


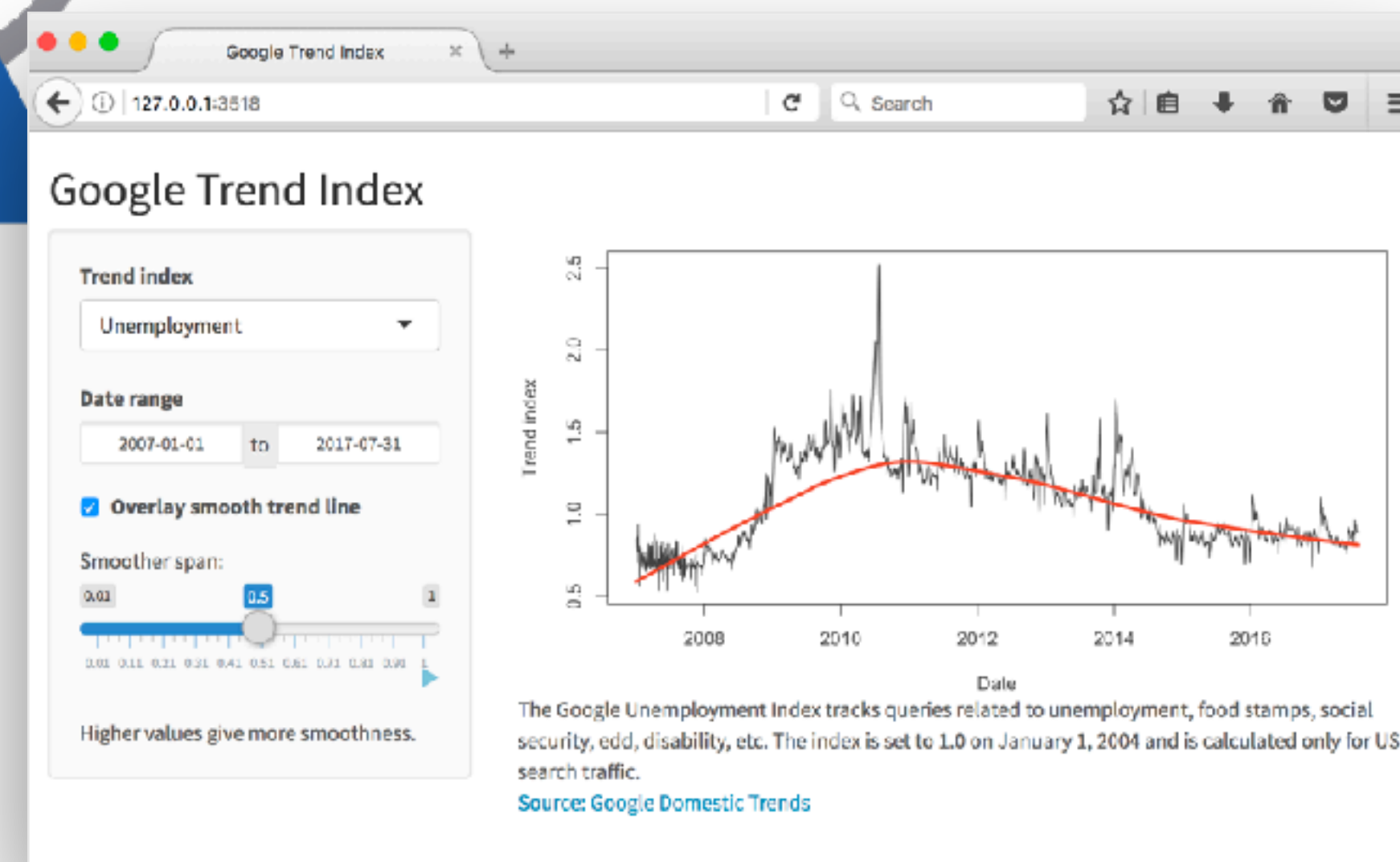
Building user interfaces



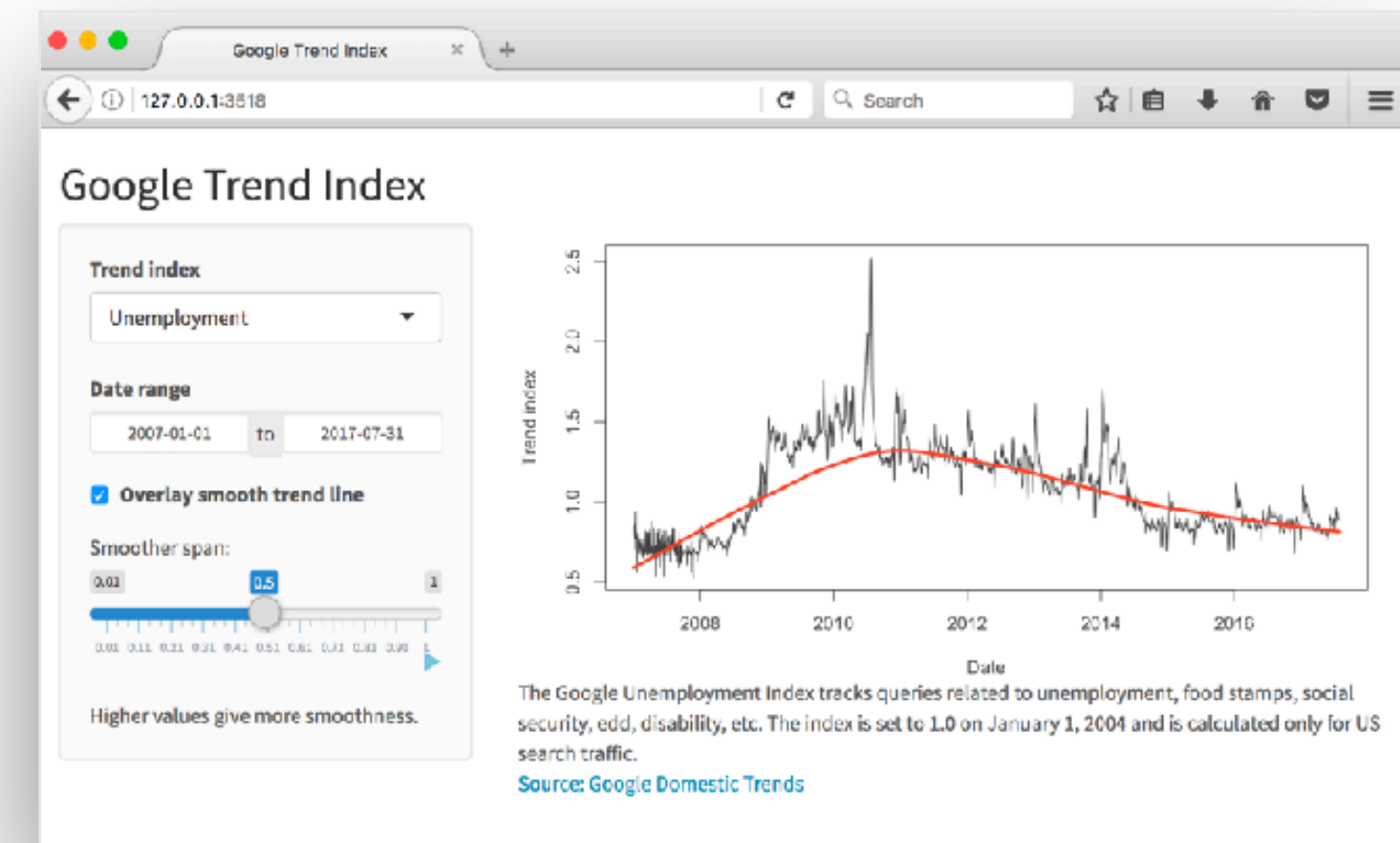
Mine Çetinkaya-Rundel

@minebocek 
mine-cetinkaya-rundel 
mine@rstudio.com 

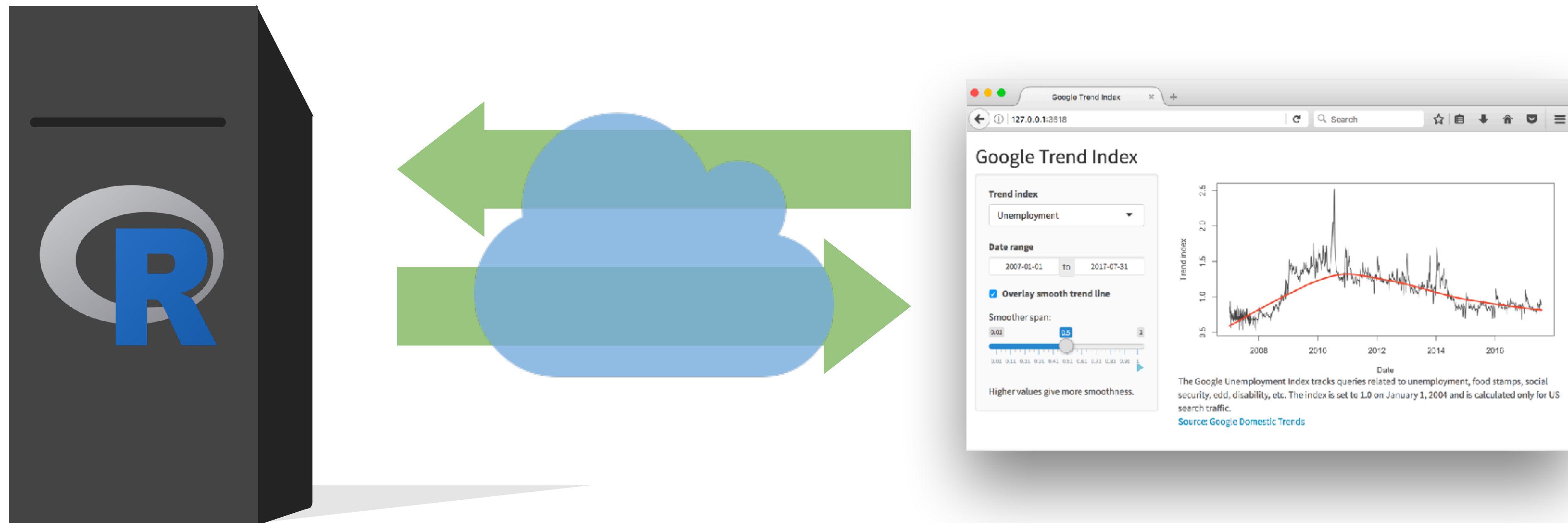
