Package: learnitdown (via r-universe)

October 15, 2024

Type Package

Version 1.8.0

Title R Markdown, Bookdown and Learnr Additions for Learning Material

Description Extension to R Markdown, Bookdown and Learnr for building better learning and e-learning material: H5P integration, course-contextual divs, differed loading of Shiny and learnr applications, and much more ...

Maintainer Philippe Grosjean <phgrosjean@sciviews.org>

Depends R (>= 3.0.0)

Imports getPass, glue, gradethis, httpuv, httr, jsonlite, keyring, later, learnr, magick, mongolite, PKI, remotes, rstudioapi, shiny, shinylogs, shinytoastr, stats, utils, webshot

Suggests covr, knitr, rmarkdown, spelling, testthat (>= 3.0.0)

Remotes rstudio/gradethis

License MIT + file LICENSE

URL https://github.com/learnitr/learnitdown, https://learnitr.github.io/learnitdown/

BugReports https://github.com/learnitr/learnitdown/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

VignetteBuilder knitr

Encoding UTF-8

Language en-US

ByteCompile yes

Config/testthat/edition 3

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

RemoteUrl https://github.com/learnitr/learnitdown

RemoteRef HEAD

RemoteSha 90320c263847ad0d12166750904fc4263685bc1a

Contents

`assignment` *Insert a GitHub (Classroom) assignment or challenge in the document*

Description

Insert a block of class `assign`, `assign2`, `challenge` or `challenge2` with data related to a GitHub (Classroom) assignment.

Usage

```
assignment(
  name,
  url,
  alturl = url,
  course.ids = NULL,
  course.urls = NULL,
  course.starts = NULL,
  course.ends = NULL,
  part = NULL,
  course.names = c(course1 = "Data Science"),
  toc = "",
  clone = TRUE,
  level = 3,
  n = 1,
  type = if (n == 1) "ind. github" else "group github",
  acad_year = "",
  term = "",
  texts = assignment_en(),
  assign.img = "images/list-assign.png",
  assign.link = "github_assignment",
  block = "assign",
  template = "assignment_en.html",
  baseurl = "/"
)

assignment2(
  name,
  url,
  alturl = url,
  course.ids = NULL,
  course.urls = NULL,
  course.starts = NULL,
  course.ends = NULL,
  part = NULL,
  course.names = c(course1 = "Data Science"),
  toc = "",
  clone = TRUE,
  level = 3,
  n = 1,
  type = if (n == 1) "ind. github" else "group github",
  acad_year = "",
  term = "",
  texts = assignment2_en(),
  assign.img = "images/list-assign2.png",
  assign.link = "github_assignment",
  block = "assign2",
  template = "assignment_en.html",
  baseurl = "/"
)

challenge(
  name,
  url,
  alturl = url,
  course.ids = NULL,
  course.urls = NULL,
  course.starts = NULL,
  course.ends = NULL,
  part = NULL,
  course.names = c(course1 = "Data Science"),
  toc = "",
  clone = TRUE,
  level = 3,
  n = 1,
  type = if (n == 1) "ind. challenge" else "group challenge",
  acad_year = "",
  term = "",
  texts = challenge_en(),
  assign.img = "images/list-challenge.png",
  assign.link = "github_challenge",
  block = "challenge",
  template = "assignment_en.html",
  baseurl = "/"
)

challenge2(
  name,
  url,
  alturl = url,
  course.ids = NULL,
  course.urls = NULL,
  course.starts = NULL,
  course.ends = NULL,
  part = NULL,
  course.names = c(course1 = "Data Science"),
  toc = "",
  clone = TRUE,
  level = 3,
  n = 1,
  type = if (n == 1) "ind. challenge" else "group challenge",
  acad_year = "",
  term = "",
  texts = challenge2_en(),
  assign.img = "images/list-challenge2.png",
  assign.link = "github_challenge",
  block = "challenge2",
  template = "assignment_en.html",
  baseurl = "/"
)

assignment_en(title, part.name, alt, sub, course, toc.def)

assignment_fr(title, part.name, alt, sub, course, toc.def)

assignment2_en(title, part.name, alt, sub, course, toc.def)

assignment2_fr(title, part.name, alt, sub, course, toc.def)

challenge_en(title, part.name, alt, sub, course, toc.def)
```

```
challenge_fr(title, part.name, alt, sub, course, toc.def)

challenge2_en(title, part.name, alt, sub, course, toc.def)

challenge2_fr(title, part.name, alt, sub, course, toc.def)
```

