

Package ‘learnr2’

October 20, 2026

Title Interactive Tutorials with 'Quarto' and 'WebR'

Version 0.1.3

Description Create interactive R tutorials that run entirely in the browser using 'Quarto' <<https://quarto.org>> and 'WebR' <<https://docs.r-wasm.org/webr/latest/>> via the 'quarto-live' extension. Provides 'learnr'-style authoring conveniences -- scaffolding, exercises, hints, solutions, and quizzes -- without 'Shiny', 'R Markdown', or a server.

License MIT + file LICENSE

Copyright See file inst/COPYRIGHTS for the bundled third-party components.

Encoding UTF-8

Depends R (>= 4.0.0)

Imports fs, htmltools, httpuv, jsonlite, knitr, quarto, rmarkdown

Suggests learnr, pkgdown, pkgload, rstudioapi, testthat (>= 3.2.0), withr

VignetteBuilder quarto

Config/testthat/edition 3

Config/Needs/website rsconnect

SystemRequirements Quarto command line tool
(<https://github.com/quarto-dev/quarto-cli>).

URL <https://github.com/PPBDS/learnr2>, <https://ppbds.github.io/learnr2/>

BugReports <https://github.com/PPBDS/learnr2/issues>

Config/roxygen2/version 8.1.0

RoxygenNote 7.3.3

NeedsCompilation no

Author David Kane [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0002-6660-3934>>),
Minh Vu [aut]

Maintainer David Kane <dave.kane@gmail.com>

Repository CRAN
Date/Publication 2026-10-20 17:50:07 UTC

Contents

add_live_extension	2
answer	3
available_tutorials	4
check_tutorial	5
create_tutorial	7
download_answers_button	8
live_extension_dir	9
prerender_tutorials	9
question	11
quiz	14
render_tutorials	14
run_tutorial	16
show_file	19
student_info	21
submission_time	22
tutorial_options	23
Index	25

add_live_extension	<i>Add the 'quarto-live' extension to a project</i>
--------------------	---

Description

Copies the bundled 'quarto-live' extension into `dir/_extensions/` so that Quarto documents in `dir` can use format: `live-html`. This is the non-interactive equivalent of `quarto add r-wasm/quarto-live`.

Usage

```
add_live_extension(dir, overwrite = TRUE)
```

Arguments

- `dir` Directory of the Quarto project or document. Required: there is no default, so nothing is written anywhere you did not name. Pass `"."` for the current working directory.
- `overwrite` Overwrite an existing copy of the extension? Defaults to `TRUE`.

Value

The path to the project's `_extensions` directory, invisibly.

Examples

```
dir <- tempfile()
add_live_extension(dir)
list.files(dir, recursive = TRUE, all.files = TRUE)[1:3]
unlink(dir, recursive = TRUE)
```

answer

Define an answer choice for a quiz question

Description

Define an answer choice for a quiz question

Usage

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

Arguments

text	The answer text shown to the reader. For type = "text" questions (see question()), this is instead one acceptable response that the reader's typed input is compared against. For "reflection" and "reflection_editable" questions, the text of every correct answer is shown to the reader as the model answer – it is not compared against anything.
correct	Is this a correct answer? Defaults to FALSE. For "reflection"/"reflection_editable" questions, this instead marks text as one of the model answers to reveal.
message	Optional feedback shown when the reader picks (or types) this specific answer. Unused for "reflection"/"reflection_editable" questions.

Value

A `learnr2_answer` object for use inside [question\(\)](#).

Examples

```
answer("4", correct = TRUE)
answer("3", message = "Close, but check your arithmetic.")
```

available_tutorials	<i>List tutorials bundled with learnr2 (or any installed package)</i>
---------------------	---

Description

Scans one package – or, by default, every package installed – for a bundled `inst/tutorials/` directory, the same convention 'learnr' uses. This lets tools like the "R Tutorials" VS Code extension discover tutorials from separately-installed content packages (in the style of 'primer.tutorials') without knowing their names in advance.

Usage

```
available_tutorials(package = NULL, type = "all")
```

Arguments

package	Name of a single package to scan. Defaults to NULL, which scans every installed package, plus any package currently loaded with <code>pkgload::load_all()</code> .
type	Which authoring format to include: "quarto" (tutorials whose top-level document is a <code>.qmd</code>), "rmarkdown" (a <code>.Rmd</code>), or "all" (the default) for both.

Details

A package loaded from its source tree with `pkgload::load_all()` (as `devtools::load_all()` and `devtools::test()` do) counts as well: its tutorials are read from the source `inst/tutorials/`, so a content package's own tests see its working tree, not a stale installed copy. See the section below.

Value

A data frame with one row per tutorial and columns `package`, `name`, `title` (NA if the tutorial's `.qmd/.Rmd` has no YAML title), `format` ("quarto" or "rmarkdown"), `path` (the installed `.qmd/.Rmd` file; NA if the directory has neither), and `package_dependencies` (a list column: for each tutorial, the character vector of R packages that must be installed locally before it can run). `name` can be passed to [run_tutorial\(\)](#); `path` to [render_tutorials\(\)](#) and [check_tutorial\(\)](#).

Classic learnr tutorials

A "quarto" tutorial's exercises run in the reader's browser via WebR, so it needs no R packages installed locally beyond learnr2 itself and its `package_dependencies` is `character(0)`. An "rmarkdown" tutorial is a classic 'learnr' tutorial (an `.Rmd` with `runtime: shiny_prerendered`), which runs as a Shiny app in the local R session. Its `package_dependencies` are whatever 'learnr' finds by scanning the tutorial's directory (`learnr::available_tutorials()`), which always includes 'learnr' itself. If 'learnr' is not installed there is nothing to ask, and such a tutorial could not run anyway, so the entry is NA.

