named, differences will only be ignored between the pair of the element and its name. For example, `ignore_class = c("integer" = "numeric")` will ignore class differences only if `object` has class integer and `expected` has class numeric, or vice versa. If all the classes of `expected` are included in `ignore_class`, a `class` problem will never be returned.
`check_length`	`[logical(1)]` Whether to check that `object` and `expected` have the same length.
`check_levels`	`[logical(1)]` Whether to check that `object` and `expected` have the same factor levels.
`check_values`	`[logical(1)]` Whether to check that `object` and `expected` contain the same values.
`tolerance`	`[numeric(1)  0]` `values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.
`check_names`	`[logical(1)]` Whether to check that `object` and `expected` have the same names.
`env`	The environment in which to find `.result` and `.solution`.
`max_diffs`	`[numeric(1)]` The maximum number of mismatched values to print. Defaults to 3.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `vec_check()` or a `gradethis::fail()` message from `vec_grade()`. Otherwise, invisibly returns `NULL`.

Problems

1. `class`: `object` doesn't have the same classes as `expected`.
2. `length`: `object` doesn't have the same length as `expected`.
3. `levels_n`, `levels`, `levels_reversed`, `levels_order`: See `vec_check_levels()`.
4. `values`: `object` doesn't contain the same values as `expected`.
5. `names`: `object` has different names than `expected`.
6. `names_order`: `object` has the same names as `expected`, but in a different order.

Examples

```
.result <- 1:10
.solution <- letters[1:10]
vec_check()
vec_grade()

.result <- 1:10
.solution <- 1:11
vec_check()
vec_grade()

.result <- 1:10
.solution <- rlang::set_names(1:10, letters[1:10])
vec_check()
vec_grade()
vec_grade(max_diffs = 5)
vec_grade(max_diffs = Inf)

.result <- 1:10
.solution <- 11:20
vec_check()
vec_grade()
vec_grade(max_diffs = 5)
vec_grade(max_diffs = Inf)
```

`vec_check_levels` *Check that the levels of two factors are the same*

Description

Checks if `object` and `expected` have the same levels. If the levels differ

- `vec_check_levels()` returns a list describing the problem
- `vec_grade_levels()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
vec_check_levels(object = .result, expected = .solution, env = parent.frame())

vec_grade_levels(
  object = .result,
  expected = .solution,
  max_diffs = 3,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	An object to be compared to `expected`.
`expected`	An object containing the expected result.
`env`	The environment in which to find `.result` and `.solution`.
`max_diffs`	`[numeric(1)]` The maximum number of missing and/or unexpected names to include in an informative failure message. Defaults to 3.
`...`	Arguments passed on to `gradethis::fail` `hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option. `encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `vec_check_levels()` or a `gradethis::fail()` message from `vec_grade_levels()`. Otherwise, invisibly returns `NULL`.

Problems

1. `levels_n`: `object` and `expected` have a different number of levels.
2. `levels`: The object has levels that are not expected, or is missing levels that are expected.
3. `levels_reversed`: The `levels` of `object` are in the opposite order of `expected`.
4. `level_order`: The levels of `object` are not in the same order as `expected`.

Examples

```
.result <- as.factor(rep_len(letters[1:3], 6))
.solution <- as.factor(rep_len(letters[1:2], 6))
vec_check_levels()
```

```
vec_grade_levels()

.result <- as.factor(letters[1:6])
.solution <- as.factor(letters[21:26])
vec_check_levels()
vec_grade_levels()
vec_grade_levels(max_diffs = 5)
vec_grade_levels(max_diffs = Inf)
```

`vec_check_values` *Checks that two vectors are contain the same values*

Description

Check if two vectors contain the same values. If the values differ

- `vec_check_values()` returns a list describing the problem
- `vec_grade_values()` returns a failing grade and informative message with `gradethis::fail()`

Usage

```
vec_check_values(
  object = .result,
  expected = .solution,
  tolerance = sqrt(.Machine$double.eps),
  env = parent.frame()
)

vec_grade_values(
  object = .result,
  expected = .solution,
  tolerance = sqrt(.Machine$double.eps),
  max_diffs = 3,
  env = parent.frame(),
  ...
)
```

Arguments

`object`	A vector to be compared to `expected`.
`expected`	A vector containing the expected result.
`tolerance`	`[numeric(1)  0]` `values` differences smaller than `tolerance` are ignored. The default value is close to `1.5e-8`.
`env`	The environment in which to find `.result` and `.solution`.
`max_diffs`	`[numeric(1)]` The maximum number of mismatched values to print. Defaults to 3.

`...` Arguments passed on to `gradethis::fail`

`hint` Include a code feedback hint with the failing message? This argument only applies to `fail()` and `fail_if_equal()` and the message is added using the default options of `give_code_feedback()` and `maybe_code_feedback()`. The default value of `hint` can be set using `gradethis_setup()` or the `gradethis.fail.hint` option.

`encourage` Include a random encouraging phrase with `random_encouragement()`? The default value of `encourage` can be set using `gradethis_setup()` or the `gradethis.fail.encourage` option.

Value

If there are any issues, a list from `vec_check_values()` or a `gradethis::fail()` message from `vec_grade_values()`. Otherwise, invisibly returns `NULL`.

Problems

1. `values`: `object` doesn't contain the same values as `expected`

Examples

```
.result <- 1:10
.solution <- letters[1:10]
vec_check_values()
vec_grade_values()

.result <- 1:10
.solution <- 1:11
vec_check_values()
vec_grade_values()

.result <- 1:10
.solution <- rlang::set_names(1:10, letters[1:10])
vec_check_values()
vec_grade_values()
vec_grade_values(max_diffs = 5)
vec_grade_values(max_diffs = Inf)

.result <- 1:10
.solution <- 11:20
vec_check_values()
vec_grade_values()
vec_grade_values(max_diffs = 5)
vec_grade_values(max_diffs = Inf)
```

Index