Every Shiny app has a webpage that the user visits,
and behind this webpage there is a computer
that serves this webpage by running R.

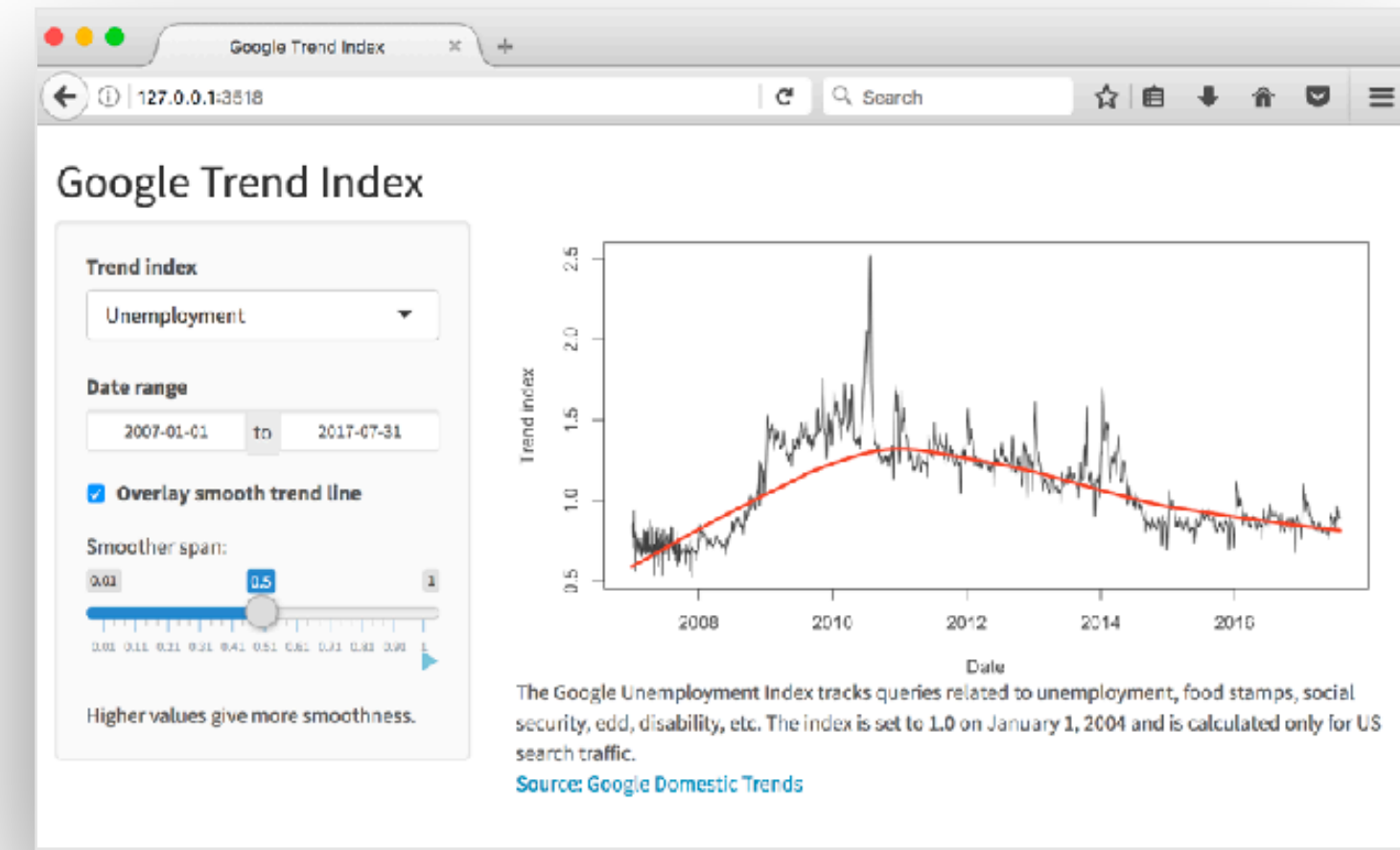
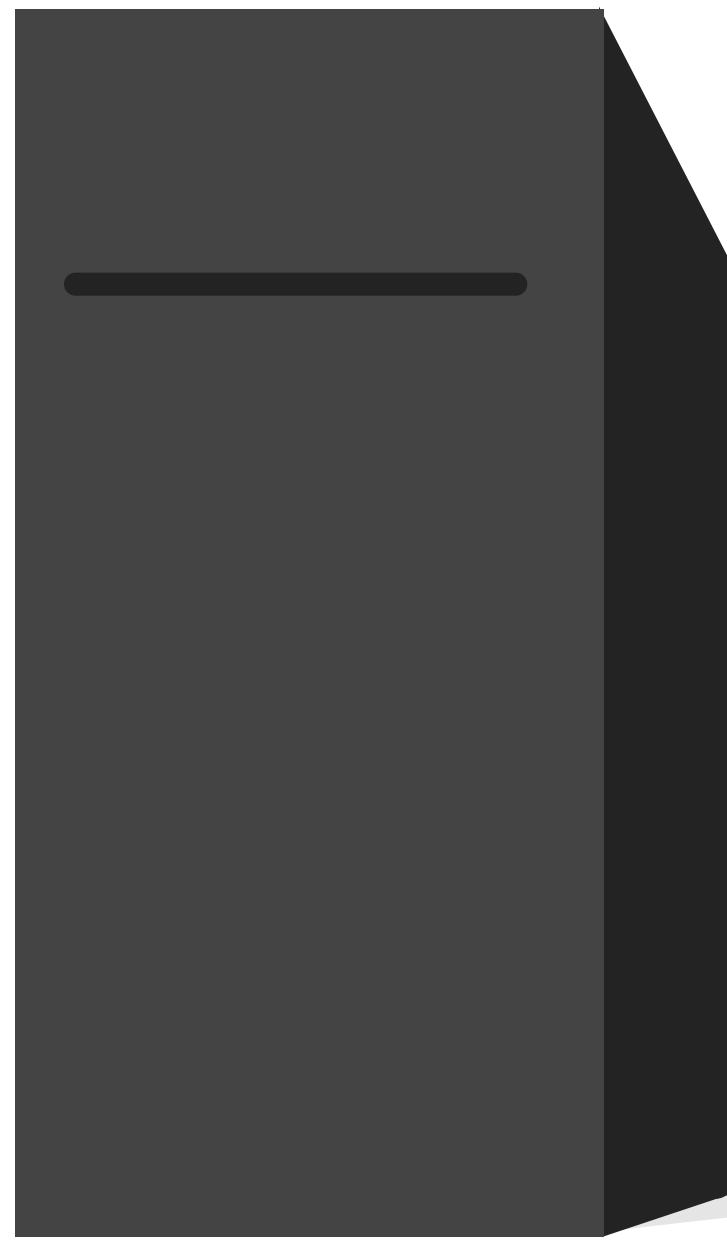