Packages loaded with pkgload

`system.file()` resolves against the *installed* copy of a package, so a content package under development used to be invisible here (or, worse, silently read from an old install) when its own tests ran under `devtools::test()`: `'pkgload'` redirects `system.file()` only for code inside the package being developed, not for `learnr2`'s calls. This function and `run_tutorial()` therefore check whether package is a namespace loaded by `pkgload::load_all()` and, if so, read its tutorials from the source tree's `inst/tutorials/` directly. Nothing changes for installed packages, and `'pkgload'` itself is not required.

Examples

```
# Qualified with learnr2:: because the learnr package exports a function of
# the same name; this guarantees learnr2's version is used even if learnr is
# also attached and masks it on the search path.
learnr2::available_tutorials(package = "learnr2")
learnr2::available_tutorials(package = "learnr2", type = "quarto")
```

check_tutorial

Check that a tutorial has the recommended components

Description

Static checks on a tutorial's `.qmd` source for the mistakes that look fine in the file and only show up later – in the rendered page, in a reader's browser, or when a reader's saved progress silently resets. Each one is a rule from this package's authoring guide; this function encodes them so a content package can enforce them in its tests, the way `tutorial.helpers::check_tutorial_defaults()` did for `'learnr'` tutorials. Pair it with `render_tutorials()`: rendering proves the document builds, these checks catch what a successful render does not.

Usage

```
check_tutorial(paths, skip = NULL, error = TRUE)
```

Arguments

<code>paths</code>	Character vector of tutorials to check: paths to <code>.qmd</code> files, or to directories containing them, as for <code>render_tutorials()</code> .
<code>skip</code>	Character vector of check names (see "Checks") to leave out, e.g. "minutes" for a tutorial that deliberately has no "how many minutes" question.
<code>error</code>	If TRUE (the default), stop with an error listing every problem found. If FALSE, just return them.

Value

A data frame of problems with columns `path`, `check`, and `message`, invisibly. It has zero rows when every check passed.

Checks

Each check has a name, used in skip:

- `format` – the YAML header has `format: live-html`.
- `engine` – the YAML header has `engine: knitr`.
- `include` – the document includes the 'quarto-live' runtime partial, `{{< include _extensions/r-wasm/live/_knitr`. Without it no `{webr}` cell works.
- `gradethis` – if any `{webr}` cell has `check: true`, the document also includes `_extensions/r-wasm/live/_gradethis`.
- `student_info` – an `{r}` chunk calls `learnr2::student_info()`.
- `minutes` – an `{r}` chunk has the "how many minutes" question, i.e. a `learnr2::question()` with `validate = "integer"`.
- `download` – an `{r}` chunk calls `learnr2::download_answers_button()`.
- `labels` – every `{r}` and `{webr}` chunk has its own `#| label:` line, and no two chunks share a label. A question's chunk label is the key its saved answer is stored under, so a missing or duplicated label loses or merges readers' progress.
- `echo` – every `{r}` chunk that renders a learnr2 widget (`question()`, `quiz()`, `student_info()`, `download_answers_button()`, `tutorial_options()`) has `#| echo: false`, so the reader sees the widget, not the R code that produced it.
- `persist` – every `{webr}` exercise cell (one with `#| exercise:` that is not a setup, check, solution, or hint cell) has `#| persist: true`, without which the reader's code is never saved and never appears in a download.
- `solution` – every exercise that has a `check: true` cell also has a `solution: true` cell. A `.solution` div is not enough: the grader fails with "No solution code was found".
- `packages` – every non-base package a `{webr}` cell uses (via `library()`, `require()`, or `pkg::`) is listed under `webr: packages:` in the YAML header, the only place WebR learns what to install. `gradethis` is exempt, since the `_gradethis.qmd` include provides it.

Examples

```
qmd <- create_tutorial("checked", dir = tempfile(), open = FALSE)
check_tutorial(qmd)

# Break the template, then see the problems instead of an error.
lines <- readLines(qmd)
writeLines(lines[!grepl("^#\\| echo: false", lines)], qmd)
problems <- check_tutorial(qmd, error = FALSE)
problems[, c("check", "message")]

unlink(dirname(dirname(qmd)), recursive = TRUE)
```

create_tutorial	Create a new learnr2 tutorial
-----------------	-------------------------------

Description

Scaffolds a new interactive tutorial: a directory containing a starter .qmd document wired up for format: live-html, with the bundled 'quarto-live' extension copied alongside it so it renders out of the box. The starter document opens with `student_info()` and ends with a "how many minutes did this take" question plus `download_answers_button()`, so every new tutorial collects name/email (and an optional ID) and lets the reader turn in their answers by default; delete either section if a given tutorial doesn't need it.

Usage

```
create_tutorial(name, dir, title = name, open = interactive())
```

Arguments

name	Name of the tutorial. Used for the directory and the .qmd file name, so it should be a valid file name (e.g. "my-tutorial").
dir	Parent directory in which to create the tutorial directory. Required: there is no default, so nothing is written anywhere you did not name. Pass "." for the current working directory.
title	Human-readable title placed in the document's YAML header. Defaults to name.
open	Whether to open the new .qmd file in an interactive session. Defaults to TRUE when interactive.

Value

The path to the created .qmd file, invisibly.

Examples

```
# Scaffold into a temporary directory, without opening the new file.
dir <- tempfile()
qmd <- create_tutorial("my-first-tutorial", dir = dir, open = FALSE)
list.files(dirname(qmd), all.files = TRUE, no.. = TRUE)
unlink(dir, recursive = TRUE)
```

download_answers_button

Add a "download my answers" button

Description

Adds a button that, when clicked, gathers every `question()` and `student_info()` answer currently on the page – each already saved to the browser's `localStorage` when the reader submitted it – into a single readable JSON file and downloads it. Only submitted answers count: an answer reopened with "Edit" and not resubmitted is reported as its last submitted version, and the download is blocked until every `student_info()` form is submitted. This happens entirely in the reader's browser; there is no server to submit to, so this is meant for a reader to save and turn in themselves (e.g. attach to an email or upload to an LMS).