Arguments

`name`	The name of the assignment or the challenge (usually the same as the name as the GitHub Classroom assignment).
`url`	The URL of the assignment or challenge (could be a named list for different courses).
`alturl`	An alternate URL to propose to external users not registered in a course. If not provided, it is the same as `url=`.
`course.ids`	Named vector with the Classroom identifiers for the assignments for each course and also possibly, for each group.
`course.urls`	Named vector with the Classroom URLS for each course, and also possibly for groups. Names are the course identifiers in Moodle, or the groups defined for the users. If `NULL`, no course-specific sections are added.
`course.starts`	Named vector (same logic as for `course.urls=`) with starting dates (characters like "YYYY-mm-dd HH:MM:SS"). A corresponding entry in `course.urls=` is required, or the dates will be ignored.
`course.ends`	Named vector (same logic as for `course.urls=`) with ending dates (characters like "YYYY-mm-dd HH:MM:SS"). A corresponding entry in `course.urls=` is required, or the dates will be ignored.
`part`	If the assignment presents several parts in the `README.md` file (because the same assignment is used at different places), indicate the part here. Otherwise, leave the default value `NULL` if there are no parts.
`course.names`	A named character vector with the name of the course. Names used must be the same as for `course.urls`, but there may be more here, and not necessarily in the same order.
`toc`	The exercise table of content (ex-toc) label. If `toc = ""` (default), a generic label is calculated using `toc.def` from `texts`. To not display the exercise in the table of content, use `toc = NULL`.
`clone`	Should the exercise be listed for cloning the repositories (`TRUE` by default)? If `TRUE`, an entry is added in the list of assignments whose repositories should be cloned.
`level`	The difficulty level (1 = easiest, 2 = more difficult, ...)
`n`	The number of students per project (by default, n = 1 for individual assignments and 2 for group assignments).
`type`	The type of exercise. By default, it is `"ind. github"` or `"ind. challenge"` if n = 1, or `"group github"`/`"group challenge"` otherwise.
`acad_year`	The academic year (e.g., 2021-2022).
`term`	The term (e.g., Q1, Q2, Q3).

`texts`	Various sentences used to construct the assignment bloc. You can make a call to `assignment_en()` or `assignment_fr()` as a basis and modify only the sentences you want.
`assign.img`	The relative path to the image to use before the label in the ex-toc.
`assign.link`	The link to the assignment help page (when the user clicks on the image in the ex-toc).
`block`	The class of the div, or the LaTeX environment to use for the assignment block.
`template`	The template file to use for the URL redirection page (currently only assignment_en.html or assignment_fr.html).
`baseurl`	The base URL for the web site.
`title`	The title of the block.
`part.name`	The word to use for "part".
`alt`	The text to display for alternate access to the repository (for non-registered users). Use a string following the `glue()` syntax to replace variables.
`sub`	The text that appears at the bottom of the assignment block.
`course`	The text for items corresponding to courses.
`toc.def`	The default ex-toc label (using `glue()` syntax).

Details

If the URL contains several entries, names are used to create show/hide divs according to the `icourse` user information.

Value

Markdown code that generates the GitHub Classroom assignment block. It is most conveniently used inside an R chunk in your R Markdown document. If you do not want to break your code chunks inside RStudio, you may use something like `if (exists("assignment")) assignment(...)`.

`checker_ack_learnr` *A default checker that just acknowledges submission*

Description

Check code submitted during an exercise. This version just acknowledges reception of the submission. This function is used internally by the tutorials and is not intended for the end-user.

Usage

```
checker_ack_learnr(
  label,
  user_code,
  solution_code,
  check_code,
  envir_result,
  evaluate_result,
  ...
)
```

Arguments

`label`	The label for the learnr exercise.
`user_code`	The code submitted by the user.
`solution_code`	The code provided by the "-solution" chunk.
`check_code`	The code provided by the "-check" chunk.
`envir_result`	The environment after the execution of the chunk.
`evaluate_result`	Result from evaluation of the code.
`...`	Additional parameters (currently not used).

Details

This is a simple checker function for learnitdown learnr applications that just indicates to the user that its answer is taken into account.

Value

A list with components `message`, `correct` and `location`.

See Also

`record_learnr()`

`config` *Configure the R environment for the course (including database information) and provide (or cache) user information in ciphered form.*

Description

Call these functions every time you need to get environment variables set, like the URL, user and password of the MongoDB database used by the course.