When running your app locally,
the computer serving your app is your computer.



When your app is deployed,
the computer serving your app is a web server.





Server instructions



User interface

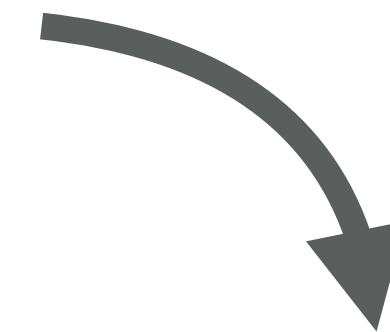
High level view

Multiple levels of abstraction

High-level funcs
`fluidRow(...)`

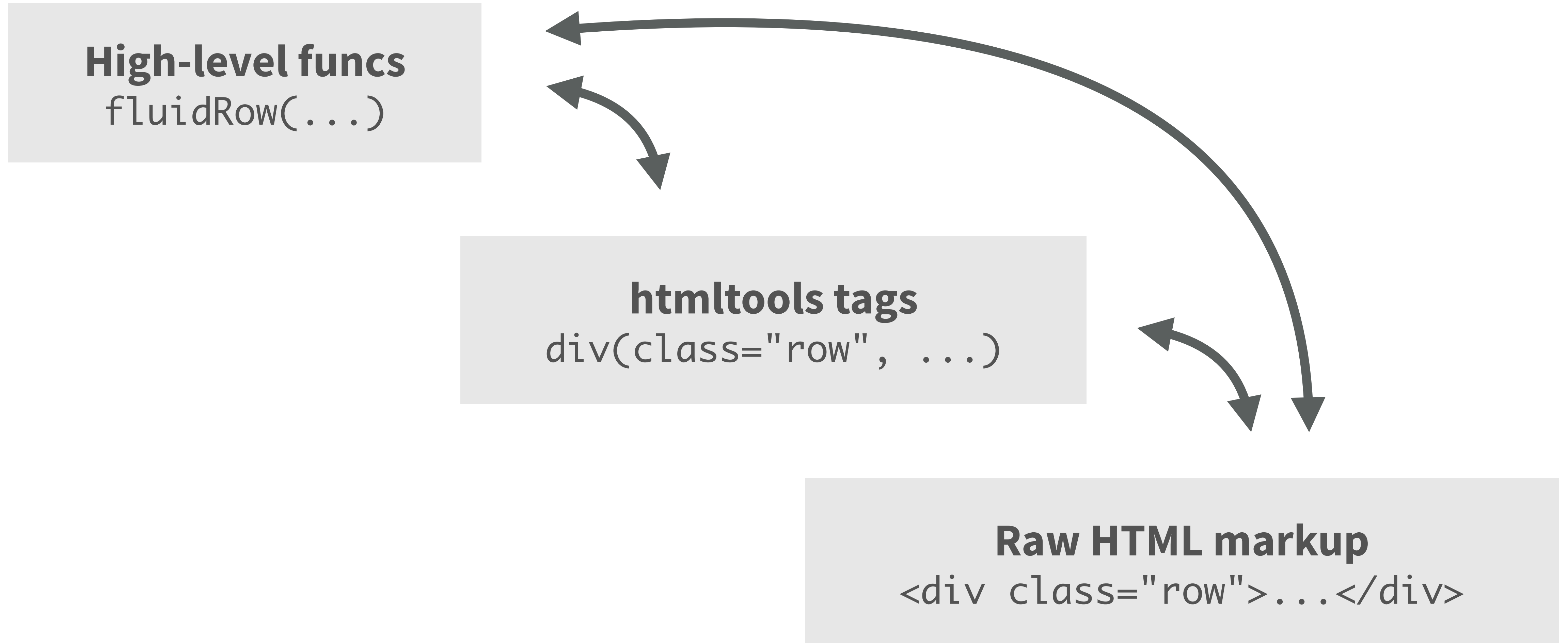


htmltools tags
`div(class="row", ...)`



Raw HTML markup
`<div class="row">...</div>`

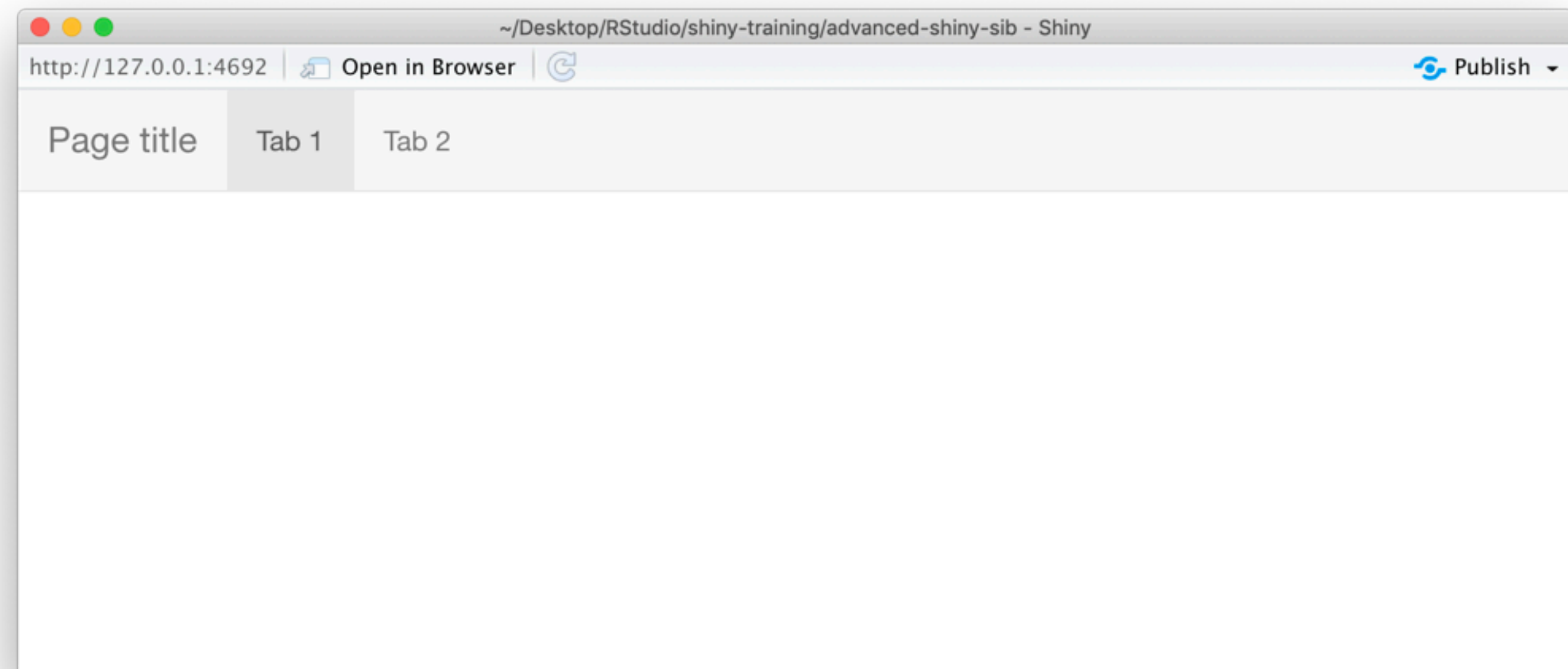
Mix and match freely



High level functions

- ▶ Functions that return htmltools objects
- ▶ Pros
 - ▶ Less code, clearer intent
 - ▶ Anyone can make their own
- ▶ Cons
 - ▶ Less flexible

```
01 navbarPage(  
02   "Page title", id = "nav",  
03   tabPanel("Tab 1"),  
04   tabPanel("Tab 2")  
05 )
```



HTMLtools objects

- ▶ HTML-generating R functions
- ▶ Pros
 - ▶ All the power of HTML, but looks like R
 - ▶ Automated CSS/JS dependency handling
 - ▶ More composable, programmable than HTML
- ▶ Cons
 - ▶ Easy to misplace commas
 - ▶ Almost as verbose as raw HTML

```
nav(class="navbar navbar-default navbar-static-top",
  div(class="container-fluid",
    div(class="navbar-header",
      span(class="navbar-brand", "Page title")
    ),
    ul(class="nav navbar-nav shiny-tab-input",
      li(class="active",
        a(href="#tab-7546-1", data-toggle="tab",
          "Tab 1")
      ),
      li(class="active",
        a(href="#tab-7546-2", data-toggle="tab",
          "Tab 2")
      )
    )
  )
)
```

Raw HTML

- ▶ Pros
 - ▶ Can do anything that's possible in a webpage
 - ▶ Familiar for designers, web developers
- ▶ Cons
 - ▶ Unfamiliar for many R users
 - ▶ Potentially lots of HTML needed for conceptually simple tasks
 - ▶ CSS/JavaScript dependencies must be handled manually

```
<nav class="navbar navbar-default navbar-static-top">
  <div class="container-fluid">
    <div class="navbar-header">
      <span class="navbar-brand">Page title</span>
    </div>
    <ul class="nav navbar-nav shiny-tab-input">
      <li class="active">
        <a href="#tab-7546-1" data-toggle="tab">Tab 1</a>
      </li>
      <li>
        <a href="#tab-7546-2" data-toggle="tab">Tab 2</a>
      </li>
    </ul>
  </div>
</nav>
<div class="container-fluid">
  <div class="tab-content" data-tabsetid="7546">
    <div class="tab-pane active" data-value="Tab 1">
      <div class="tab-pane" data-value="Tab 2" id="tab-7546-2">
      </div>
    </div>
  </div>
</div>
```



lets R users write user interfaces using a simple, familiar-looking API...

...but there are no limits for advanced users

Ladder of UI progression

1. Use built-in Shiny inputs/outputs and layouts
2. Use functions from external packages
3. Use tag objects and write UI functions
4. Author HTML templates
5. Create custom inputs/outputs, wrap existing CSS/JS libraries and frameworks



Built-in Shiny inputs/outputs and layouts

Inputs

Shiny: CHEAT SHEET

Basics
A Shiny app is a web page (UI) connected to a computer running a live R session (Server).
Users can manipulate the UI, which will cause the server to update the UI's outputs by running R code.
APP TEMPLATE
Sign writing a new app with the template. Preview the app by running the code at the R command line.
The easiest way to share your app is to host it on shinyapps.io, a cloud-based service from RStudio.

Building an App
Complete the template by adding arguments to fluidPage() and a body() for the server function.
Add outputs with "output()" functions.
Tell server how to render outputs with R in the server function. To do this:
1. Refer to outputs with output\$id.
2. Refer to inputs with input\$id.
3. Wrap code in a render() function before saving to output.
Save your template as app.R. Alternatively, split your template into two files named ui.R and server.R.
ui.R contains everything you would save to ui.
server.R ends with the function you would save to server.
No need to call shinyApp().
I want an app with runApp() path to shinyApp().
The directory name is the same as the app (options): define objects available to both ui.R and server.R (options): used in showcase mode (options): data, script, etc (options): directory others to share with web browser (images, CSS, js, etc) Must be named "www".

Inputs
Collect values from the user.
Accept the current value or an input object with input() or input\$ - input values are reactive.
actionButton(inputId, label, icon, ...)
actionLink(inputId, label, icon, ...)
checkboxGroupInput(inputId, label, choices, selected, inline)
checkboxInput(inputId, label, value)
dateInput(inputId, label, value, min, max, format, startview, weekstart, language)
dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, separator, language)
fileInput(inputId, label, multiple, accept)
numericInput(inputId, label, value, min, max, step)
passwordInput(inputId, label, value)
radioButtons(inputId, label, choices, selected, inline)
selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())
sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
submitButtons(text, icon) (Prevents reactions across entire app)
textInput(inputId, label, value)