Usage

```
download_answers_button(
  filename_prefix = "learnr2-answers",
  label = "Download My Answers"
)
```

Arguments

filename_prefix	Prefix for the downloaded file's name. Defaults to "learnr2-answers".
label	Button label. Defaults to "Download My Answers".

Details

The download's answers array is one flat list: every `question()` plus every `{webr}` code exercise, each as a `{ id, answer }` pair keyed by the widget's (or exercise's) stable id. A question never submitted has `answer: null`; a choice question's answer is an array of the picked options; an image-paste reflection's answer is the pasted screenshot as a PNG data-URL string.

A `{webr}` exercise appears only under the condition quarto-live itself requires to keep a record of the reader's code at all: `#| persist: true` (already the convention every bundled tutorial follows). An exercise without `persist: true` has no saved copy of the reader's code anywhere – learnr2 included – so it can't appear in the download; this is a structural limit of quarto-live's own editor, not something `download_answers_button()` chooses to skip.

The download also carries a metadata block (timezone, browser info, and a random per-device id persisted across the reader's visits) and a time field: the moment the reader clicked "Download", lightly obfuscated so the raw timestamp isn't readable or hand-editable in the file. Recover it with `submission_time()`. Nothing else is hashed or signed – a determined reader can still edit their answers; this only keeps the submission time honest at a glance.

Value

A `learnr2_download_button` object, printed as an interactive HTML button.

Examples

```
download_answers_button()
```

live_extension_dir	<i>Path to the bundled 'quarto-live' extension</i>
--------------------	--

Description

learnr2 ships a copy of the 'quarto-live' Quarto extension so that authors do not need to run quarto add themselves. This returns the path, inside the installed package, to the _extensions directory that contains it.

Usage

```
live_extension_dir()
```

Value

A length-one character path to the bundled _extensions directory.

Examples

```
live_extension_dir()
```

prerender_tutorials	<i>Render every installed Quarto tutorial into the cache</i>
---------------------	--

Description

Fills `run_tutorial()`'s render cache ahead of time, so that the first launch of each tutorial is as fast as every later one. Intended for environment builds – a container image, a Codespaces prebuild, a lab machine setup – where the render cost can be paid once for everyone, before any student is waiting. Tutorials whose cached render is already current are skipped. Classic "rmarkdown" tutorials are not rendered: they are Shiny apps and have nothing to cache.

Usage

```
prerender_tutorials(
  package = NULL,
  output_dir = tools::R_user_dir("learnr2", "cache"),
  refresh = FALSE
)
```

Arguments

package	Name of a single package whose tutorials to render. Defaults to NULL, which renders the Quarto tutorials of every installed package.
output_dir	Root of the render cache for "quarto" tutorials (ignored for an "rmarkdown" one). Each tutorial is rendered into output_dir/<package>/<name>/. Defaults to a persistent per-user directory (see <code>tools::R_user_dir()</code>), <i>not</i> <code>tempfile()</code> : a persistent location is what makes the render cache (below) work at all, and R deletes its session temp directory as soon as the R process exits, which races with the browser actually loading the page when <code>open = TRUE</code> is used from <code>Rscript</code> .
refresh	Re-render a "quarto" tutorial even if the cached render is current. Defaults to FALSE.

Value

A data frame, invisibly, with one row per Quarto tutorial and columns package, name, html (the rendered file) and rendered (TRUE if it was rendered on this call, FALSE if the cached copy was already current).

Where the cache must live

The default `output_dir` is a per-user directory (see `tools::R_user_dir()`), so pre-rendering only helps if it runs *as the user who will later run the tutorials*, with the same HOME – in a Dockerfile, as the image's runtime user, not root – or with an `output_dir` both can see. The cache is keyed on the installed tutorial files and the `learnr2` version, so it stays valid for as long as those are the ones baked in alongside it.

See Also

`run_tutorial()`, whose "Render cache" section explains what makes a cached render current.

Examples

```
# One package, into a temporary directory rather than the user cache.
# Called with no arguments, prerender_tutorials() renders every installed
# package's tutorials into the default per-user cache. Runs only
# interactively, with the Quarto command line tool installed: a full
# tutorial takes several seconds to render.
if (interactive() && !is.null(quarto::quarto_path())) {
  out <- tempfile()
  prerender_tutorials(package = "learnr2", output_dir = out)
  unlink(out, recursive = TRUE)
}
```

question

*Create a quiz question***Description**

A learnr-style quiz question, rendered as a small self-contained, client-side interactive widget — no Shiny or R server required. Supports single-choice ("radio"), multiple-choice ("checkbox"), and free-text questions, graded entirely in the reader's browser.

Usage

```
question(
  text,
  ...,
  type = c("auto", "single", "multiple", "text", "reflection", "reflection_editable"),
  correct = "Correct!",
  incorrect = "Incorrect.",
  allow_retry = FALSE,
  random_answer_order = FALSE,
  submit_button = "Submit Answer",
  try_again_button = "Try Again",
  edit_button = "Edit Answer",
  id = NULL,
  allow_image = FALSE,
  show_text = TRUE,
  validate = c("none", "integer")
)
```

Arguments

text	Question prompt.
...	One or more <code>answer()</code> objects. Optional for "reflection"/"reflection_editable" questions (see type); required, with at least one marked correct, for every other type.
type	Question type. "auto" (the default) picks "single" when exactly one answer is marked correct, and "multiple" otherwise. Other types: "text" — a free-text question, graded by comparing (trimmed, whitespace-collapsed, case-insensitive) the reader's response against the <code>answer()</code> text(s). "reflection" — an ungraded free-response question. After submitting, the reader sees the model answer (the correct <code>answer()</code> text(s)) and their own response is locked. If no <code>answer()</code> is marked correct — including passing none at all, e.g. for a genuinely open-ended prompt like "how many minutes did this take?" with no right answer to demonstrate — nothing is revealed; the reader's response is still saved and locked exactly the same.