Usage

```
config(
  url,
  password,
  cache = file.path(tempdir(), ".learnitdown_config"),
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)

sign_in(
  data,
  password,
  cipher = "aes-256-cbc",
  iv = NULL,
  cache = file.path(tempdir(), ".learnitdown_user"),
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)

sign_out(
  title = "Signing out",
  message = "Do you really want to sign out with learnitdown?",
  cache = file.path(tempdir(), ".learnitdown_user"),
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)

encrypt(
  object,
  password,
  cipher = "aes-256-cbc",
  iv = NULL,
  serialize = TRUE,
  base64 = FALSE,
  url.encode = FALSE
)

decrypt(
  object,
  password,
  cipher = "aes-256-cbc",
  iv = NULL,
  unserialize = TRUE,
  base64 = FALSE,
  url.decode = FALSE
)

lock(
  object,
  password,
  key = "",
  message = "Password for learnitdown:",
  reset = FALSE,
  ref1 = NULL,
  ref2 = NULL
)

unlock(
  object,
  password,
  key = "",
  message = "Password for learnitdown:",
  reset = FALSE,
  ref1 = NULL,
  ref2 = NULL
)
```

Arguments

`url`	The URL of the encrypted file that contains the configuration information.
`password`	The password to crypt, decrypt, lock or unlock the data.
`cache`	The path to the file to use to store a cached version of these data. Access to the database will be checked, and if it fails, the configuration data are refreshed from the URL.
`debug`	Do we issue debugging messages? By default, it is set according to the `LEARNITDOWN_DEBUG` environment variable (yes, if this variable is not `0`).
`data`	The fingerprint data in clear or ciphered form (in this case, the string must start with "fingerprint=").
`cipher`	The cryptography algorithm to use.
`iv`	The initialization vector for the cipher.
`title`	The title of the dialog box prompting to sign out.
`message`	The message of the dialog box prompting to sign out, or asking for a password.
`object`	An object to be encrypted, decrypted, locked or unlocked.
`serialize`	Do we serialize `object` before ciphering it (`TRUE` by default)?
`base64`	Do we encode/decode base64 the object (`FALSE` by default)?
`url.encode`	Do we encode for URL (query) use (`FALSE` by default) ?
`unserialize`	Do we unserialize the resulting object after deciphering it (`TRUE` by default)?
`url.decode`	Do we decode URL before deciphering it (`FALSE` by default)?
`key`	The key that stores the password. It is advised to use something like `course_year`, so that you can manage different passwords for the same course given at different academic years. Using a key, the course password must be entered only once. If not provided, the password is not stored. Note also that the name of an environment variable could be used too for the key. This is convenient on a server like RStudio Connect, for instance.
`reset`	Should we reset the password (`FALSE` by default)?
`ref1`	Code to check the validity of the password
`ref2`	Second code to check password validity.

Value

Invisibly returns `TRUE` if success, or `FALSE` otherwise for `config()`. The encrypted/decrypted object for `encrypt()` and `decrypt()`, or the locked/unlocked object for `lock()` and `unlock()`. The user information for `sign_in()`.

`diagnose_login` *Insert code in an R Markdown document to diagnose personal data information*

Description

This function is used, inside an R Markdown document, to dynamically elaborate a short report on the client-side (by using JavaScript code) that displays the login status of the user (login, contextual data, ...).

Usage

```
diagnose_login(block = "info", lang = "en")
```

Arguments

`block`	Should the diagnostic be included in a block construct (not if `NULL`)?
`lang`	The language to use for diagnostic output (only `'en'` or `'fr'` currently)

Details

Wherever you want to display this diagnostic report in your bookdown page, add on its own line the following instruction: 'r learnitdown::diagnose_login()' between backquotes (you can also change `lang`, if you want).

Value

Nothing, the function is used for its side-effect of adding diagnostic code in the document

---

`h5p` *Insert H5P content in the document*

Description

Insert H5P content in the document

Usage

```
h5p(
  id,
  baseurl,
  idurl = "wp-admin/admin-ajax.php?action=h5p_embed&id=",
  width = 780,
  height = 500,
  toc = "",
  toc.def = "H5P exercise {id}",
  h5p.img = "images/list-h5p.png",
  h5p.link = "h5p"
)
```

Arguments

`id`	The ID of the H5P content in your Wordpress.
`baseurl`	The first part of the URL for your domain, usually something like `https://my.site.com` **without** the trailing `/`.
`idurl`	The URL to the H5P content, usually something like `""` for exercises from h5p.org or h5p.com, or `"wp-admin/admin-ajax.php?action=h5p_embed&id="` for Wordpress (by default).
`width`	The width of the iframe where the H5P content is displayed.
`height`	The height of the iframe.
`toc`	Entry to use in the exercises table of content (`NULL` if no entry, `""` for a default entry based on `toc.def =`).
`toc.def`	Text for a default toc entry using `glue()` syntax for replacement, e.g., `{var}`.
`h5p.img`	The image to display in front of the toc entry
`h5p.link`	The link when the image is clicked (sends to an help page about learnr tutorials).

Details

This function is designed to work inside a Wordpress site where the H5P plugin has been installed. You should also serve your bookdown pages as a subdirectory inside of the same Wordpress site to allow free communication between the parent (the bookdown page) and the child document in the iframe (the H5P content).

Value

HTML code that generates the iframe. It is most conveniently used inside and R inline expression in your R Markdown document on its own line with one blank line above and bellow it.