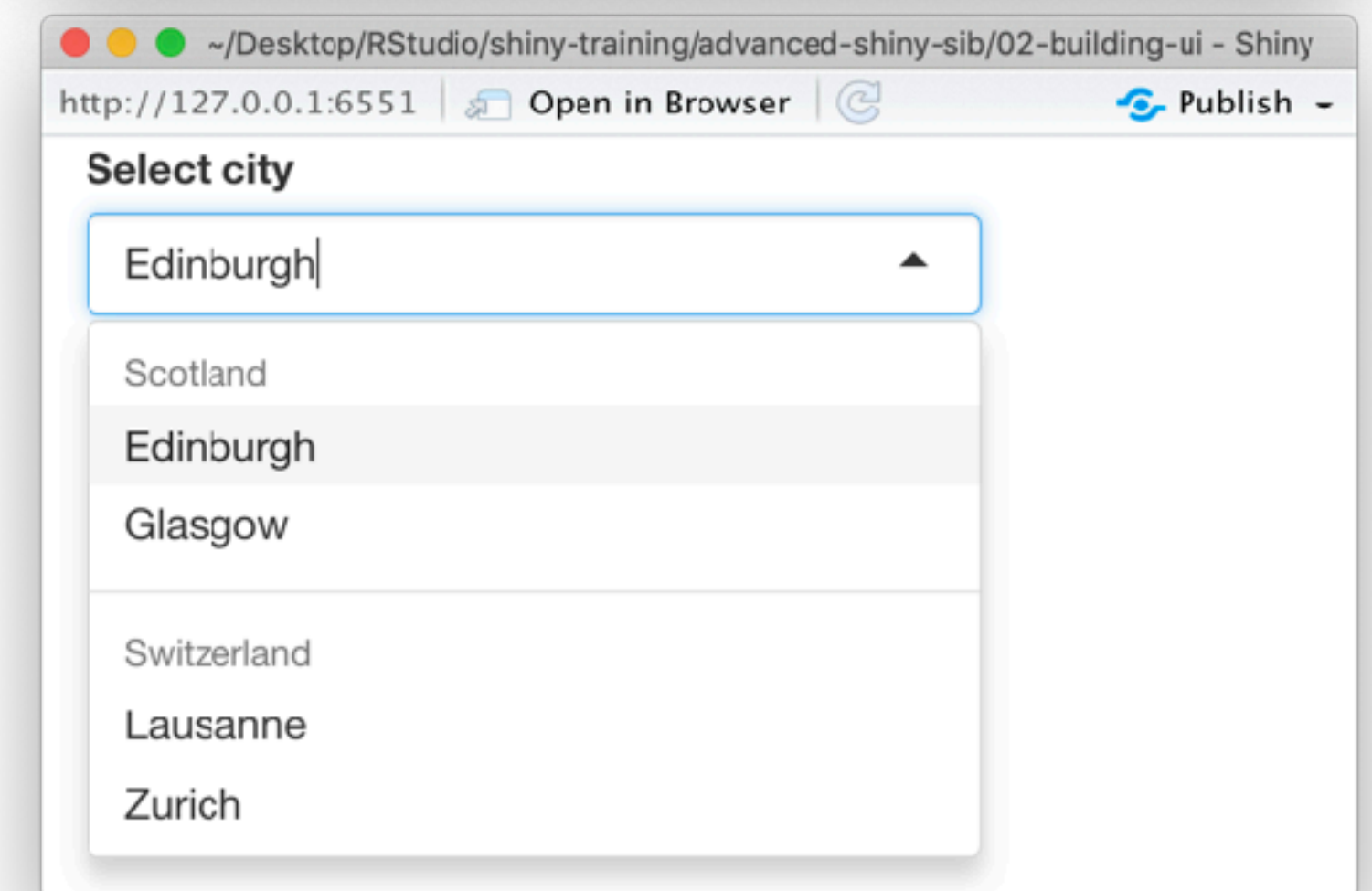
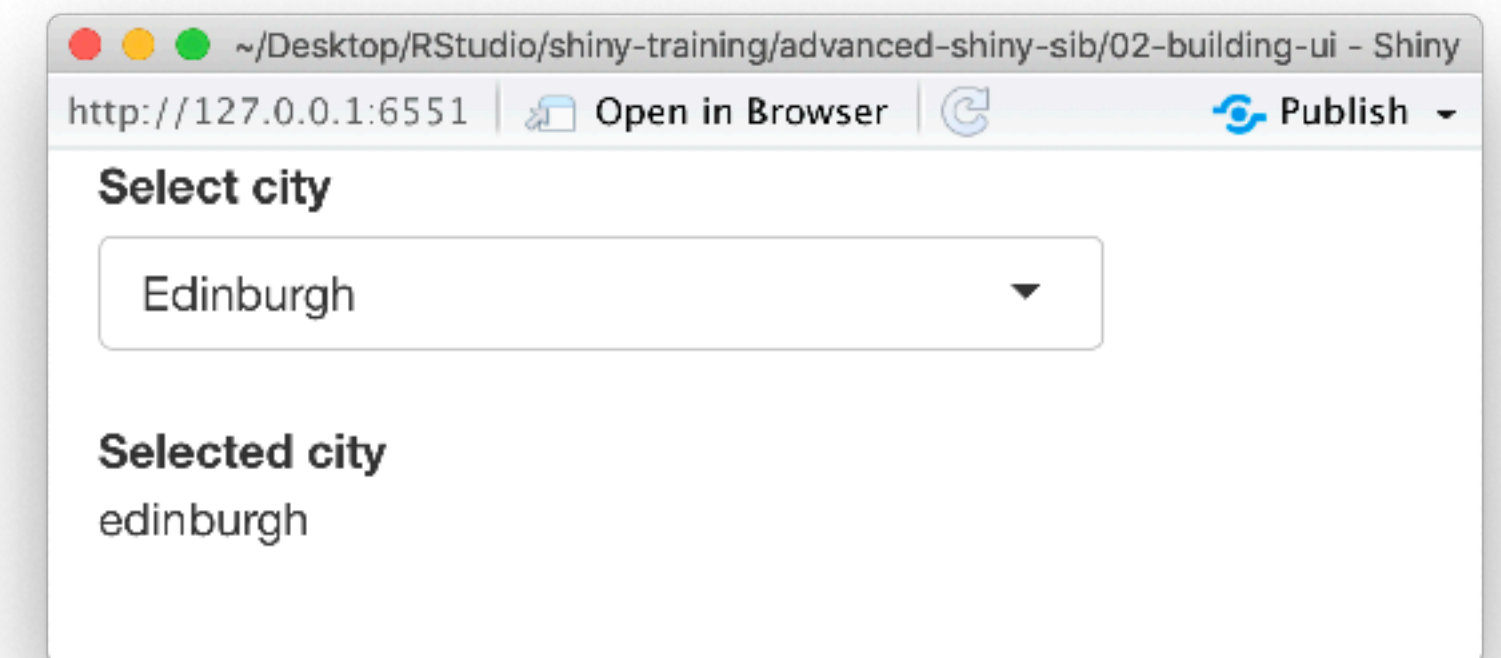
Outputs
render() and "Output()" functions work together to send R output to the UI.
renderDataTable(outputId, options, dataTableOutput, con, ...)
renderForm(inputId, env, quoted, selectId, ...)
renderImage(inputId, env, quoted, src, alt, ...)
renderText(inputId, env, quoted, text, ...)
renderTextOutput(outputId, ...)
renderTextOutput(outputId, container, inline, ...)
renderTextOutput(outputId, inline, container, ...)
renderTextOutput(outputId, inline, container, ...)