	• "reflection_editable" – like "reflection", but a submitted response can be reopened. Submit locks it behind an "Edit" button; Edit reopens it, with a note that the change isn't saved until the next Submit. The saved answer is always the last submitted one. Meant for answers with nothing to copy, such as a minutes-spent count: an ordinary "reflection" stays locked for good, so a reader can't reopen it and paste in the model answer they were just shown.
correct	Message shown when the reader answers correctly. Unused for "reflection"/"reflection_editable" questions.
incorrect	Message shown when the reader answers incorrectly. Unused for "reflection"/"reflection_editable" questions.
allow_retry	Allow the reader to try again after an incorrect answer? Defaults to FALSE. Unused for "reflection"/"reflection_editable" questions.
random_answer_order	Shuffle answer order each time the page loads? Defaults to FALSE. Only applies to "single"/"multiple" questions.
submit_button, try_again_button	Button labels.
edit_button	Button label shown while a submitted "reflection_editable" question is locked; clicking it reopens the answer for editing. Ignored for every other type.
id	Stable identifier for this question: it keys the reader's saved answer (see "Progress persistence" below) and is the id the question appears under in a download_answers_button() submission. Defaults to the label of the {r} chunk the question() call sits in — which, following tutorial.helpers' section-header-N chunk-naming convention, is already a unique, readable identifier (## Quiz questions -> quiz-questions-1, quiz-questions-2, ...). Falls back to a slug of text when there is no usable chunk label, e.g. when printing a question() at the console. Pass id explicitly to pin it regardless of where the call sits.
allow_image	For "reflection"/"reflection_editable" questions, let the reader paste an image (e.g. a screenshot) from their clipboard, alongside their typed response – not a file upload, just Ctrl+V/Cmd+V into the question. Defaults to FALSE. Ignored for other question types. Accepts PNG, JPEG, GIF, WebP, or BMP (whatever the reader's platform actually put on the clipboard – this varies, and isn't guaranteed to be PNG just because they took a screenshot) and re-encodes it as PNG before storing it, so what ends up saved is always PNG regardless of the source format. Capped at 2MB.
show_text	Show text as the question's prompt inside the widget? Defaults to TRUE. Set it to FALSE when the prompt is written as ordinary text on the page, just above the question, and should not be repeated inside the box. text is still required: it is kept (and read by screen readers, via visually hidden text) and is what the reader is reminded of if they try to download their answers without submitting this question.
validate	Client-side format check applied before the reader can submit a "text", "reflection", or "reflection_editable" answer. "none" (the default) accepts anything. "integer" requires the typed response to be a whole number (optionally signed, e.g. -3) – useful for a question like "how many minutes did this take?" where

any honest number is fine, but free-form prose is not; see the "reflection" example below. Ignored (forced to "none") for "single"/"multiple" questions.

Value

A `learnr2_question` object. Printed as an interactive HTML widget in a rendered Quarto document; at the console it opens a browser preview in an interactive session and prints the HTML source otherwise.

Progress persistence

Once a reader submits an answer, it is saved in the browser's `localStorage` (keyed by page URL and id) and restored on the next visit. Because the default id is the enclosing `{r}` chunk's label, renaming that chunk (or moving the question into a different one) resets any saved answers for it — but editing only the question's wording does not. Pass id explicitly to pin it.

`localStorage` is written straight to disk, not held only in memory, so this survives closing and re-opening the browser, and restarting the computer — confirmed with automated tests (`tests/js/persistence.spec.js`) that fully quit and relaunch a real browser against the same profile, for both a `file://` tutorial and one served over HTTP. Two things it does *not* survive, by browser design rather than anything `learnr2` controls: private/incognito windows (their storage is wiped when the window closes) and the exact page URL changing — a tutorial opened from a different server, port or path starts fresh. That is why `run_tutorial()` always serves on one fixed port and gives every tutorial its own stable path, `/<package>/<name>/`: the address, and so the saved answers, are the same on every launch, and no two tutorials share them.

Every page also gets a "Start Over" button (nothing to opt into — it's added by the same JavaScript that renders `question()/student_info()`). It sits at the bottom of Quarto's TOC sidebar when the page shows one, and otherwise — a tutorial rendered with `toc: false`, or any tutorial on a screen too narrow for Quarto to show the sidebar — at the top of the tutorial, directly under the title, before the first section. Clicking it, after a confirmation prompt, clears every `question()/student_info()` answer *and* every `{webr}` exercise's persisted code (`persist: true`) for this page on this device, then reloads — a clean slate, without needing to know that both live under different `localStorage` key prefixes. It deliberately leaves alone the random per-device id `download_answers_button()` embeds in a submission's metadata, since that identifies this browser across every tutorial and visit, not this one tutorial's progress.

Examples

```
question(
  "What is 6 times 7?",
  answer("42", correct = TRUE),
  answer("36"),
  answer("48"),
  allow_retry = TRUE
)

# No answer() at all -- a genuinely open-ended prompt with nothing to
# reveal after the reader submits.
question(
  "How many minutes, approximately, did this take?",
  type = "reflection_editable",
```

```

    validate = "integer"
  )

```

quiz
Group questions into a quiz

Description

Group questions into a quiz

Usage

```
quiz(..., caption = "Quiz")
```

Arguments

<code>...</code>	One or more question() objects.
<code>caption</code>	Heading shown above the questions.

Value

A `learnr2_quiz` object, printed as a set of interactive widgets.