`launch_shiny` *Launch Shiny application by clicking on its screenshot.*

Description

Shiny applications can be embedded in certain R Markdown documents. However, the application is automatically loaded at the same time as the main page, and that may not be the desired behavior. With `launch_shiny()`, you display just a screenshot of the Shiny application in the page. The user has to click on it to actually launch the application (that replaces the screenshot on the page).

Usage

```
launch_shiny(
  url,
  app = sub("\\?.+$", "", basename(url)),
  imgdir = "images/shinyapps",
  img = paste0(imgdir, "/", app, ".png"),
  createimg = TRUE,
  width = 790,
  height = 500,
  fun = NULL,
  alt1 = "*Click to start the Shiny application.*",
  alt2 = paste0("*Click to start",
    "or [run `{run.cmd}`]({run.url}{run.arg}){{target=\"_blank\"}}.*"),
  toc = "",
  toc.def = "Shiny application {app}",
  run.url = "start_rstudio.html?runrcode=",
  run.cmd = glue("{fun}(\"{app}\")"),
  run.arg = URLencode(run.cmd, reserved = TRUE),
  app.img = "images/list-app.png",
  app.link = "shiny_app",
  ...
)
```

Arguments

`url`	The URL of the Shiny application. If both `app =` and `baseurl =` are provided, you don't need to specify it.
`app`	Name of the shiny application (cannot be duplicated on a page).
`imgdir`	The directory without trailing "/" where images relative to Shiny applications are stored. By default, it is relative to current directory, in `images/shinyapps` subdirectories.
`img`	The relative or absolute path to the image with a screenshot of the Shiny application, as produced by `webshot_shiny()`.
`createimg`	If the app image (`img`) is not found, and there is no default image in `imgdir =`, do we put it there (yes be default)?

`width`	The width of the image and iframe for the app.
`height`	The height of image and iframe.
`fun`	The function to run as alternative to start the Shiny application locally. It is better to fully specify it (`package::function`), and it should take one argument which is the application name in `app =`.
`alt1`	Alternate text to display at the bottom of the screenshot when nothing is provided for `fun =`. If `NULL`, nothing is displayed below the screenshot.
`alt2`	Alternate text to display at the bottom of the screenshot in case `fun =` is provided.
`toc`	Entry to use in the exercises table of content (`NULL` if no entry, `""` for a default entry based on `toc.def =`).
`toc.def`	Text for a default toc entry using `glue()` syntax for replacement, e.g., `{app}`.
`run.url`	The URL to use to start the Shiny application in RStudio server in the SciViews Box. It should generally end with `?runrcode=`, and the R code to execute will be appended to it from `run.arg =`.
`run.cmd`	The command to use to launch the Shiny application in RStudio.
`run.arg`	The URL encoded version of `run.cmd =`.
`app.img`	The image to display in front of the toc entry
`app.link`	The link when the image is clicked (sends to an help page about Shiny applications).
`...`	Not used here, but it allows to add more arguments used by the screenshot addin, like `delay =`, `offsetx =` or `offsety =`, see `webshot_shiny()`.

Value

The HTML content that creates the image and the iframe. The function must be called from within an R inline expression or from an R chunk with `results='asis'` in an HTML-rendered version of the R Markdown document to get the correct result.

See Also

`webshot_shiny()`

Examples

```
# TODO...
```

`learnitdownLearnrSetup` *Set up a learnitdown Learnr application*

Description

This function eases the configuration of the learnr document to get user and database info, record events, use grade this and parameterize learnr.

Usage

```
learnitdownLearnrSetup(
  config,
  sign_in,
  time.limit = 60,
  cap = "R Code",
  echo = FALSE,
  comment = NA,
  use.gradethis = TRUE,
  event.recorder = learnitdown::record_learnr,
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)

learnitdownLearnrBanner(
  title,
  text,
  image,
  align = "left",
  msg.nologin = "Anonymous user, no record!",
  msg.login = "Recording activated for ",
  msg.error = "Error recording activity! "
)

learnitdownLearnrServer(input, output, session)
```

Arguments

`config`	The `config()` command to use to get database info.
`sign_in`	The `sign_in()` command to use to get user info.
`time.limit`	The maximum time allowed to evaluate R code.
`cap`	The caption for R code widgets.
`echo`	Do we echo commands in R chunks?
`comment`	The prefix added before each line of R chunk output.
`use.gradethis`	Do we use gradethis?

`event.recorder`	The function to use as event recorder. you should probably not change the default value here.
`debug`	Do we issue additions debugging informations?
`title`	The Title for the banner.
`text`	Text to print beneath the title.
`image`	URL to an image to display in the banner.
`align`	How is the image aligned: "left" (default), "right", "middle", "top" or "bottom".
`msg.nologin`	The message to display if no user is logged in.
`msg.login`	The message to display if a user is logged in (will be followed by the login).
`msg.error`	The message when an error during recording of activity in the database occurs.
`input`	The Shiny input.
`output`	The Shiny output.
`session`	The Shiny session.