<input type="button" value="Action"/>	actionButton (inputId, label, icon, ...)
Link	actionLink (inputId, label, icon, ...)
<input checked="" type="checkbox"/> Choice 1 <input checked="" type="checkbox"/> Choice 2 <input type="checkbox"/> Choice 3	checkboxGroupInput (inputId, label, choices, selected, inline)
<input checked="" type="checkbox"/> Check me	checkboxInput (inputId, label, value)
<input type="text" value="2015-06-08"/>	dateInput (inputId, label, value, min, max, format, startview, weekstart, language)
<input type="text" value="2015-06-08"/> to <input type="text" value="2015-06-08"/>	dateRangeInput (inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
<input type="button" value="Choose File"/>	fileInput (inputId, label, multiple, accept)

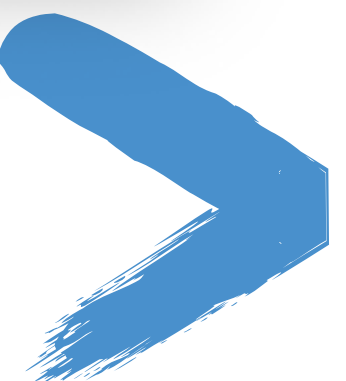
<input type="text" value="1"/>	numericInput (inputId, label, value, min, max, step)
<input type="password" value="....."/>	passwordInput (inputId, label, value)
<input checked="" type="radio"/> Choice A <input type="radio"/> Choice B <input type="radio"/> Choice C	radioButtons (inputId, label, choices, selected, inline)
<input type="text" value="Choice 1"/>	selectInput (inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())
<input type="range" value="5"/>	sliderInput (inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)
<input type="button" value="Apply Changes"/>	submitButton (text, icon) (Prevents reactions across entire app)
<input type="text" value="Enter text"/>	textInput (inputId, label, value)

Your turn

- ▶ If you have build a Shiny app before, you've probably used a `selectInput()` widget. Sometimes the choices you want to show your users are spelled/formatted differently than how you want to use them in your Shiny code, e.g. you might want to use Titlecase in the UI but lowercase under the hood. Modify **02-building-ui** > **01-ui.R** in this way.
- ▶ **Stretch goal:** If you have a moderately long or hierarchical list, you might want to organise your choices under subheadings. Modify **02-building-ui** > **01-ui.R** further break up the list of cities into two under two subheadings: Scotland and Switzerland. **Hint:** Read the documentation for `selectInput()`.



5_m 00_s



Solution

Solutions to the previous exercises

> **02-building-ui** > **02-ui.R**

> **02-building-ui** > **03-ui.R**



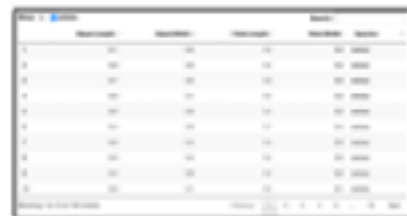
Question

```
01 ui ← fluidPage(  
02   selectInput(inputId = "city",  
03     label = "Select city",  
04     choices = c("edinburgh",  
05       "glasgow",  
06       "lausanne",  
07       "zurich")),  
08   strong("Selected city"),  
09   textOutput(outputId = "selected_city"),  
10 )
```

Would you expect this piece of code to result in an error? Why or why not?



Outputs



`DT::renderDataTable(expr, options, callback, escape, env, quoted)`

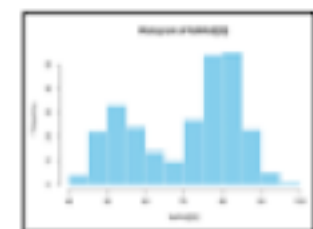


`dataTableOutput(outputId, icon, ...)`



`renderImage(expr, env, quoted, deleteFile)`

`imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)`



`renderPlot(expr, width, height, res, ..., env, quoted, func)`

`plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, hoverDelayType, brush, clickId, hoverId, inline)`

```
data.frame("a" = 1, "b" = 2)
#> # A tibble: 1 x 2
#>   a     b
#>   <dbl> <dbl>
#> 1     1     2
```

`renderPrint(expr, env, quoted, func, width)`

`verbatimTextOutput(outputId)`

Report Length	Report Width	Print Length	Print Width	Options
1	1.00	0.00	0.00	width
2	0.00	0.00	0.00	width
3	0.75	0.00	0.00	width
4	0.50	0.00	0.00	width
5	0.25	0.00	0.00	width
6	0.00	0.00	0.00	width

`renderTable(expr, ..., env, quoted, func)`

`tableOutput(outputId)`

foo

`renderText(expr, env, quoted, func)`

`textOutput(outputId, container, inline)`



`renderUI(expr, env, quoted, func)`

`uiOutput(outputId, inline, container, ...)`
& `htmlOutput(outputId, inline, container, ...)`