Examples

```

quiz(
  caption = "Arithmetic",
  question(
    "What is 2 + 2?",
    answer("4", correct = TRUE),
    answer("22")
  )
)

```

render_tutorials
Render tutorials, as a test that they build

Description

Renders each tutorial with Quarto, exactly the way [run_tutorial\(\)](#) and the package's own GitHub Pages publishing do: the tutorial's directory is copied under `output_dir`, the bundled 'quarto-live' extension is added next to the copy with [add_live_extension\(\)](#), and the copy is rendered with [quarto::quarto_render\(\)](#). A tutorial that fails to render stops with an error naming it. This is the `learnr2` counterpart of `tutorial.helpers::knit_tutorials()`: "testing" a tutorial means confirming it renders without error, which catches a large class of mistakes that look fine in the `.qmd` source (see also [check_tutorial\(\)](#) for the static checks that complement it).

Usage

```
render_tutorials(paths, output_dir = tempfile("learnr2-render-"), quiet = TRUE)
```

Arguments

<code>paths</code>	Character vector of tutorials to render: paths to .qmd files, or to the directories that contain them (the first .qmd in each directory is used). <code>available_tutorials()</code> 's path column and <code>create_tutorial()</code> 's return value are both accepted directly.
<code>output_dir</code>	Directory to render into. Each tutorial is copied to <code>output_dir/<tutorial-directory-name>/</code> first, so the installed source is never written to. Defaults to a fresh directory under the session's temporary directory, which is also the only place CRAN permits a test to write.
<code>quiet</code>	Passed to <code>quarto::quarto_render()</code> . Defaults to TRUE; set FALSE to see Quarto's own progress output, e.g. in CI logs.

Details

Nothing here boots WebR or exercises the rendered page in a browser – that happens in the reader's browser, not at render time – so a successful render proves the document builds, not that every exercise behaves. Each tutorial's render time is reported, since a slow one is usually the first sign of something that will also be slow for readers.

Value

A character vector of paths to the rendered .html files, named by tutorial (the directory name), invisibly.

In a content package's tests

A package of tutorials can check every one of them from `tests/testthat/test-tutorials.R`:

```
tutorials <- available_tutorials(package = "my.tutorials")
render_tutorials(tutorials$path)
check_tutorial(tutorials$path)
```

Guard that test with `testthat::skip_on_cran()` and `testthat::skip_if(is.null(quarto::quarto_path()))`, since it needs the Quarto command line tool.

Examples

```
# Scaffold a tutorial, then render it the way a test would. Needs the
# Quarto command line tool, so this is skipped where it isn't installed.
if (!is.null(quarto::quarto_path())) {
  dir <- tempfile()
  qmd <- create_tutorial("render-me", dir = dir, open = FALSE)
  html <- render_tutorials(qmd)
  file.exists(html)
  unlink(c(dir, dirname(html)), recursive = TRUE)
}
```

run_tutorial

*Run a bundled tutorial***Description**

Runs a tutorial bundled with an installed package, whichever of the two formats [available_tutorials\(\)](#) reports it is. A "quarto" tutorial (learnr2's own format) is rendered into a per-user cache – or reused from it, if it was rendered before and nothing has changed – and, when open is TRUE, served to a browser. An "rmarkdown" tutorial – a classic 'learnr' tutorial – is handed to `learnr::run_tutorial()`, so a tool built on learnr2 (such as the "R Tutorials" VS Code extension) can run both kinds through this one function and depend only on learnr2.

Usage

```
run_tutorial(
  name = NULL,
  package = "learnr2",
  output_dir = tools::R_user_dir("learnr2", "cache"),
  open = interactive(),
  refresh = FALSE
)
```

Arguments

name	Name of the tutorial to run. See available_tutorials() . If NULL, the available tutorials in package are listed.
package	Name of the package the tutorial is bundled with. Defaults to "learnr2"; set this to run a tutorial from another installed package (e.g. a 'primer.tutorials'-style content package), or from one loaded with <code>pkgload::load_all()</code> (see available_tutorials()).
output_dir	Root of the render cache for "quarto" tutorials (ignored for an "rmarkdown" one). Each tutorial is rendered into <code>output_dir/<package>/<name>/</code> . Defaults to a persistent per-user directory (see tools::R_user_dir()), <i>not</i> tempfile() : a persistent location is what makes the render cache (below) work at all, and R deletes its session temp directory as soon as the R process exits, which races with the browser actually loading the page when <code>open = TRUE</code> is used from Rscript.
open	Whether to serve the tutorial and open it in a browser. Defaults to TRUE when interactive. When TRUE, this call blocks (like httpuv::runStaticServer() or shiny::runApp()) until you interrupt it (Ctrl+C, or the console's Stop button) – see the sections below for why. When FALSE, a "quarto" tutorial is rendered (or found in the cache) and its path returned without serving or blocking; an "rmarkdown" tutorial has no render-only mode (it is a Shiny app), so <code>open = FALSE</code> is an error for one. Note that under Rscript the default is FALSE, so pass <code>open = TRUE</code> explicitly there.
refresh	Re-render a "quarto" tutorial even if the cached render is current. Defaults to FALSE.

Value

Path to the rendered HTML file for a "quarto" tutorial, or to the .Rmd source for an "rmarkdown" one, invisibly.

Render cache

Rendering a Quarto tutorial takes several seconds on a laptop and well over ten on a small cloud machine, and nothing about an installed tutorial changes between one launch and the next. So the render is cached: alongside the HTML in `output_dir/<package>/<name>/`, a stamp file records the `learnr2` version and a fingerprint of every file in the installed tutorial directory. On the next launch, if both still match, the cached HTML is served at once. Reinstalling the tutorial's package or upgrading `learnr2` invalidates the cache; so does `refresh = TRUE`. Before any re-render the old copy is deleted, so files a previous version of the tutorial had and the current one does not cannot linger.