Value

Nothing. The function is used to setup the learnr environment.

`learnitdownShiny` *Create and manage learnitdown Shiny applications*

Description

A learnitdown Shiny application is an application whose events (start, stop, inputs, outputs, errors, result, quit) are recorded. It also provides a 'Submit Answer' and a 'Quit' buttons that manage to check the answer provided by the user and to close the application cleanly.

Usage

```
learnitdownShiny(title, windowTitle = title)

learnitdownShinyVersion(version)

submitAnswerButton(
  inputId = "learnitdown_submit_",
  label = "Submit",
  class = "btn-primary",
  ...
)

quitButton(
  inputId = "learnitdown_quit_",
  label = "Quit",
  class = "btn-secondary",
  ...
)

submitQuitButtons()
```

Arguments

`title`	The title of the Shiny application
`windowTitle`	The title of the window that holds the Shiny application, by default, it is the same as `title`. If `title` = `NULL`, no title is added.
`version`	The version number (in a string character format) of the Shiny application to use in the events logger.
`inputId`	The identifier of the button ("learnitdown_submit_" or "learnitdown_quit_").
`label`	The button text ("Submit" or "Quit").
`class`	The bootstrap class of the button.
`...`	Further arguments passed to `shiny::actionButton()`.

Value

The code to be inserted at the beginning of the Shiny application UI for `learnitdownShiny()` or where the buttons should be located for the other function.

See Also

`trackEvents()`, `record_shiny()`

Examples

```
learnitdownShiny("My title")
```

`learnitdown_init` *Initialize learnitdown features in an R Markdown document*

Description

This function must be called in a script run by `before_chapter_script` entry in `_bookdown.yml` to create required `style.css` and `header.html` files.

Usage

```
learnitdown_init(
  shiny = TRUE,
  h5p = TRUE,
  use.query = FALSE,
  iframe.links = TRUE,
  details.css = TRUE,
  baseurl = "https://example.org",
  institutions = c("institution1", "institution2"),
  courses = c("course1", "course2", "course3"),
  style = "style.css",
  style0 = "style0.css",
  header = "header.html",
  header0 = "header0.html",
  hide.code.msg = "See code"
)
```

Arguments

`shiny`	Do we use Shiny applications and do we want to pass parameters and or launch the application on a click?
`h5p`	Do we use H5P served from a Wordpress site in the same domain as our R Markdown document? The H5P integration plugin, and the H5PxAPIkatchu Wordpress plugins must be installed in order to serve H5P apps and to record the H5P events through the xAPI interface.
`use.query`	Do we collect user/course/institution data through the URL query string (the part after the question mark in the URL).
`iframe.links`	If our document is displayed in an iframe, external link should better target their parent window. With this option, external links with no defined target are automatically retargeted when the page loads.
`details.css`	Do we want to enhance the `<details>` section with a summary surrounded by a light gray box in order to better evidence it.
`baseurl`	The URL where the site is server from (for H5P integration), it is also the base URL for the associated Wordpress server with H5P plugin. Provide it **without** the trailing /!
`institutions`	The list of possible institutions that have specific sections in this document.
`courses`	The list of courses with specific sections in this document.
`style`	The path to the 'style.css' file.
`style0`	The path to a file with additional content to add to the 'style.css' file.
`header`	The path to the 'header.html' file.
`header0`	The path to a file with additional content to add to 'header.html'.
`hide.code.msg`	The message to display for hidden code.

Value

A list with `css` and `html` components with the content that was added to respective files is returned invisibly for debugging purposes '(the function is mainly used for its side effect of creating `style.css` and `header.html` files for the bookdown format).

Examples

```
# This is better placed in a setup R chunk or an R inline expression on its
# own line. To see the code injected, use `cat()` at the R prompt:
odir <- setwd(tempdir())
dir.create("temp")
setwd("temp")
# Create fake style0.css and header0.html files to see what happens
cat("\n/* Content from style0.css */\n", file = "style0.css")
cat("\n<!-- Content from header0.html -->\n", file = "header0.html")
# Create style.css and header.html files
(learnitdown_init())
cat(readLines('style.css'), sep = "\n")
cat(readLines('header.html'), sep = "\n")
setwd("..")
unlink("temp")
setwd(odir)
rm(odir)
```

`learnr` *Insert a block for a learnr tutorial*

Description

Insert Markdown text to link to a learnr tutorial.

Usage

```
learnr(
  id,
  title = NULL,
  package,
  toc = "",
  text = "Now let's make the exercises in the following tutorial:",
  toc.def = "Tutorial {id}",
  rstudio.url = "start_rstudio.html",
  tuto.img = "images/list-tuto.png",
  tuto.link = "tutorial"
)
```

Arguments

`id`	Identifier of the learnr tutorial (as `tutorial:id` in the YAML section).
`title`	The title of the tutorial (`title` in the YAML section). If `NULL`, the `id` is used instead.
`package`	Package where the learnr tutorial is defined.

