Package: learnr (via r-universe)

October 4, 2024

Type Package

Title Interactive Tutorials for R

Version 0.11.5.9000

Description Create interactive tutorials using R Markdown. Use a combination of narrative, figures, videos, exercises, and quizzes to create self-paced tutorials for learning about R and R packages.

License Apache License (>= 2.0)

URL `https://rstudio.github.io/learnr/`, `https://github.com/rstudio/learnr`

BugReports `https://github.com/rstudio/learnr/issues`

Imports checkmate, digest, evaluate, htmltools (>= 0.3.5), htmlwidgets, jsonlite, knitr (>= 1.31), lifecycle, markdown (>= 1.3), parallel, promises, rappdirs, renv (>= 0.8.0), rlang (>= 1.0.0), rmarkdown (>= 1.12.0), rprojroot, shiny (>= 1.0), stats, utils, withr

Suggests bslib, callr, curl, DBI (>= 0.4-1), httpuv, later, reticulate, RSQLite, rstudioapi (>= 0.11), shinytest2, sortable, testthat (>= 3.0.3)

VignetteBuilder knitr

Config/Needs/connect rsconnect

Config/Needs/coverage covr

Config/Needs/website pkgdown, tidyverse/tidytemplate

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

SystemRequirements pandoc (>= 1.14) - http://pandoc.org

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

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

RemoteRef HEAD

RemoteSha a644d2dd4c083c7596fd73edc0431bcb52811357

Contents

`answer` *Question answer options*

Description

Create options for users when used in `question_checkbox()` and `question_radio()` learnr questions. For `question_text()` and `question_numeric()`, the individual answers aren't directly presented to students, but their values can be used in determining if the student submitted the correct answer. For flexible feedback from checkbox, text, and numeric questions, `answer_fn()` can be used to provide a function that evaluates the student's submission and returns a custom result.

Usage

```
answer(text, correct = FALSE, message = NULL, label = text)

answer_fn(fn, label = NULL)
```

Arguments

`text`	The answer text or value; for selection-type questions this value is shown to the user.
`correct`	Logical value indicating whether the `answer()` corresponds to a correct or incorrect option.
`message`	A custom message shown when this answer is selected and when the overall question result matches the state of this answer. For example, the `message` of a correct solution is not shown when the entire submission is incorrect, but *will* be shown when the user both picks this answer option and the question is *correct*.
`label`	The label shown when the option is presented to the user.
`fn`	A function used to evaluate the submitted answer. The function is called with the student's submitted value as the first argument, so the function should take at least one argument where the user's value will be passed to the first argument. Inline **purrr**-style lambda functions are allowed, see `rlang::as_function()` for complete details on the syntax. In the body of the function, you can perform arbitrary calculations to decide if the submitted answer is or is not correct and to compose the message presented to the user. To signal a final answer, call `mark_as()` or its helper functions `correct()` or `incorrect()`. All other return values are ignored; e.g. by returning `NULL` you may yield the submission evaluation to other `answer()` or `answer_fn()` options for the question.

Value

Returns a list with the `"tutorial_question_answer"` class.

Functions

- `answer()`: Create an answer option
- `answer_fn()`: Evaluate the student's submission to determine correctness and to return feedback.

Examples

```
answer(32, correct = FALSE)
answer(42, correct = TRUE, message = "The meaning of life.")
```

`available_tutorials` *List available tutorials*

Description

List the tutorials that are currently available via installed R packages. Or list the specific tutorials that are contained within a given R package.

Usage

```
available_tutorials(package = NULL)
```

Arguments

`package`	Name of package

Value

`available_tutorials()` returns a `data.frame` containing "package", "name", "title", "description", "package_dependencies", "private", and "yaml_front_matter".

Examples

```
available_tutorials(package = "learnr")
```

`correct` *Mark submission as correct or incorrect*

Description

Helper method to communicate that the user's submission was correct or incorrect. These functions were originally designed for developers to create `question_is_correct()` methods for custom question types, but they can also be called inside the functions created by `answer_fn()` to dynamically determine the result and message provided to the user.

Usage

```
correct(messages = NULL)

incorrect(messages = NULL)

mark_as(correct, messages = NULL)
```

Arguments

`messages`	A vector of messages to be displayed. The type of message will be determined by the `correct` value. Note that markdown messages are not rendered into HTML, but you may provide HTML using `htmltools::HTML()` or htmltools::tags.
`correct`	Logical: is the question answer is correct

Value

Returns a list with class `learnr_mark_as` to be returned from the `question_is_correct()` method for the learnr question type.

See Also

`answer_fn()`

Examples

```
# Radio button question implementation of `question_is_correct`
question_is_correct.radio <- function(question, value, ...) {
  for (ans in question$answers) {
    if (as.character(ans$option) == value) {
      return(mark_as(ans$correct, ans$message))
    }
  }
  mark_as(FALSE, NULL)
}
```

`disable_all_tags` *Disable all html tags*

Description

Method to disable all html tags to not allow users to interact with the html.

Usage

```
disable_all_tags(ele)
```

Arguments

`ele`	html tag element

Value

An **htmltools** HTML object with appended `class = "disabled"` and `disabled` attributes on all tags.

Examples

```
# add an href to all a tags
disable_all_tags(
  htmltools::tagList(
    htmltools::a(),
    htmltools::a()
  )
)
```

---

duplicate_env *Create a duplicate of an environment*

Description

Copy all items from the environment to a new environment. By default, the new environment will share the same parent environment.

Usage

```
duplicate_env(envir, parent = parent.env(envir))
```

Arguments

envir	environment to duplicate
parent	parent environment to set for the new environment. Defaults to the parent environment of `envir`.

Value

A duplicated copy of `envir` whose parent env is `parent`.