`prerender_tutorials()` fills the cache for every installed Quarto tutorial ahead of time – for instance while building a container image – so that even a student's first launch is instant.

Serving, the fixed port, and saved answers

A reader's answers and typed exercise code are saved in the browser's `localStorage`, keyed by the page URL (see "Progress persistence" in `question()`). For them to be found again tomorrow, the URL has to be the same tomorrow. So `learnr2` always serves on one fixed port, 7446 (`getOption("learnr2.port")` overrides it), and never falls back to a random one; and it serves the whole cache root rather than a single tutorial, so every tutorial has its own stable address, `http://127.0.0.1:7446/<package>/<name>/`, and no two tutorials share saved state or get wiped together by one tutorial's "Start Over".

Because the server serves the whole cache, one server is enough for any number of tutorials. If a `learnr2` server is already running on the port (say, in another terminal), this function renders into the cache if needed and simply opens the tutorial's address in the browser, without starting a second server or blocking. If the port is held by something other than a `learnr2` server, it stops with an error rather than silently serving somewhere the browser has no saved answers for.

The cache root itself (`http://127.0.0.1:7446/`) is a small page that forwards to the tutorial launched most recently and lists every other rendered tutorial in the cache. That root is what a tool that only knows the port opens – notably the "open in browser" notification VS Code and GitHub Codespaces show when they detect a new local port – so landing on it has to reach the tutorial rather than a 404.

Running in GitHub Codespaces (or another remote container)

The server runs inside the container, so `http://127.0.0.1:7446/...` is only reachable from inside it; in the browser on your own machine that address refuses to connect. Codespaces forwards the port to a public address, `https://<codespace>-7446.app.github.dev/`, and this function detects a codespace (the `CODESPACE_NAME` and `GITHUB_CODESPACES_PORT_FORWARDING_DOMAIN` environment variables) and prints the tutorial's forwarded address. The browser is still opened on the local address, through the helper VS Code puts in `BROWSER`, which forwards the port as part of opening it, just as clicking a localhost link in the terminal does; opening the forwarded address directly can race the port forwarding and show an empty 404 until reloaded. The forwarded root page forwards to the tutorial too, so the "Open in Browser" button on the port notification lands in

the right place. Saved answers are keyed by page URL, so they live under the forwarded address and are found again as long as the codespace keeps its name.

Why this blocks and serves over local HTTP instead of opening the file directly

Every {webr} exercise compiles down to Observable JS (OJS), which Quarto's runtime loads via ES modules – and browsers refuse to load ES modules from a `file://` URL. Opening the rendered HTML directly (e.g. `utils::browseURL()` on the local path, or double-clicking the file) hits this and shows an "OJS runtime" error, even though plain `question()/student_info()` widgets (not OJS-based) work fine over `file://`.

An earlier version of this function used `quarto::quarto_preview()` to both render and serve the tutorial via a background daemon process, on the theory that it would keep running after `run_tutorial()` returned. In practice that daemon did not reliably stay alive (confirmed: it could exit within seconds, even with the calling R session still running and pumping its event loop), silently leaving you back at a `file://` URL with no server behind it. This function now renders with the same one-shot `quarto::quarto_render()` call the package's own pkgdown publishing script uses, and serves the result with httpuv's static server – an in-process server with no separate daemon to lose track of. Its trade-off is that it blocks the caller while serving, matching how the original 'learnr' package's `run_tutorial()` (built on a blocking Shiny app) behaved – stop the server to get your prompt back.

Classic learnr tutorials

Many existing content packages (those built on 'tutorial.helpers', for instance) bundle classic 'learnr' tutorials: .Rmd files with `runtime: shiny_prerendered` that run as a Shiny app in the local R session. `learnr2` cannot run those itself – the Shiny machinery lives in 'learnr' – so for an "rmarkdown" tutorial this function calls `learnr::run_tutorial(name, package = package)`, which blocks while the app runs just as the "quarto" path blocks while serving.

'learnr' is only a suggested dependency of `learnr2`, not a required one, because a package that bundles classic learnr tutorials already depends on 'learnr' itself (directly, or via 'tutorial.helpers'). So whenever an "rmarkdown" tutorial is installed, 'learnr' is too; this function only errors with an install hint if that invariant is somehow broken.

See Also

`prerender_tutorials()` to fill the render cache in advance.

Examples

```
# With no `name`, just lists the tutorials that can be run.
run_tutorial()

# Render without serving (open = FALSE), into a temporary directory
# rather than the user cache. Calling it again would reuse this render;
# pass refresh = TRUE to force a new one. Runs only interactively, with the
# Quarto command line tool installed: a full tutorial takes several
# seconds to render.
if (interactive() && !is.null(quarto::quarto_path())) {
  out <- tempfile()
  html <- run_tutorial("hello-learnr2", output_dir = out, open = FALSE)
```

```

    file.exists(html)
    unlink(out, recursive = TRUE)
}

# Not run: with open = TRUE, starts a local web server that blocks the
# session until interrupted.
## Not run:
run_tutorial("hello-learnr2")

# A classic learnr tutorial from a content package is handed to learnr.
run_tutorial("hello", package = "learnr", open = TRUE)

## End(Not run)

```

show_file	<i>Display all or part of a text file</i>
-----------	---

Description

Prints a text file – or the part of it an author most often wants to see – to the console. By default (`chunk = "auto"`), a file containing fenced code chunks (a `.qmd` or `.Rmd`) shows just its *last* code chunk, which is by far the most common use ("show me the code you just wrote"), while a file with no code chunks (a `.gitignore`, `.yaml`, `.R`, ...) is shown whole. Everything else is opt-in: a row range, a regular-expression filter, all chunks, one chunk picked out by its label, or the YAML header.