`toc`	Entry to use in the exercises table of content (`NULL` if no entry, `""` for a default entry based on `toc.def =`).
`text`	The text to display in the learnr block.
`toc.def`	Text for a default toc entry using `glue()` syntax for replacement, e.g., `{id}`.
`rstudio.url`	The URL to open a page in RStudio server in the SciViews box.
`tuto.img`	The image to display in front of the toc entry
`tuto.link`	The link when the image is clicked (sends to an help page about learnr tutorials).

Value

The Markdown chunk to insert a learnr tutorial block in the document.

`obfuscate` *Obfuscate answers in learnr documents*

Description

Obfuscate answers in learnr documents

Usage

```
obfuscate(object, password)

._(object)

obfuscate_logical(x, y, r = FALSE)

ans(correct, text, message = NULL)
```

Arguments

`object`	An R object to obfuscate, or Obfuscation string to decrypt.
`password`	The password to use (otherwise, the password defined by the application is used)
`x`	A logical value or an integer.
`y`	Obfuscation parameter.
`r`	Reset obfuscation (not intended for end-user)?
`correct`	Is the answer is correct or not (either a logical value, or an integer that obfuscates the results).
`text`	Text of the option.
`message`	Message displayed if the item is selected

Value

The encrypted or decrypted object.

Examples

```
obfuscate("test", "password")
```

`read_shinylogs` *Read shinylogs log data and format them in a data.frame*

Description

Learnitdown Shiny applications are a special kind of Shiny applications that logs events and check results. It uses the `shinylogs` package to log Shiny events and `read_shinylogs()` reads such logs and convert their data into a format that is suitable to include, say in a MongoDB database.

Usage

```
read_shinylogs(file, version = "0", log.errors = TRUE, log.outputs = FALSE)
```

Arguments

`file`	The path to the RDS file that contains the shinylogs log data.
`version`	The version of the Shiny application. Version is not recorded by shinylogs, so, we must provide it here.
`log.errors`	Do we record errors too?
`log.outputs`	Do we record outputs too (note that results are recorded with inputs)?

Value

A data frame with the different events logged (one per line).

See Also

`record_shiny()`, `trackEvents()`

`record_learnr` *Record results of learnr exercises in a MongoDB database*

Description

Record tutorial submissions in a MongoDB database. The function is used by learnitdown learnr tutorials and is not for end-users.

Usage

```
record_learnr(tutorial_id, tutorial_version, user_id, event, data)

user_name(value)

user_email(value)
```

Arguments

`tutorial_id`	The identifier of the tutorial.
`tutorial_version`	The version of the tutorial.
`user_id`	The user identifier for this learnr process.
`event`	The event that triggers the record, like `exercise_submission` or `question_submission`
`data`	A JSON field with event-dependent data content. If `NULL`, only a test to see if the database is responding is performed.
`value`	The new value for user name or email (if not provided, the current value is returned).

Value

Nothing. The function is used for its side-effects.

See Also

`send_mail_learnr()`

`record_shiny` *Record Shiny events in a MongoDB database*

Description

Given a path that contains `shinylogs` events in .rds format, read these events and transfer them into a MongoDB database.

Usage

```
record_shiny(
  path,
  url,
  db,
  collection = "events",
  version = "0",
  log.errors = TRUE,
  log.outputs = FALSE,
  drop.dir = FALSE,
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)
```

Arguments

`path`	The directory that contains shinylogs .rds files
`url`	The mongodb url.
`db`	The database name.
`collection`	The name of the collection where to insert the documents.
`version`	The version of the running Shiny application.
`log.errors`	Do we record errors too?
`log.outputs`	Do we record outputs too (note that results are recorded with inputs)?
`drop.dir`	If `TRUE` and path is empty at the end of the process, drop the logs directory.
`debug`	Do we debug the events recording by issuing extra messages? By Default the value in the environment variable `LEARNITDOWN_DEBUG` is used and debugging will be done if this value is different to `0`.

Value

`TRUE`if there where log files to export, `FALSE` otherwise.

See Also

`read_shinylogs()`, `trackEvents()`

`run` *Run learnitdown learnr tutorials or Shiny apps from a package after update*

Description

These functions are convenient in the framework of a course whose learnrs and Shiny applications may be updated during the course. The `update()` function checks if an update is available (respecting the version of R), and the `run()` (for learnrs), or `runApp()` (for Shiny applications) manage to run the item in a friendly way.