Examples

```
# Make a new environment with the object 'key'
envir <- new.env()
envir$key <- "value"
"key" %in% ls() # FALSE
"key" %in% ls(envir = envir) # TRUE

# Duplicate the envir and show it contains 'key'
new_envir <- duplicate_env(envir)
"key" %in% ls(envir = new_envir) # TRUE
```

`event_register_handler` *Register an event handler callback*

Description

Register an event handler on a per-tutorial basis. Handlers for an event will be fired in the order that they were registered.

Usage

```
event_register_handler(event, callback)
```

Arguments

`event`	The name of an event.
`callback`	A function to be invoked when an event with a specified name occurs. The callback must take parameters `session`, `event`, and `data`.

Details

In most cases, this will be called within a learnr document. If that is the case, then the handler will exist as long as the document (that is, the Shiny application) is running.

If this function is called in a learnr .Rmd document, it should be in a chunk with `context="server-start"`. If it is called with `context="server"`, the handler will be registered at least two times (once for the application as a whole, and once per user session).

If this function is called outside of a learnr document, then the handler will persist until the learnr package is unloaded, typically when the R session is stopped.

Value

A function which, if invoked, will remove the callback.

`external_evaluator` *External execution evaluator*

Description

Lifecycle: experimental

Usage

```
external_evaluator(
  endpoint = getOption("tutorial.external.host",
    Sys.getenv("TUTORIAL_EXTERNAL_EVALUATOR_HOST", NA)),
  max_curl_conns = 50
)
```

Arguments

`endpoint`	The HTTP(S) endpoint to POST the exercises to
`max_curl_conns`	The maximum number of simultaneous HTTP requests to the endpoint.

Value

A function that takes an expression (`expr`), `timelimit`, `exercise` and `session`.

`filesystem_storage` *Filesystem-based storage for tutor state data*

Description

Tutorial state storage handler that uses the filesystem as a backing store. The directory will contain tutorial state data partitioned by `user_id`, `tutorial_id`, and `tutorial_version` (in that order)

Usage

```
filesystem_storage(dir, compress = TRUE)
```

Arguments

`dir`	Directory to store state data within
`compress`	Should `.rds` files be compressed?

Value

Storage handler suitable for `options(tutorial.storage = ...)`

`finalize_question` *Finalize a question*

Description

Mark a question as finalized by adding a `question-final` class to the HTML output at the top level, in addition to disabling all tags with `disable_all_tags()`.

Usage

```
finalize_question(ele)
```

Arguments

`ele`	html tag element

Value

An **htmltools** HTML object with appropriately appended classes such that a tutorial question is marked as the final answer.

Examples

```
# finalize the question UI
finalize_question(
  htmltools::div(
    class = "custom-question",
    htmltools::div("answer 1"),
    htmltools::div("answer 2")
  )
)
```

`format.tutorial_question_answer`
Formatting and printing quizzes, questions, and answers

Description

Notes:

- If custom question types are created, custom s3 formating methods may be implemented as well.
- Due to the shiny runtime of questions, a text representation of quizzes, questions, and answers will be presented.

Usage

```
## S3 method for class 'tutorial_question_answer'
format(x, ..., spacing = "")

## S3 method for class 'tutorial_question'
format(x, ..., spacing = "")

## S3 method for class 'tutorial_quiz'
format(x, ...)

## S3 method for class 'tutorial_question'
print(x, ...)

## S3 method for class 'tutorial_question_answer'
print(x, ...)

## S3 method for class 'tutorial_quiz'
print(x, ...)
```

Arguments

`x`	object of interest
`...`	ignored
`spacing`	Text to be placed at the beginning of each new line

See Also

`quiz`, `question`, `answer`

Examples

```
ex_question <- question("What number is the letter A in the alphabet?",
  answer("8"),
  answer("14"),
  answer("1", correct = TRUE),
  answer("23"),
 incorrect = "See [here](https://en.wikipedia.org/wiki/English_alphabet) and try again.",
  allow_retry = TRUE
)
cat(format(ex_question), "\n")
```

`get_tutorial_info` *Get information about the current tutorial*

Description

Returns information about the current tutorial. Ideally the function should be evaluated in a Shiny context, i.e. in a chunk with option `context = "server"`. Note that the values of this function may change after the tutorial is completely initialized. If called in a non-reactive context, `get_tutorial_info()` will return default values that will most likely correspond to the current tutorial.

Usage

```
get_tutorial_info(
  tutorial_path = NULL,
  session = getDefaultReactiveDomain(),
  ...,
  encoding = "UTF-8"
)
```

Arguments

`tutorial_path`	Path to a tutorial `.Rmd` source file
`session`	The `session` object passed to function given to `shinyServer`. Default is `shiny::getDefaultReactiveD`
`...`	Arguments passed on to `rmarkdown::render`

output_format The R Markdown output format to convert to. The option "all" will render all formats defined within the file. The option can be the name of a format (e.g. "html_document") and that will render the document to that single format. One can also use a vector of format names to render to multiple formats. Alternatively, you can pass an output format object (e.g. html_document()). If using NULL then the output format is the first one defined in the YAML frontmatter in the input file (this defaults to HTML if no format is specified there). If you pass an output format object to output_format, the options specified in the YAML header or _output.yml will be ignored and you must explicitly set all the options you want when you construct the object. If you pass a string, the output format will use the output parameters in the YAML header or _output.yml.

output_dir The output directory for the rendered output_file. This allows for a choice of an alternate directory to which the output file should be written (the default output directory of that of the input file). If a path is provided with a filename in output_file the directory specified here will take precedence. Please note that any directory path provided will create any necessary directories if they do not exist.

output_options List of output options that can override the options specified in metadata (e.g. could be used to force self_contained or mathjax = "local"). Note that this is only valid when the output format is read from metadata (i.e. not a custom format object passed to output_format).