Usage

```
show_file(path, start = 1, end = NULL, pattern = NULL, chunk = "auto")
```

Arguments

path	Path to the text file.
start	Integer: the first row (inclusive) to show. Default is 1. If negative, show the last <code>abs(start)</code> lines of the file instead. If 0, show the entire file. Supplying <code>start</code> switches <code>chunk = "auto"</code> off.
end	Integer: the last row (inclusive) to show. Default is the last row of the file. Supplying <code>end</code> switches <code>chunk = "auto"</code> off.
pattern	A regular expression; only rows matching it are shown. Default is <code>NULL</code> (no filtering). Applied to the whole-file (<code>start == 0</code>), last-lines (<code>start < 0</code>), and row-range cases; ignored when <code>chunk</code> selects code chunks or the YAML header. Supplying <code>pattern</code> switches <code>chunk = "auto"</code> off.
chunk	What to show. One of: "auto" (the default): the last code chunk if the file has any and no row arguments were given; otherwise the whole file. "None": no chunk processing; show rows (the whole file by default).

- "All": the code of every chunk, separated by blank lines.
- "Last": the code of the last chunk only.
- "YAML": the YAML header, without its --- delimiters.
- Any other string: the code of the chunk with that label. It is an error if no chunk has that label.

Details

The arguments are resolved in a fixed order. `chunk = "YAML"` is handled first; then `start == 0` (whole file); then `start < 0` (the last `abs(start)` lines); then `chunk` when it selects code chunks ("All", "Last", or a label); and finally the `start/end` row range. The pattern filter applies within the whole-file, last-lines, and row-range cases, and is ignored when `chunk` selects code chunks or the YAML header.

`chunk = "auto"` resolves to "Last" only when the file contains at least one fenced code chunk *and* none of `start`, `end`, or `pattern` were supplied. Passing any of those three is taken as a request to see rows, not chunks, so e.g. `show_file("analysis.qmd", end = 4)` always means the first four lines, and `show_file("analysis.qmd", pattern = "library")` always searches the whole file.

A chunk's label is read from any of the usual places: the positional label in the fence header (``{r setup}``), a `label =` option in the header (``{r, label = "setup"}``), or a `#| label: setup` line inside the chunk. Chunks of any language are recognized (`r`, `python`, `bash`, ...), not just `R`.

Rendered output that VS Code's interactive chunk execution caches back into a `.qmd` file (a pagetable HTML widget) is stripped before anything is printed, so a chunk shows only its source.

Value

Called for its side effect of printing to the console; returns `NULL`, invisibly. Prints nothing if no rows match pattern, or if the selected chunk is empty. A file that is empty, or contains only blank lines, prints `File is empty`.

Examples

```
# A small Quarto document to look at:
qmd <- tempfile(fileext = ".qmd")
writeLines(c(
  "----",
  "title: \"An example\"",
  "----",
  "Some prose, with an example in it.",
  "````{r setup}",
  "library(dplyr)",
  "x <- 1:10",
  "````",
  "More prose.",
  "````{r}",
  "#| label: summary",
  "mean(x)",
  "````"
), qmd)
```

```

# The default shows the last code chunk, the most common thing to want
show_file(qmd)

# Every chunk, or one picked out by its label
show_file(qmd, chunk = "All")
show_file(qmd, chunk = "setup")
show_file(qmd, chunk = "summary")

# The YAML header, without its delimiters
show_file(qmd, chunk = "YAML")

# Rows instead of chunks: the whole file, a range, a filter, the tail
show_file(qmd, chunk = "None")
show_file(qmd, start = 4, end = 8)
show_file(qmd, pattern = "example")
show_file(qmd, start = -3)

# A file with no code chunks is shown whole by default
gitignore <- tempfile(fileext = ".gitignore")
writeLines(c(".Rproj.user", ".Rhistory", "*.html"), gitignore)
show_file(gitignore)

unlink(c(qmd, gitignore))

```

student_info

Collect student identifying information

Description

Adds a small, ungraded form for the reader to fill in identifying information before starting a tutorial: name and email required, an ID optional, by default. Unlike [question\(\)](#), nothing here is graded and there is no model answer to reveal.

Usage

```

student_info(
  fields = c(name = "Name:", email = "Email:", id =
    "ID (if requested by your instructor):"),
  required = intersect(c("name", "email"), names(fields)),
  id = "student-info",
  submit_button = "Submit",
  edit_button = "Edit"
)

```

Arguments

fields	A named character vector of field key/label pairs to collect. Defaults to name, email, and an optional ID, matching 'tutorial.helpers's info_section.Rmd. A field named "email" is additionally checked for an "@" character, regardless of whether it is required.
--------	---

required	Character vector of keys (from <code>fields</code>) the reader must fill in. Defaults to whichever of "name" and "email" are actually present in <code>fields</code> , so ID is optional by default and supplying custom <code>fields</code> does not require also supplying <code>required</code> . A required field left blank, or an "email" field missing an "@", is flagged inline (on blur, and again when the button is clicked) and the form can't be submitted until it's fixed. Passing a key that is not in <code>fields</code> is an error.
id	Stable identifier used to key the saved values in <code>localStorage</code> . Defaults to "student-info"; change it if a single tutorial embeds more than one <code>student_info()</code> form.
submit_button	Button label while the form is open for editing.
edit_button	Button label while the form is locked on a submission. Clicking it reopens the form.

Details

The form has two states. While *editing*, the fields are open and the button reads "Submit". A valid Submit saves the entry, locks the fields, and switches the button to "Edit". Clicking Edit only reopens the fields and switches the button back to "Submit", with a note that the change isn't saved until the next Submit. So the button always tells the reader whether what they see is what was received. Typing is kept as a draft in the browser's `localStorage`, so a reload mid-edit loses nothing, but a draft is never the answer: `download_answers_button()` reports only the last submitted entry, and is blocked until the form is submitted. A "reflection_editable" `question()` works the same way. Pair with `download_answers_button()` so a reader can turn their work in.