Usage

```
run(
  tutorial,
  package,
  github_repos = NULL,
  ...,
  update = ask,
  ask = interactive(),
  upgrade = "never"
)
```

```
run_app(
  app,
  package,
  github_repos = NULL,
  ...,
  update = ask,
  ask = interactive(),
  upgrade = "never",
  in.job = TRUE,
  max.wait = 60
)

update_pkg(package, github_repos, upgrade = "never")
```

Arguments

`tutorial`	The name of the tutorial to use. If not provided, a list of available tutorials is displayed.
`package`	The package from where to run the tutorial.
`github_repos`	The GitHub repository where the package is developed (for updates), use `NULL` to prevent any updates.
`...`	Further arguments passed to `run_tutorial()` (for learnrs), or to `runApp()` (for Shiny applications).
`update`	Do we check for an updated version first, and if it is found, update the package automatically?
`ask`	In case `tutorial` or app is not provided, do we ask to select in a list?
`upgrade`	When a new version of the main package is found, do we also upgrade dependencies ? By default, never, but use `"ask"` to ask user.
`app`	The name of the Shiny application to run. If not provided, a list of available apps is displayed.
`in.job`	Should the application be run in a Job in RStudio (`TRUE` by default)?
`max.wait`	How many seconds do we wait for the Shiny app to start (60 sec by default)?

Value

The result returned by `run_tutorial()` for `run()`, or by `runApp()` for `run_app()`. The `update()` function return `TRUE` or `FALSE`, depending if the package is updated or not.

See Also

`run_tutorial()`, `runApp()`

Examples

```
## Not run:
#' # To start from a list of available tutorials:
run(package = "my_package")
run("my_tutorial", package = "my_package")
run_app(package = "mypackage")
run_app("my_shiny_app", package = "mypackage")

## End(Not run)
```

`send_mail_learnr` *Send your learnr submissions by email*

Description

Your submissions are send to a central database. However, in case that database is not accessible, the data is stored locally. This function uses your plain email to send your records. Note that, once the email is created, the local version of your records is reset. So, if you finally decide to NOT send the email, these records are lost (in this case, call your teachers to recover them, if you have to.)

Usage

```
send_mail_learnr(
  address,
  subject = "Learnr activity",
  file = Sys.getenv("LOCAL_STORAGE")
)
```

Arguments

`address`	The mail address to send the data to.
`subject`	The title of the mail.
`file`	The file that contains your learnr activity information.

Details

In case the MongoDB cannot be reach, learnr events are stored in a local file. This function allows to submits its content through email as an alternate way to collect learnr activity.

Value

The data are returned invisibly.

See Also

`record_learnr()`

Examples

```
## Not run:
send_mail_learnr("me\@mymail.org")

## End(Not run)
```

show_ex_toc *Insert a table of content for the exercises at the end of a bookdown chapter*

Description

For the various exercise types (h5p, shiny apps, learnrs & GitHub assignments/challenges) we add toc entries with `h5p()`, `launch_shiny()`, `learnr()`, `assignment()`, and `challenge()` respectively. This function creates the table of content for all these exercises.

Usage

```
show_ex_toc(header = "", clear.it = TRUE)

clean_ex_toc()
```

Arguments

`header`	A Markdown text to place as header of the exercises toc.
`clear.it`	Do we clear the toc list (`TRUE` by default)?

Value

The Markdown chunk with the exercises toc.

trackEvents *Track events, the submit or the quit buttons*

Description

These functions provide the required code to be inserted in the server part of a Shiny application to track events (start, stop, inputs, outputs, errors, result or quit), to check the answer when the user clicks on the submit button and to cleanly close the application when the user clicks on the quit button.

Usage

```
trackEvents(
  session,
  input,
  output,
  sign_in.fun = NULL,
  url = Sys.getenv("MONGO_URL"),
  url.server = Sys.getenv("MONGO_URL_SERVER"),
  db = Sys.getenv("MONGO_BASE"),
  user = Sys.getenv("MONGO_USER"),
  password = Sys.getenv("MONGO_PASSWORD"),
  version = getOption("learnitdown.shiny.version"),
  path = Sys.getenv("LEARNITDOWN_LOCAL_STORAGE", "shiny_logs"),
  log.errors = TRUE,
  log.outputs = FALSE,
  drop.dir = TRUE,
  config = NULL,
  debug = Sys.getenv("LEARNITDOWN_DEBUG", 0) != 0
)

trackSubmit(
  session,
  input,
  output,
  solution = NULL,
  max_score = NULL,
  comment = "",
  message.success = "Correct",
  message.error = "Incorrect",
  score.txt = "Score",
  check.solution = check_shiny_solution
)

trackQuit(session, input, output, delay = 60)

check_shiny_solution(answer, solution)
```

Arguments

`session`	The current Shiny `session`.
`input`	The Shiny `input` object.
`output`	The Shiny `output` object.
`sign_in.fun`	The function that can get user info from the `sign_in()`, or `NULL` by default to disable using local user data.
`url`	The URL to reach the MongoDB database. By default, it is read from the `MONGO_URL` environment variable.
`url.server`	The URL to reach the MongoDB database. By default, it is read from the `MONGO_URL_SERVER` environment variable.
`db`	The database to populate in the MongoDB database. By default, it is read from the `MONGO_BASE` environment variable.
`user`	The user login to the MongoDB database. By default, it is read from the `MONGO_USER` environment variable.
`password`	The password to access the MongoDB database. By default, it is read from the `MONGO_PASSWORD` environment variable.
`version`	The version of the current Shiny application. By default, it is the `learnitdown.shiny.version` option, as set by `learnitdownShinyVersion()`.