output_yaml Paths to YAML files specifying output formats and their configurations. The first existing one is used. If none are found, then the function searches YAML files specified to the output_yaml top-level parameter in the YAML front matter, _output.yml or _output.yaml, and then uses the first existing one.

intermediates_dir Intermediate files directory. If a path is specified then intermediate files will be written to that path. If NULL, intermediate files are written to the same directory as the input file.

knit_root_dir The working directory in which to knit the document; uses knitr's root.dir knit option. If NULL then the behavior will follow the knitr default, which is to use the parent directory of the document.

runtime The runtime target for rendering. The static option produces output intended for static files; shiny produces output suitable for use in a Shiny document (see run). The default, auto, allows the runtime target specified in the YAML metadata to take precedence, and renders for a static runtime target otherwise.

clean Using TRUE will clean intermediate files that are created during rendering.

params A list of named parameters that override custom params specified within the YAML front-matter (e.g. specifying a dataset to read or a date range to confine output to). Pass "ask" to start an application that helps guide parameter configuration.

knit_meta (This option is reserved for expert use.) Metadata generated by **knitr**.

envir The environment in which the code chunks are to be evaluated during knitting (can use new.env() to guarantee an empty new environment).

run_pandoc An option for whether to run pandoc to convert Markdown output.

quiet An option to suppress printing during rendering from knitr, pandoc command line and others. To only suppress printing of the last "Output created: " message, you can set rmarkdown.render.message to FALSE

encoding	Ignored. The encoding is always assumed to be UTF-8.

Value

Returns an ordinary list with the following elements:

- tutorial_id: The ID of the tutorial, auto-generated or from the tutorial$id key in the tutorial's YAML front matter.
- tutorial_version: The tutorial's version, auto-generated or from the tutorial$version key in the tutorial's YAML front matter.
- items: A data frame with columns order, label, type and data describing the items (questions and exercises) in the tutorial. This item is only available in the running tutorial, not during the static pre-render step.
- user_id: The current user.
- learnr_version: The current version of the running learnr package.
- language: The current language of the tutorial, either as chosen by the user or as specified in the language item of the YAML front matter.

See Also

get_tutorial_state()

Examples

```
if (rmarkdown::pandoc_available("1.4")) {
  tutorial_rmd <- local({
    # Use a temp copy of "Hello learnr" tutorial for this example
    src <- system.file(
      "tutorials", "hello", "hello.Rmd", package = "learnr"
    )
    dest <- tempfile(fileext = ".Rmd")
    file.copy(src, dest)
    dest
  })

  # ---- This is the example! ------------ #
  info <- get_tutorial_info(tutorial_rmd)
  # ------------------------------------- #

  # clean up the temporary Rmd used in this example
  unlink(tutorial_rmd)

  # This is the result of the example
  info
}
```

`get_tutorial_state` *Observe the user's progress in the tutorial*

Description

As a student progresses through a **learnr** tutorial, their progress is stored in a Shiny reactive values list for their session (see `shiny::reactiveValues()`). Without arguments, `get_tutorial_state()` returns the full reactiveValues object that can be converted to a conventional list with `shiny::reactiveValuesToList()`. If the `label` argument is provided, the state of an individual question or exercise with that label is returned.

Calling `get_tutorial_state()` introduces a reactive dependency on the state of returned questions or exercises unless called within `isolate()`. Note that `get_tutorial_state()` will only work for the tutorial author and must be used in a reactive context, i.e. within `shiny::observe()`, `shiny::observeEvent()`, or `shiny::reactive()`. Any logic observing the user's tutorial state must be written inside a `context="server"` chunk in the tutorial's R Markdown source.

Usage

```
get_tutorial_state(label = NULL, session = getDefaultReactiveDomain())
```

Arguments

`label`	A length-1 character label of the exercise or question.
`session`	The `session` object passed to function given to `shinyServer`. Default is `shiny::getDefaultReactiveD`

Value

A reactiveValues object or a single reactive value (if `label` is provided). The names of the full reactiveValues object correspond to the label of the question or exercise. Each item contains the following entries:

- `type`: One of `"question"` or `"exercise"`.
- `answer`: A character vector containing the user's submitted answer(s).
- `correct`: A logical indicating whether the user's answer was correct, or a logical `NA` if the submission was not checked for correctness.
- `timestamp`: The time at which the user's submission was completed, as a character string in UTC, formatted as `"%F %H:%M:%OS3 %Z"`.

See Also

`get_tutorial_info()`

`initialize_tutorial` *Initialize tutorial R Markdown extensions*

Description

One time initialization of R Markdown extensions required by the **learnr** package. This function is typically called automatically as a result of using exercises or questions.

Usage

```
initialize_tutorial()
```

Value

If not previously run, initializes knitr hooks and provides the required `rmarkdown::shiny_prerendered_chunk()`s to initialize **learnr**.

`knit_print.tutorial_question` *Knitr quiz print methods*

Description

`knitr::knit_print` methods for `question` and `quiz`

Usage

```
## S3 method for class 'tutorial_question'
knit_print(x, ...)

## S3 method for class 'tutorial_quiz'
knit_print(x, ...)
```

Arguments

`x`	An R object to be printed
`...`	Additional arguments passed to the S3 method. Currently ignored, except two optional arguments `options` and `inline`; see the references below.

`one_time` *Wrap an expression that will be executed one time in an event handler*

Description

This wraps an expression so that it will be executed one time for a tutorial, based on some condition. The first time the condition is true, the expression will be executed; after that, the expression will not be evaluated again.

The execution state is stored so that if the expression is executed, then the user quits the tutorial and then returns to it, the expression will not be executed a second time.

A common use for `one_time` is to execute an expression when a section is viewed for the first time.