Shiny :: CHEAT SHEET

Basics

A Shiny app is a web page (UI) connected to a computer running a live R session (Server). Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {
  output$greet <- "Hi!"
}
shinyApp(ui, server)
```

Building an App

Complete the template by adding arguments to `fluidPage()` and a `render` function.

Add outputs with `render` functions. Tell server logic under `output` with the server function. To do this:

- Refer to outputs with `output$`.
- Wrap code in a `render` function before taking its output.
- Save your app as `app.R`. Alternatively, put your template into two files named `ui.R` and `server.R`.

Inputs

Access the current value of an input object with `input$`. Input values are **reactive**.

Outputs

`render` and `Output` functions work together to load R output to the UI.

SHARE YOUR APP

The easiest way to share your app is to host it on a server or a cloud-based service from RStudio.

1. Create a free or professional account at <https://shiny.shinyapps.io>

2. Click the **Publish** icon in the RStudio IDE server pane.

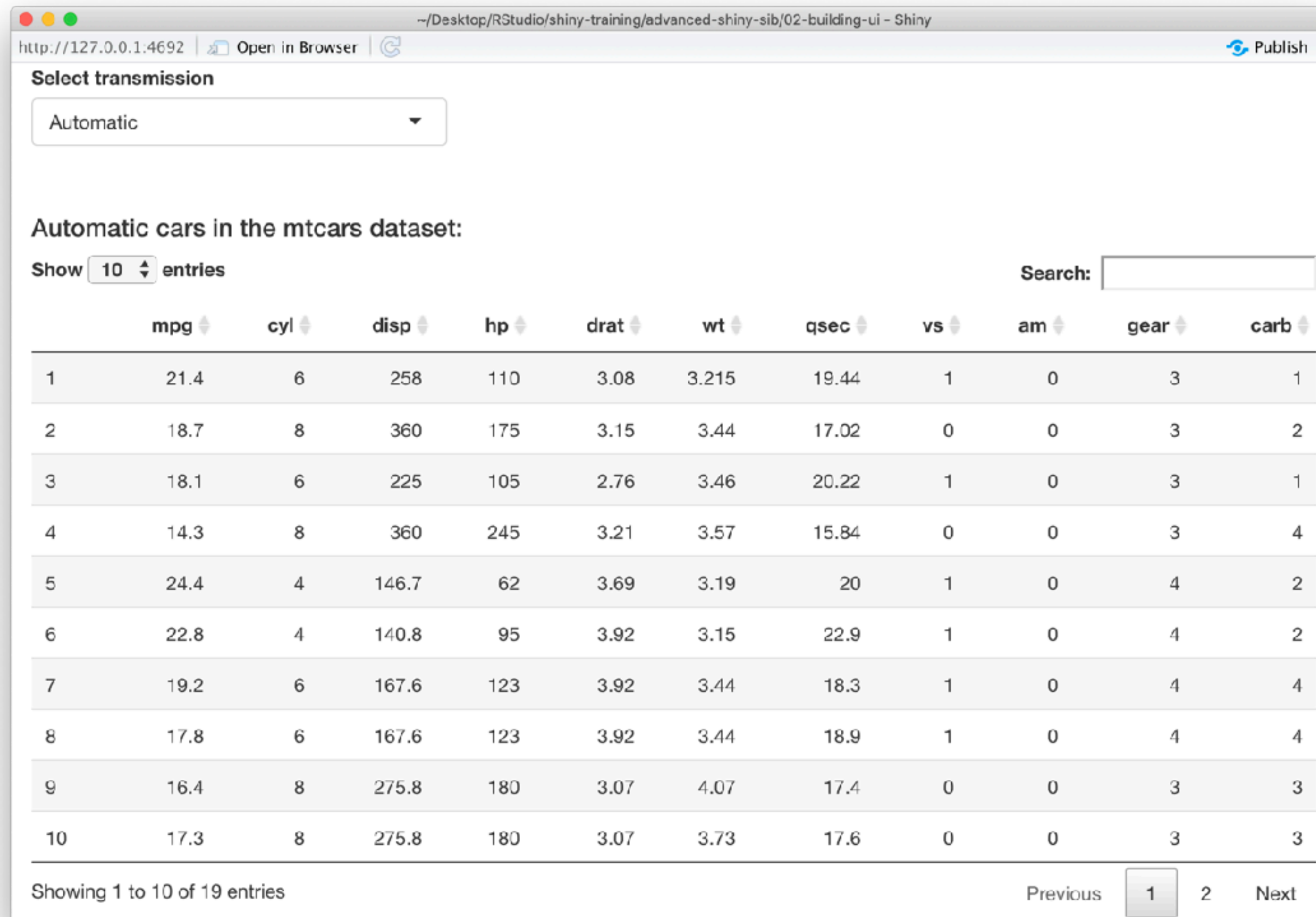
3. `reconnectShinyApp()` or `publishShinyApp()`

Build or purchase your own Shiny Server at www.rstudio.com/products/shiny-server/

RStudio

Which `render*` and `*Output` function duo is used to display this table?

Question



The screenshot shows a Shiny web application interface. At the top, there is a browser address bar with the URL `http://127.0.0.1:4692` and a "Publish" button. Below the browser bar, there is a "Select transmission" section with a dropdown menu set to "Automatic". Underneath, the text "Automatic cars in the mtcars dataset:" is displayed. To the left of the table, there is a "Show 10 entries" control, and to the right, there is a "Search:" input field. The table itself has 11 columns: an index column (1-10), `mpg`, `cyl`, `disp`, `hp`, `drat`, `wt`, `qsec`, `vs`, `am`, `gear`, and `carb`. The data is as follows:

	<code>mpg</code>	<code>cyl</code>	<code>disp</code>	<code>hp</code>	<code>drat</code>	<code>wt</code>	<code>qsec</code>	<code>vs</code>	<code>am</code>	<code>gear</code>	<code>carb</code>