Value

A `learnr2_info` object, printed as an interactive HTML form.

Examples

```
student_info()
student_info(fields = c(name = "Full name:", section = "Section:"), required = "name")
```

submission_time	<i>Recover a submission's download time</i>
-----------------	---

Description

`download_answers_button()` writes the moment the reader clicked "Download" into the JSON as a time field, lightly obfuscated: base-36 of the epoch second run through a fixed multiply-and-offset. It is *not* encrypted — the scheme is public, in `inst/extdata/quiz/quiz.js` — just enough that the raw timestamp isn't sitting in the file where a student could read it, or swap in a different plausible time without re-running the encoder. This function reverses it.

Usage

```
submission_time(x)
```

Arguments

`x` Path to a JSON file downloaded via `download_answers_button()`, or the raw time string from one.

Details

Nothing else in the file is hashed or signed, so this does **not** detect edited answers. If you need that, compare against work submitted through a channel you control.

Value

The download time as a POSIXct (UTC), invisibly. Also prints a short summary — with the reader's name, email, and device id when `x` is a file.

Examples

```
# Decode a bare time code:
submission_time("52mz7x0xv")

# Or read it, along with the reader's details, from a downloaded file:
answers <- tempfile(fileext = ".json")
writeLines(
  '{"time": "52mz7x0xv", "info": {"name": "Ada", "email": "ada@example.com"}}',
  answers
)
submission_time(answers)
unlink(answers)
```

tutorial_options	<i>Set tutorial-wide options</i>
------------------	----------------------------------

Description

Configures behaviour that applies to the whole tutorial page rather than to one widget. Call it once, in its own `{r}` chunk with `#| echo: false`, anywhere in the `.qmd`; it renders nothing visible. Every option has a default, so a tutorial that never calls this function behaves as described under "Defaults" below.

Usage

```
tutorial_options(allow_skip = FALSE, require_submission = TRUE)
```

Arguments

`allow_skip` Logical. May the reader use a table-of-contents link to unlock and jump to a section they have not reached yet? Default FALSE.

`require_submission` Logical. Must every question and student-info form in a section be submitted before its "Continue" button works? Default TRUE.

Value

A `learnr2_options` object, printed as an invisible HTML element that the page's JavaScript reads at load.

Skipping ahead via the table of contents

`learnr2` reveals a tutorial one #### section at a time, behind "Continue" buttons (see the "Progress persistence" section of [question\(\)](#)). Quarto's table-of-contents sidebar lists every section from the start, and each entry is a plain link to that section's heading. By default (`allow_skip = FALSE`) an entry for a section the reader has not reached yet is dimmed and inert: clicking it does nothing, and it becomes a working link the moment that section is unlocked. Entries for sections already reached navigate normally. So a reader sees the shape of the tutorial and how far along they are, but cannot use the sidebar to see the whole tutorial at once.

With `allow_skip = TRUE`, clicking any entry instead unlocks every section up to and including that one and jumps there – the behaviour of a classic 'learnr' tutorial with `allow_skip: yes`. Use it for reference-style material a reader is meant to move around in freely, such as `learnr2`'s own `hello-learnr2` feature tour.

Neither setting matters for a tutorial rendered with `toc: false`, which has no sidebar and so only ever moves forward one Continue at a time.

Submitting before continuing

By default (`require_submission = TRUE`) the "Continue" button at the end of a section stays disabled, with a short note under it, until every [question\(\)](#) and [student_info\(\)](#) form above it in that section has been submitted. A reader can type anything, but they must submit *something* before the tutorial moves on. Combined with `type = "reflection"` questions, which lock once submitted, this is what lets a tutorial show its own answer right after the reader's without inviting them to copy it back into the box: by the time they see our answer, theirs is already in and cannot be changed. Set `require_submission = FALSE` for reference material a reader should be free to skim, such as `learnr2`'s own `hello-learnr2` feature tour. {webr} code cells are not part of this check; they have no notion of being "submitted".

Defaults

- `allow_skip = FALSE`
- `require_submission = TRUE`

Examples

```
tutorial_options()
tutorial_options(allow_skip = TRUE)
tutorial_options(require_submission = FALSE)
```

Index

add_live_extension, [2](#)
add_live_extension(), [14](#)
answer, [3](#)
answer(), [11](#)
available_tutorials, [4](#)
available_tutorials(), [15](#), [16](#)

check_tutorial, [5](#)
check_tutorial(), [4](#), [14](#)
create_tutorial, [7](#)
create_tutorial(), [15](#)

download_answers_button, [8](#)
download_answers_button(), [6](#), [7](#), [12](#), [13](#),
[22](#), [23](#)

httpuv::runStaticServer(), [16](#)

live_extension_dir, [9](#)

prerender_tutorials, [9](#)
prerender_tutorials(), [17](#), [18](#)

quarto::quarto_preview(), [18](#)
quarto::quarto_render(), [14](#), [15](#), [18](#)
question, [11](#)
question(), [3](#), [6](#), [8](#), [13](#), [14](#), [17](#), [18](#), [21](#), [22](#), [24](#)
quiz, [14](#)
quiz(), [6](#)

render_tutorials, [14](#)
render_tutorials(), [4](#), [5](#)
run_tutorial, [16](#)
run_tutorial(), [4](#), [5](#), [9](#), [10](#), [13](#), [14](#)

show_file, [19](#)
student_info, [21](#)
student_info(), [6–8](#), [13](#), [18](#), [24](#)
submission_time, [22](#)
submission_time(), [8](#)

tempfile(), [10](#), [16](#)

tools::R_user_dir(), [10](#), [16](#)
tutorial_options, [23](#)
tutorial_options(), [6](#)