Usage

```
one_time(session, cond, expr, label = deparse(substitute(cond)))
```

Arguments

`session`	A Shiny session object.
`cond`	A condition that is used as a filter. The first time the condition evaluates to true, `expr` will be evaluated; after that, `expr` will not be evaluated again.
`expr`	An expression that will be evaluated once, the first time that cond is true.
`label`	A unique identifier. This is used as an ID for the condition and expression; if two calls to `one_time()` uses the same label, there will be an ID collision and only one of them will execute. By default, cond is deparsed and used as the label.

Value

The result of evaluating `expr` (`one_time()` is intended to be called within an event handler).

Examples

```
## Not run:
# This goes in a {r context="server-start"} chunk

# The expression with message() will be executed the first time the user
# sees the section with ID "section-exercise-with-hint".
event_register_handler("section_viewed",
  function(session, event, data) {
    one_time(
      session,
      data$sectionId == "section-exercise-with-hint",
      {
        message("Seeing ", data$sectionId, " for the first time.")
      }
    )
  }
)


## End(Not run)
```

question_checkbox *Checkbox question*

Description

Creates a checkbox group tutorial quiz question. The student may select one or more checkboxes before submitting their answer.

Usage

```
question_checkbox(
  text,
  ...,
  correct = "Correct!",
  incorrect = "Incorrect",
  try_again = "Incorrect. Be sure to select every correct answer.",
  allow_retry = FALSE,
  random_answer_order = FALSE
)
```

Arguments

`text`	Question or option text
`...`	Answers created with `answer()` or `answer_fn()`, or extra parameters passed onto `question()`. Function answers do not appear in the checklist, but are checked first in the order they are specified.
`correct`	For `question`, text to print for a correct answer (defaults to "Correct!"). For `answer`, a boolean indicating whether this answer is correct.
`incorrect`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `FALSE`.
`try_again`	Text to print for an incorrect answer (defaults to "Incorrect. Be sure to select every correct answer.") when `allow_retry` is `TRUE`.
`allow_retry`	Allow retry for incorrect answers. Defaults to `FALSE`.
`random_answer_order`	Display answers in a random order.

Value

Returns a learnr question of type `"learnr_checkbox"`.

See Also

Other Interactive Questions: `question_numeric()`, `question_radio()`, `question_text()`, `quiz()`

Examples

```
question_checkbox(
  "Select all the toppings that belong on a Margherita Pizza:",
  answer("tomato", correct = TRUE),
  answer("mozzarella", correct = TRUE),
  answer("basil", correct = TRUE),
  answer("extra virgin olive oil", correct = TRUE),
  answer("pepperoni", message = "Great topping! ... just not on a Margherita Pizza"),
  answer("onions"),
  answer("bacon"),
  answer("spinach"),
  random_answer_order = TRUE,
  allow_retry = TRUE,
  try_again = "Be sure to select all four toppings!"
)

# Set up a question where there's no wrong answer. The answer options are
# always shuffled, but the answer_fn() answer is always evaluated first.
question_checkbox(
  "Which of the tidyverse packages is your favorite?",
  answer("dplyr"),
  answer("tidyr"),
  answer("ggplot2"),
  answer("tibble"),
  answer("purrr"),
  answer("stringr"),
  answer("forcats"),
  answer("readr"),
  answer_fn(function(value) {
    if (length(value) == 1) {
      correct(paste(value, "is my favorite tidyverse package, too!"))
    } else {
      correct("Yeah, I can't pick just one favorite package either.")
    }
  }),
  random_answer_order = TRUE
)
```

`question_numeric` *Number question*

Description

Creates a tutorial question asking the student to submit a number.

Usage

```
question_numeric(
  text,
  ...,
  correct = "Correct!",
  incorrect = "Incorrect",
  try_again = incorrect,
  allow_retry = FALSE,
  value = NULL,
  min = NA,
  max = NA,
  step = NA,
  options = list(),
  tolerance = 1.5e-08
)
```

Arguments

`text`	Question or option text
`...`	Answers created with `answer()` or `answer_fn()`, or extra parameters passed onto `question()`.
`correct`	For `question`, text to print for a correct answer (defaults to "Correct!"). For `answer`, a boolean indicating whether this answer is correct.
`incorrect`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `FALSE`.
`try_again`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `TRUE`.
`allow_retry`	Allow retry for incorrect answers. Defaults to `FALSE`.
`value`	Initial value.
`min`	Minimum allowed value
`max`	Maximum allowed value
`step`	Interval to use when stepping between min and max
`options`	Extra options to be stored in the question object. This is useful when using custom question types. See `sortable::question_rank()` for an example question implementation that uses the `options` parameter.
`tolerance`	Submitted values within an absolute difference less than or equal to `tolerance` will be considered equal to the answer value. Note that this tolerance is for all `answer()` values. For more specific answer value grading, use `answer_fn()` to provide your own evaluation code.

Value

Returns a learnr question of type `"learnr_numeric"`.

See Also

Other Interactive Questions: `question_checkbox()`, `question_radio()`, `question_text()`, `quiz()`

Examples

```
question_numeric(
  "What is pi rounded to 2 digits?",
  answer(3, message = "Don't forget to use the digits argument"),
  answer(3.1, message = "Too few digits"),
  answer(3.142, message = "Too many digits"),
  answer(3.14, correct = TRUE),
  allow_retry = TRUE,
  min = 3,
  max = 4,
  step = 0.01
)

question_numeric(
  "Can you think of an even number?",
  answer_fn(function(value) {
    if (value %% 2 == 0) {
      correct("even")
    } else if (value %% 2 == 1) {
      incorrect("odd")
    }
  }, label = "Is the number even?"),
  step = 1
)
```

`question_radio` *Radio question*

Description

Creates a radio button tutorial quiz question. The student can select only one radio button before submitting their answer. Note: Multiple correct answers are allowed.