`path`	The path where the temporary `shinylogs` log files are stored. By default, it is set to the `LEARNITDOWN_LOCAL_STORAGE` environment variable, or to `shiny_logs` subdirectory of the application if not defined. If that directory is not writable, a temporary directory is used instead.
`log.errors`	Do we log error events (yes by default)?
`log.outputs`	Do we log output events (no by default)?
`drop.dir`	Do we erase the directory indicated by `path` = if it is empty at the end of the process (yes by default).
`config`	The result of the call to `config()`, if done.
`debug`	Do we debug recording of events using extra messages? By default, it is the value of the environment variable `LEARNITDOWN_DEBUG`, and debugging is activated when that value is different to `0`.
`solution`	The correct solution as a named list. Names are the application inputs to check and their values are the correct values. The current state of these inputs will be compared against the solution, and the `message.success` or `message.error` is displayed accordingly. Also a result event is triggered with the `correct` field set to `TRUE` or `FALSE` accordingly. If `solution` = `NULL`, then the answer is considered to be always correct (use this if you just want to indicate that the user's answer is recorded in the database). For numeric values, use 'c(min = x, max = y) to check if values are within a range.
`max_score`	The highest score value that could be attributed if the exercise is correctly done. By default, it is `NULL` meaning that the higher score would be equal to the number of items to check in `solution`.
`comment`	A string with a comment to append to the value of the result event.
`message.success`	The message to display is the answer is correct ("Correct" by default).
`message.error`	The message to display if the answer is wrong ("Incorrect" by default).
`score.txt`	The word to use for "score" ("Score" by default).
`check.solution`	The function to use to check if solution provided by a Shiny application is correct or not. By default, it is `check_shiny_solution()`.
`delay`	The time to wait before we close the Shiny application in sec (60 sec by default). If `delay` = `-1`, the application is **not** closed, only the session is closed when the user clicks on the quit button.
`answer`	A named list of the answer data to check against solution. This object is provided by `trackSubmit()`.

Value

The code to be inserted in the server part of the learnitdown Shiny application in order to properly identify the user and record the events.

See Also

`learnitdownShinyVersion()`, `sign_in()`

webshot_shiny *Create the screenshot image of a Shiny application with a click icon*

Description

The `webshot_shiny()` function is designed to create the screenshot image of a Shiny application, with an icon suggesting to click on it for launching the application. The image created can then be used by `launch_shiny()` in a bookdown to differ the start of Shiny application, while displaying useful information to the user (how the Shiny application would look like if it was started).

Usage

```
webshot_shiny(
  url,
  app = NULL,
  imgdir = "images/shinyapps",
  img = paste0(imgdir, "/", app, ".png"),
  width = 790,
  height = 500,
  offsetx = 30,
  offsety = 30,
  delay = 10,
  ...
)
```

Arguments

`url`	The URL to launch the Shiny app, or to a screenshot of the app in PNG format.
`app`	The name of the Shiny application. If `NULL`, the base name of the URL without the extension is used. are provided, you don't need to specify it.
`imgdir`	The directory without trailing "/" where images relative to Shiny applications are stored. By default, it is relative to current directory, in `images/shinyapps` subdirectories.
`img`	The path to the image that is created. Not needed if `app =` and `imgdir =` are provided.
`width`	The requested weight of the screenshot (it may differ if the Shiny application defines other (limit) values.
`height`	The requested height of the screenshot (idem).
`offsetx`	The offset from left where to place the click icon in pixels.
`offsety`	The offset to bottom where to place the click icon in pixels.
`delay`	Time to wait (in sec) after the Shiny application has started and before the screenshot is taken. If the screenshot does not contain the complete application UI, try increase this value.
`...`	Further arguments passed to `launch_shiny()` but not used here.

Value

The path to the created image, invisibly.

See Also

launch_shiny()

Examples

```
## Not run:
# We wait 10 sec to make sure it is loaded when the screenshot is taken
(webshot_shiny("https://phgrosjean.shinyapps.io/histogram/", delay = 10))
 # Now, look at this image. You can use it with launch_shiny()

## End(Not run)
```

Index