1	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
2	18.7	8	360	175	3.15	3.44	17.02	0	0	3	2
3	18.1	6	225	105	2.76	3.46	20.22	1	0	3	1
4	14.3	8	360	245	3.21	3.57	15.84	0	0	3	4
5	24.4	4	146.7	62	3.69	3.19	20	1	0	4	2
6	22.8	4	140.8	95	3.92	3.15	22.9	1	0	4	2
7	19.2	6	167.6	123	3.92	3.44	18.3	1	0	4	4
8	17.8	6	167.6	123	3.92	3.44	18.9	1	0	4	4
9	16.4	8	275.8	180	3.07	4.07	17.4	0	0	3	3
10	17.3	8	275.8	180	3.07	3.73	17.6	0	0	3	3

At the bottom of the table, there is a pagination control showing "Showing 1 to 10 of 19 entries" and buttons for "Previous", "1", "2", and "Next".



Your turn

- ▶ Modify **02-building-ui** > **04-ui.R** to so that the table is displayed, but nothing else, i.e. remove the search, ordering, and filtering options.
- ▶ **Hint 1:** You'll need to read `?renderDataTable` and review the options at <https://rstudio.github.io/DT/options.html> and <https://datatables.net/reference/option>.
- ▶ **Hint 2:** Remember how many automatic and manual cars there are and make sure all are visible in the table output now that you don't have a way of scrolling through multiple pages.
- ▶ **Stretch goal:** Hide row numbers.

10_m 00_s



Solution

Solutions to the previous exercises

> **02-building-ui** > **05-ui.R**

> **02-building-ui** > **06-ui.R**



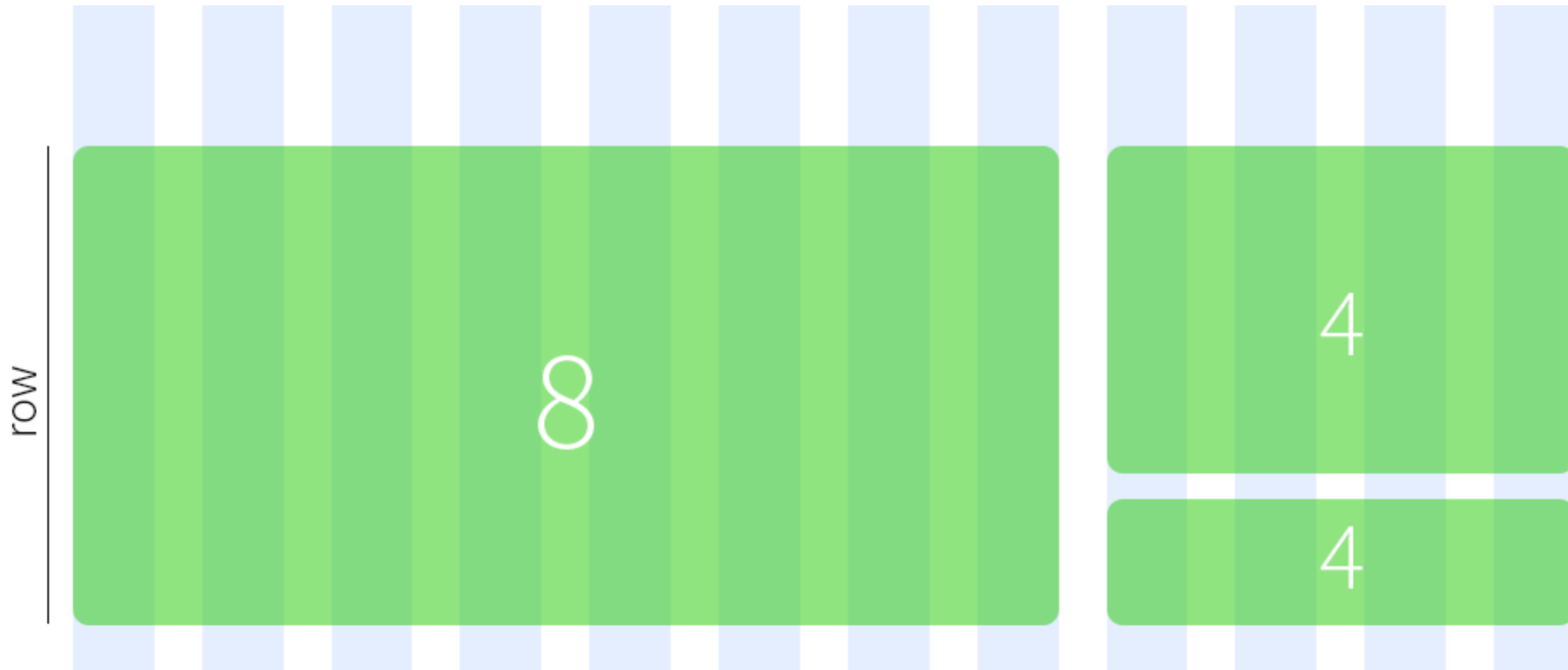
Layouts

Move beyond the familiar sidebar layout with facilities Shiny offers out of the box:

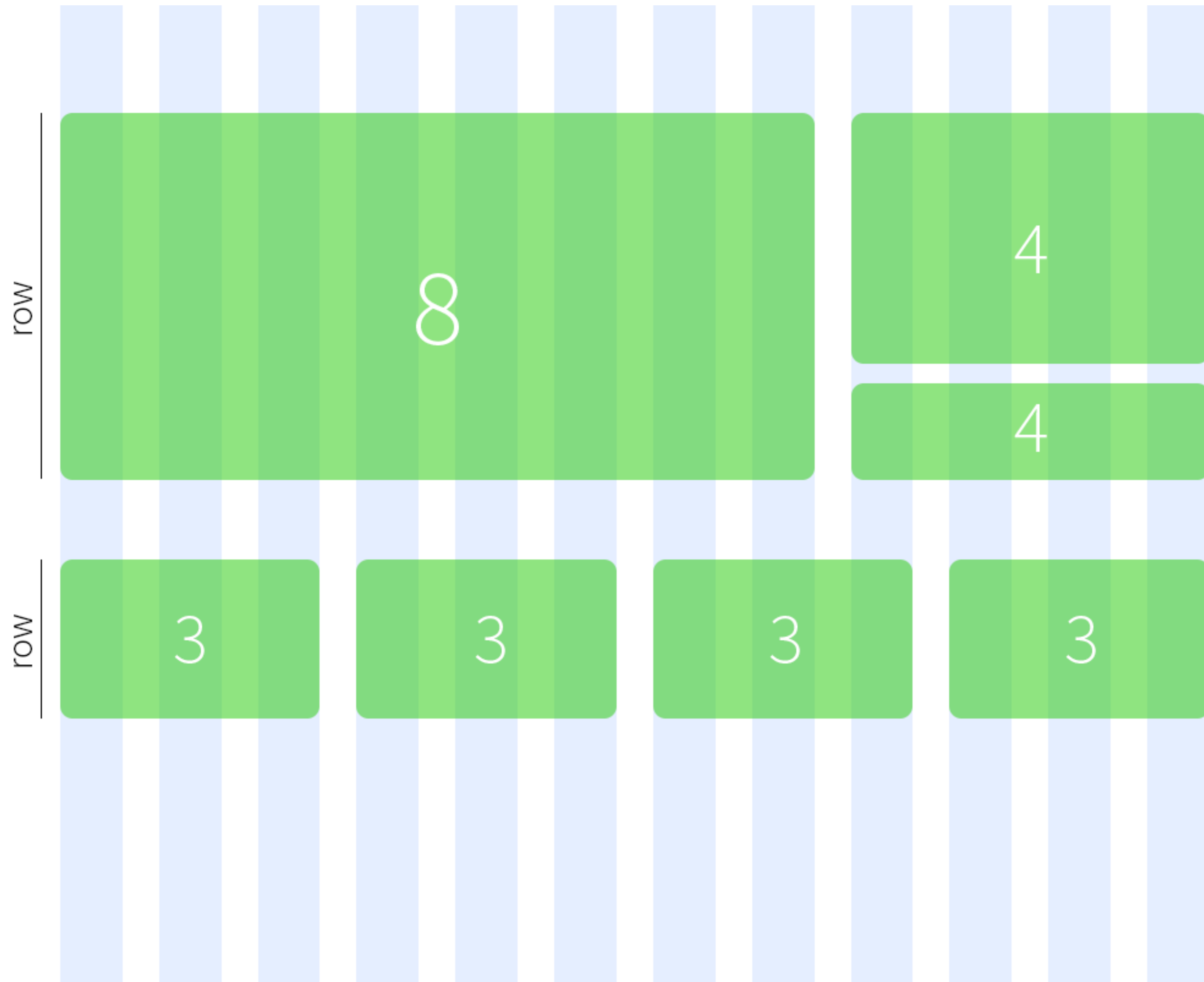
- **Bootstrap grid framework** – `fluidPage`, `fixedPage`, `fluidRow`, `column`
- **Containers** – `wellPanel`, `absolutePanel`, `fixedPanel`
- **Navigation panels** – `tabsetPanel`, `navlistPanel`, `navbarPage`
- **Fill layouts** – `fillPage`, `fillRow`, `fillCol`
- **Modals and notifications** – `showModal`, `modalDialog`

Bootstrap grid framework

- ▶ Every page has 12 invisible columns
- ▶ Each column of content must span an integral number of columns
- ▶ Simple R API for implementing Bootstrap grid
 - ▶ `fluidPage(...)` wraps the entire page
 - ▶ `fluidRow(...)` wraps each row's column
 - ▶ `column(width, ...)` wraps each column's content