Usage

```
question_radio(
  text,
  ...,
  correct = "Correct!",
  incorrect = "Incorrect",
  try_again = incorrect,
  allow_retry = FALSE,
  random_answer_order = FALSE
)
```

Arguments

`text`	Question or option text
`...`	Answers created with `answer()` or extra parameters passed onto `question()`. Function answers are ignored for radio questions because the user is required to select a single answer.
`correct`	For `question`, text to print for a correct answer (defaults to "Correct!"). For `answer`, a boolean indicating whether this answer is correct.
`incorrect`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `FALSE`.
`try_again`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `TRUE`.
`allow_retry`	Allow retry for incorrect answers. Defaults to `FALSE`.
`random_answer_order`	Display answers in a random order.

Value

Returns a learnr question of type `"learnr_radio"`.

See Also

Other Interactive Questions: `question_checkbox()`, `question_numeric()`, `question_text()`, `quiz()`

Examples

```
question_radio(
  "Pick the letter B",
  answer("A"),
  answer("B", correct = TRUE),
  answer("C"),
  answer("D"),
  allow_retry = TRUE,
  random_answer_order = TRUE
)
```

`question_text` *Text box question*

Description

Creates a tutorial question asking the student to enter text. The default text input is appropriate for short or single-line text entry. For longer text input, set the `rows` and/or `cols` argument to create a larger text area.

When used with `answer()`, the student's submission must match the answer exactly, minus whitespace trimming if enabled with `trim = TRUE`. For more complicated submission evaluation, use `answer_fn()` to provide a function that checks the student's submission. For example, you could provide a function that evaluates the user's submission using regular expressions.

Usage

```
question_text(
  text,
  ...,
  correct = "Correct!",
  incorrect = "Incorrect",
  try_again = incorrect,
  allow_retry = FALSE,
  random_answer_order = FALSE,
  placeholder = "Enter answer here...",
  trim = TRUE,
  rows = NULL,
  cols = NULL,
  options = list()
)
```

Arguments

`text`	Question or option text
`...`	Answers created with `answer()` or `answer_fn()`, or extra parameters passed onto `question()`. Answers with custom function checking
`correct`	For `question`, text to print for a correct answer (defaults to "Correct!"). For `answer`, a boolean indicating whether this answer is correct.
`incorrect`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `FALSE`.
`try_again`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `TRUE`.
`allow_retry`	Allow retry for incorrect answers. Defaults to `FALSE`.
`random_answer_order`	**[Deprecated]** Random answer order for text questions is automatically disabled to ensure that the submission is checked against each answer in the order they were provided by the author.
`placeholder`	A character string giving the user a hint as to what can be entered into the control. Internet Explorer 8 and 9 do not support this option.
`trim`	Logical to determine if whitespace before and after the answer should be removed. Defaults to `TRUE`.

rows, cols	Defines the size of the text input area in terms of the number of rows or character columns visible to the user. If either rows or cols are provided, the quiz input will use shiny::textAreaInput() for the text input, otherwise the default input element is a single-line shiny::textInput().
options	Extra options to be stored in the question object. This is useful when using custom question types. See sortable::question_rank() for an example question implementation that uses the options parameter.

Value

Returns a learnr question of type "learnr_text".

See Also

Other Interactive Questions: question_checkbox(), question_numeric(), question_radio(), quiz()

Examples

```
question_text(
  "Please enter the word 'C0rrect' below:",
  answer("correct", message = "Don't forget to capitalize"),
  answer("c0rrect", message = "Don't forget to capitalize"),
  answer("Correct", message = "Is it really an 'o'?"),
  answer("C0rrect ", message = "Make sure you do not have a trailing space"),
  answer("C0rrect", correct = TRUE),
  allow_retry = TRUE,
  trim = FALSE
)

# This question uses an answer_fn() to give a hint when we think the
# student is on the right track but hasn't found the value yet.
question_text(
  "What's the most popular programming interview question?",
  answer("fizz buzz", correct = TRUE, "That's right!"),
  answer_fn(function(value) {
    if (grepl("(fi|bu)zz", value)) {
      incorrect("You're on the right track!")
    }
  }, label = "fizz or buzz")
)
```

---

question_ui_initialize *Custom question methods*

Description

There are five methods used to define a custom question. Each S3 method should correspond to the `type` = `TYPE` supplied to the question.

- `question_ui_initialize.TYPE(question, value, ...)`
  - Determines how the question is initially displayed to the users. This should return a shiny UI object that can be displayed using shiny::renderUI. For example, in the case of `question_ui_initialize.radio`, it returns a shiny::radioButtons object. This method will be re-executed if the question is attempted again.
- `question_ui_completed.TYPE(question, ...)`
  - Determines how the question is displayed after a submission. Just like `question_ui_initialize`, this method should return an shiny UI object that can be displayed using shiny::renderUI.
- `question_is_valid.TYPE(question, value, ...)`
  - This method should return a boolean that determines if the input answer is valid. Depending on the value, this function enables and disables the submission button.
- `question_is_correct.TYPE(question, value, ...)`
  - This function should return the output of correct, incorrect, or mark_as. Each method allows for custom messages in addition to the determination of an answer being correct. See correct, incorrect, or mark_as for more details.
- `question_ui_try_again <- function(question, value, ...)`
  - Determines how the question is displayed to the users while the "Try again" screen is displayed. Usually this function will disable inputs to the question, i.e. prevent the student from changing the answer options. Similar to `question_ui_initialize`, this should should return a shiny UI object that can be displayed using shiny::renderUI.

Usage

```
question_ui_initialize(question, value, ...)

question_ui_try_again(question, value, ...)

question_ui_completed(question, value, ...)

question_is_valid(question, value, ...)

question_is_correct(question, value, ...)

## Default S3 method:
question_ui_initialize(question, value, ...)

## Default S3 method:
question_ui_try_again(question, value, ...)

## Default S3 method:
question_ui_completed(question, value, ...)
```

```
## Default S3 method:
question_is_valid(question, value, ...)

## Default S3 method:
question_is_correct(question, value, ...)
```

Arguments

`question`	question object used
`value`	user input value
`...`	future parameter expansion and custom arguments to be used in dispatched s3 methods.

Value

learnr question objects, UI elements, results or server methods.

See Also

For more information and question type extension examples, please see the **Custom Question Types** section of the `quiz_question` tutorial: `learnr::run_tutorial("quiz_question", "learnr")`.

Examples

```
q <- question(
  "Which package helps you teach programming skills?",
  answer("dplyr"),
  answer("learnr", correct = TRUE),
  answer("base")
)
question_is_correct(q, "dplyr")
question_is_correct(q, "learnr")
```

`quiz` *Tutorial quiz questions*

Description

Add interactive quiz questions to a tutorial. Each quiz question is executed within a shiny runtime to provide more flexibility in the types of questions offered. There are four default types of quiz questions:

`learnr_radio` Radio button question. This question type will only allow for a single answer submission by the user. An answer must be marked for the user to submit their answer.

`learnr_checkbox` Check box question. This question type will allow for one or more answers to be submitted by the user. At least one answer must be marked for the user to submit their answer.

`learnr_text` Text box question. This question type will allow for free form text to be submitted by the user. At least one non-whitespace character must be added for the user to submit their answer.

`learnr_numeric` Numeric question. This question type will allow for a number to be submitted by the user. At least one number must be added for the user to submit their answer.

Note, the print behavior has changed as the runtime is now Shiny based. If `questions` and `quizes` are printed in the console, the S3 structure and information will be displayed.

Usage

```
quiz(..., caption = rlang::missing_arg())

question(
  text,
  ...,
  type = c("auto", "single", "multiple", "learnr_radio", "learnr_checkbox",
    "learnr_text", "learnr_numeric"),
  correct = "Correct!",
  incorrect = "Incorrect",
  try_again = NULL,
  message = NULL,
  post_message = NULL,
  loading = NULL,
  submit_button = rlang::missing_arg(),
  try_again_button = rlang::missing_arg(),
  allow_retry = FALSE,
  random_answer_order = FALSE,
  options = list()
)
```

Arguments

`...`	One or more questions or answers
`caption`	Optional quiz caption (defaults to "Quiz")
`text`	Question or option text
`type`	Type of quiz question. Typically this can be automatically determined based on the provided answers. Pass `"radio"` to indicate that even though multiple correct answers are specified that inputs which include only one correct answer are still correct. Pass `"checkbox"` to force the use of checkboxes (as opposed to radio buttons) even though only one correct answer was provided.
`correct`	For `question`, text to print for a correct answer (defaults to "Correct!"). For `answer`, a boolean indicating whether this answer is correct.
`incorrect`	Text to print for an incorrect answer (defaults to "Incorrect") when `allow_retry` is `FALSE`.
`try_again`	Text to print for an incorrect answer when `allow_retry` is `TRUE`. Defaults to "Incorrect. Be sure to select every correct answer." for checkbox questions and "Incorrect" for non-checkbox questions.

`message`	Additional message to display along with correct/incorrect feedback. This message is always displayed after a question submission.
`post_message`	Additional message to display along with correct/incorrect feedback. If `allow_retry` is `TRUE`, this message will only be displayed after the correct submission. If `allow_retry` is `FALSE`, it will produce a second message alongside the `message` message value.
`loading`	Loading text to display as a placeholder while the question is loaded. If not provided, generic "Loading..." or placeholder elements will be displayed.
`submit_button`	Label for the submit button. Defaults to `"Submit Answer"`
`try_again_button`	Label for the try again button. Defaults to `"Submit Answer"`
`allow_retry`	Allow retry for incorrect answers. Defaults to `FALSE`.
`random_answer_order`	Display answers in a random order.
`options`	Extra options to be stored in the question object. This is useful when using custom question types. See `sortable::question_rank()` for an example question implementation that uses the `options` parameter.

Value

A learnr quiz, or collection of questions.

See Also

`random_praise()`, `random_encouragement()`

For more information and question type extension examples, please see the help documentation for question_methods and view the `question_type` tutorial: `learnr::run_tutorial("question_type", "learnr")`.

Other Interactive Questions: `question_checkbox()`, `question_numeric()`, `question_radio()`, `question_text()`

Examples

```
quiz(
  question("What number is the letter A in the alphabet?",
    answer("8"),
    answer("14"),
    answer("1", correct = TRUE),
    answer("23"),
   incorrect = "See [here](https://en.wikipedia.org/wiki/English_alphabet) and try again.",
    allow_retry = TRUE
  ),

  question("Where are you right now? (select ALL that apply)",
    answer("Planet Earth", correct = TRUE),
    answer("Pluto"),
    answer("At a computing device", correct = TRUE),
    answer("In the Milky Way", correct = TRUE),
    incorrect = paste0("Incorrect. You're on Earth, ",
                       "in the Milky Way, at a computer.")
  )
)
```

random_phrases_add *Add phrases to the bank of random phrases*

Description

Augment the random phrases available in `random_praise()` and `random_encouragement()` with phrases of your own. Note that these phrases are added to the existing phrases, rather than overwriting them.

Usage

```
random_phrases_add(language = "en", praise = NULL, encouragement = NULL)
```

Arguments

`language`	The language of the phrases to be added.
`praise, encouragement`	A vector of praising or encouraging phrases, including final punctuation.

Value

Returns the previous custom phrases invisibly when called in the global setup chunk or interactively. Otherwise, it returns a shiny pre- rendered chunk.

Usage in learnr tutorials

To add random phrases in a learnr tutorial, you can either include one or more calls to `random_phrases_add()` in your global setup chunk:

~~~
```{r setup, include = FALSE}`r ''`
library(learnr)
random_phrases_add(
  language = "en",
  praise = "Great work!",
  encouragement = "I believe in you."
)
~~~
```

Alternatively, you can call `random_phrases_add()` in a separate, standard R chunk (with `echo = FALSE`):
```

~~~
```{r setup-phrases, echo = FALSE}`r '''`
random_phrases_add(
  language = "en",
  praise = c("Great work!", "You're awesome!"),
  encouragement = c("I believe in you.", "Yes we can!")
)
```
~~~

Examples

```
random_phrases_add("demo", praise = "Great!", encouragement = "Try again.")
random_praise(language = "demo")
random_encouragement(language = "demo")
```

---

random_praise   *Random praise and encouragement*

Description

Random praises and encouragements sayings to compliment your question and quiz experience.

Usage

```
random_praise(language = NULL)

random_encouragement(language = NULL)
```

Arguments

`language`	The language for the random phrase. The currently supported languages include: `en`, `es`, `pt`, `pl`, `tr`, `de`, `emo`, and `testing` (static phrases).

Value

Character string with a random saying

Examples

```
random_praise()
random_praise()

random_encouragement()
random_encouragement()
```

run_tutorial *Run a tutorial*

Description

Run a tutorial provided by an installed R package.

Usage

```
run_tutorial(
  name = NULL,
  package = NULL,
  ...,
  shiny_args = NULL,
  clean = FALSE,
  as_rstudio_job = NULL
)
```

Arguments

`name`	Tutorial name (subdirectory within `tutorials/` directory of installed package). Alternatively, if `package` is not provided, `name` may be a path to a local tutorial R Markdown file or a local directory containing a learnr tutorial. If `package` is provided, `name` must be the tutorial name.
`package`	Name of package. If `name` is a path to the local directory containing a learnr tutorial, then `package` should not be provided.
`...`	Unused. Included for future expansion and to ensure named arguments are used.
`shiny_args`	Additional arguments to forward to `shiny::runApp`.
`clean`	When `TRUE`, the shiny prerendered HTML files are removed and the tutorial is re-rendered prior to starting the tutorial.
`as_rstudio_job`	Runs the tutorial in the background as an RStudio job. This is the default behavior when `run_tutorial()` detects that RStudio is available and can run jobs. Set to `FALSE` to disable and to run the tutorial in the current R session. When running as an RStudio job, `run_tutorial()` sets or overrides the `launch.browser` option for `shiny_args`. You can instead use the `shiny.launch.browser` global option in your current R session to set the default behavior when the tutorial is run. See the shiny options documentation for more information.

Value

Starts a Shiny server running the learnr tutorial.

See Also

`safe` and `available_tutorials`

Examples

```
# display all "learnr" tutorials
available_tutorials("learnr")

# run basic example within learnr
## Not run:
run_tutorial("hello", "learnr")

## End(Not run)
```

---

safe *Execute R code in a safe R environment*

Description

When rendering (or running) a document with R markdown, it inherits the current R Global environment. This will produce unexpected behaviors, such as poisoning the R Global environment with existing variables. By rendering the document in a new, safe R environment, a *vanilla*, rendered document is produced.

Usage

```
safe(expr, ..., show = TRUE, env = safe_env())
```

Arguments

`expr`	expression that contains all the necessary library calls to execute. Expressions within callr do not inherit the existing, loaded libraries.
`...`	parameters passed to `callr::r`
`show`	Logical that determines if output should be displayed
`env`	Environment to evaluate the document in

Details

The environment variable `LEARNR_INTERACTIVE` will be set to `"1"` or `"0"` depending on if the calling session is interactive or not.

Using `safe` should only be necessary when locally deployed.

Value

The result of `expr`.

Examples

```
## Not run:
# Direct usage
safe(run_tutorial("hello", package = "learnr"))

# Programmatic usage
library(rlang)

expr <- quote(run_tutorial("hello", package = "learnr"))
safe(!!expr)

tutorial <- "hello"
safe(run_tutorial(!!tutorial, package = "learnr"))

## End(Not run)
```

`tutorial` *Tutorial document format*

Description

Long-form tutorial which includes narrative, figures, videos, exercises, and questions.

Usage

```
tutorial(
  fig_width = 6.5,
  fig_height = 4,
  fig_retina = 2,
  fig_caption = TRUE,
  progressive = FALSE,
  allow_skip = FALSE,
  dev = "png",
  df_print = "paged",
  smart = TRUE,
  theme = "rstudio",
  highlight = "textmate",
  ace_theme = "textmate",
  mathjax = "default",
  extra_dependencies = NULL,
  css = NULL,
  includes = NULL,
  md_extensions = NULL,
  pandoc_args = NULL,
  language = "en",
  lib_dir = NULL,
  ...
)
```

Arguments

`fig_width`	Default width (in inches) for figures
`fig_height`	Default height (in inches) for figures
`fig_retina`	Scaling to perform for retina displays (defaults to 2, which currently works for all widely used retina displays). Set to `NULL` to prevent retina scaling. Note that this will always be `NULL` when `keep_md` is specified (this is because `fig_retina` relies on outputting HTML directly into the markdown document).
`fig_caption`	`TRUE` to render figures with captions
`progressive`	Display sub-topics progressively (i.e. wait until previous topics are either completed or skipped before displaying subsequent topics).
`allow_skip`	Allow users to skip sub-topics (especially useful when `progressive` is `TRUE`).
`dev`	Graphics device to use for figure output (defaults to png)
`df_print`	Method to be used for printing data frames. Valid values include "default", "kable", "tibble", and "paged". The "default" method uses a corresponding S3 method of `print`, typically `print.data.frame`. The "kable" method uses the `knitr::kable` function. The "tibble" method uses the **tibble** package to print a summary of the data frame. The "paged" method creates a paginated HTML table (note that this method is only valid for formats that produce HTML). In addition to the named methods you can also pass an arbitrary function to be used for printing data frames. You can disable the `df_print` behavior entirely by setting the option `rmarkdown.df_print` to `FALSE`. See Data frame printing section in bookdown book for examples.
`smart`	Produce typographically correct output, converting straight quotes to curly quotes, `---` to em-dashes, `--` to en-dashes, and `...` to ellipses. Deprecated in **rmarkdown** v2.2.0.
`theme`	Visual theme ("rstudio", default", "cerulean", "journal", "flatly", "readable", "spacelab", "united", "cosmo", "lumen", "paper", "sandstone", "simplex", or "yeti").
`highlight`	Syntax highlighting style. Supported styles include "default", "tango", "pygments", "kate", "monochrome", "espresso", "zenburn", "haddock", and "textmate". Pass 'NULL' to prevent syntax highlighting. Note, this value only pertains to standard rmarkdown code, not the Ace editor highlighting.
`ace_theme`	Ace theme supplied to the ace code editor for all exercises. See `learnr:::ACE_THEMES` for a list of possible values. Defaults to `"textmate"`.
`mathjax`	Include mathjax. The "default" option uses an https URL from a MathJax CDN. The "local" option uses a local version of MathJax (which is copied into the output directory). You can pass an alternate URL or pass `NULL` to exclude MathJax entirely.
`extra_dependencies`	Extra dependencies as a list of the `html_dependency` class objects typically generated by `htmltools:htmlDependency()`.
`css`	CSS and/or Sass files to include. Files with an extension of .sass or .scss are compiled to CSS via `sass::sass()`. Also, if `theme` is a `bslib::bs_theme()` object, Sass code may reference the relevant Bootstrap Sass variables, functions, mixins, etc.

`includes`	Named list of additional content to include within the document (typically created using the `includes` function).
`md_extensions`	Markdown extensions to be added or removed from the default definition of R Markdown. See the `rmarkdown_format` for additional details.
`pandoc_args`	Additional command line options to pass to pandoc
`language`	Language or custom text of the UI elements. See `vignette("multilang", package = "learnr")` for more information about available options and formatting
`lib_dir`	Directory to copy dependent HTML libraries (e.g. jquery, bootstrap, etc.) into. By default this will be the name of the document with `_files` appended to it.
`...`	Forward parameters to html_document

Value

An `rmarkdown::output_format()` for **learnr** tutorials.

Examples

```
tutorial()
```

`tutorial_html_dependency`

Tutorial HTML dependency

Description

HTML dependency for core tutorial JS and CSS. This should be included as a dependency for custom tutorial formats that wish to ensure that that `tutorial.js` and `tutorial.css` are loaded prior their own scripts and stylesheets.

Usage

```
tutorial_html_dependency()
```

Value

learnr's HTML dependencies

`tutorial_options` *Set tutorial options*

Description

Set various tutorial options that control the display and evaluation of exercises.

Usage

```
tutorial_options(
  exercise.cap = NULL,
  exercise.eval = FALSE,
  exercise.timelimit = 30,
  exercise.lines = NULL,
  exercise.pipe = NULL,
  exercise.blanks = NULL,
  exercise.checker = NULL,
  exercise.error.check.code = NULL,
  exercise.completion = TRUE,
  exercise.diagnostics = TRUE,
  exercise.startover = TRUE,
  exercise.reveal_solution = TRUE
)
```

Arguments

`exercise.cap`	Caption for exercise chunk (defaults to the engine's icon or the combination of the engine and `" code"`).
`exercise.eval`	Whether to pre-evaluate the exercise so the reader can see some default output (defaults to `FALSE`).
`exercise.timelimit`	Number of seconds to limit execution time to (defaults to `30`).
`exercise.lines`	Lines of code for exercise editor (defaults to the number of lines in the code chunk).
`exercise.pipe`	The characters to enter when the user presses the "Insert Pipe" keyboard shortcut in the exercise editor (`Ctrl/Cmd + Shift + M`). This can be set at the tutorial level or for an individual exercise. If `NULL` (default), the base R pipe (`|>`) is used when the tutorial is rendered in R >= 4.1.0, otherwise the **magrittr** pipe (`%>%`) is used.
`exercise.blanks`	A regular expression to be used to identify blanks in submitted code that the user should fill in. If `TRUE` (default), blanks are three or more underscores in a row. If `FALSE`, blank checking is not performed.
`exercise.checker`	Function used to check exercise answers (e.g., `gradethis::grade_learnr()`).

`exercise.error.check.code`	A string containing R code to use for checking code when an exercise evaluation error occurs (e.g., `"gradethis::grade_code()"`).
`exercise.completion`	Use code completion in exercise editors.
`exercise.diagnostics`	Show diagnostics in exercise editors.
`exercise.startover`	Show "Start Over" button on exercise.
`exercise.reveal_solution`	Whether to reveal the exercise solution if a solution chunk is provided.

Value

Nothing. Invisibly sets knitr::opts_chunk settings.

Examples

```
if (interactive()) {
  tutorial_options(exercise.eval = TRUE, exercise.timelimt = 10)
}
```

`tutorial_package_dependencies` *List tutorial dependencies*

Description

List the R packages required to run a particular tutorial.

Usage

```
tutorial_package_dependencies(name = NULL, package = NULL)
```

Arguments

`name`	The tutorial name. If `name` is `NULL`, then all tutorials within `package` will be searched.
`package`	The R package providing the tutorial. If `package` is `NULL`, then all tutorials will be searched.

Value

A character vector of package names that are required for execution.

Examples

```
tutorial_package_dependencies(package = "learnr")
```

Index