```
ui ← fluidPage(  
  fluidRow(  
    column(8, item1),  
    column(4, item2, item3),  
  )  
)
```



```
ui ← fluidPage(  
  fluidRow(  
    column(8, item1),  
    column(4, item2, item3),  
  ),  
  fluidRow(  
    column(3, item4),  
    column(3, item5),  
    column(3, item6),  
    column(3, item7)  
  )  
)
```

Your turn

- ▶ Modify `02-building-ui` > `07-ui.R` to display the two outputs next to each other (instead of above and below).
- ▶ Assign the left output to be 5 columns wide, and the right output to be 7 columns wide.
- ▶ Observe what happens as you change the width of the browser window.
- ▶ **Stretch goal:**
 - ▶ What happens if you swap the order in which the two outputs are calculated in the server function?
 - ▶ What happens if the column widths don't add up to 12?

10_m 00_s



Solution

Solution to the previous exercise

> **02-building-ui** > **08-ui.R**





Functions from external packages

bslib

- Designed for use with any Shiny or R Markdown project that uses Bootstrap
- To use bslib in Shiny, provide a `bs_theme()` object to the theme parameter

```
library(shiny)
library(bslib)
ui ← fluidPage(
  theme = bs_theme(version = 5),
  ...
)
server ← ...
shinyApp(ui, server)
```

bslib + Bootswatch themes

Any Bootswatch theme is available through `bs_theme()`'s `bootswatch` argument

```
# Shiny usage
navbarPage(
  theme = bs_theme(
    bootswatch = "minty"
  ),
  ...
)
```

The screenshot displays a Shiny application interface titled "Theme demo" with navigation tabs for "Inputs", "Plots", "Tables", "Notifications", "Fonts", and "Options". The "Inputs" tab is active, showing several input widgets: a slider input with values 30 and 70, a selectize input with "AL" selected, a multi-select input with "AK", "AR", and "AL" selected, a date input with "2020-12-24", and a date range input with "2020-12-24 to 2020-12-31". Below the widgets, a console-like output shows the values bound to each widget. At the bottom, there are buttons for "Primary", "Secondary (default)", "Success", "Info", "Warning", "Danger", "Dark", and "Light". On the right side, a "Theme customizer" panel is open, showing options for "Main colors", "Overall theme" (set to "Minty"), "Background (bg) color" (#fff), and "Foreground (fg) color" (#000). The panel also includes sections for "Accent colors", "Fonts", "Options", and "Spacing".

Your turn

Modify **02-building-ui** > **08-ui.R** to use a Bootswatch theme:
bootswatch.com

5_m 00_s



Solution

Solution to the previous exercise

> **02-building-ui** > **09-ui.R**



bslib + thematic

Simplified theming of ggplot2, lattice, and base R graphics, with automatic styling of R plots in Shiny (as well as R Markdown and RStudio)

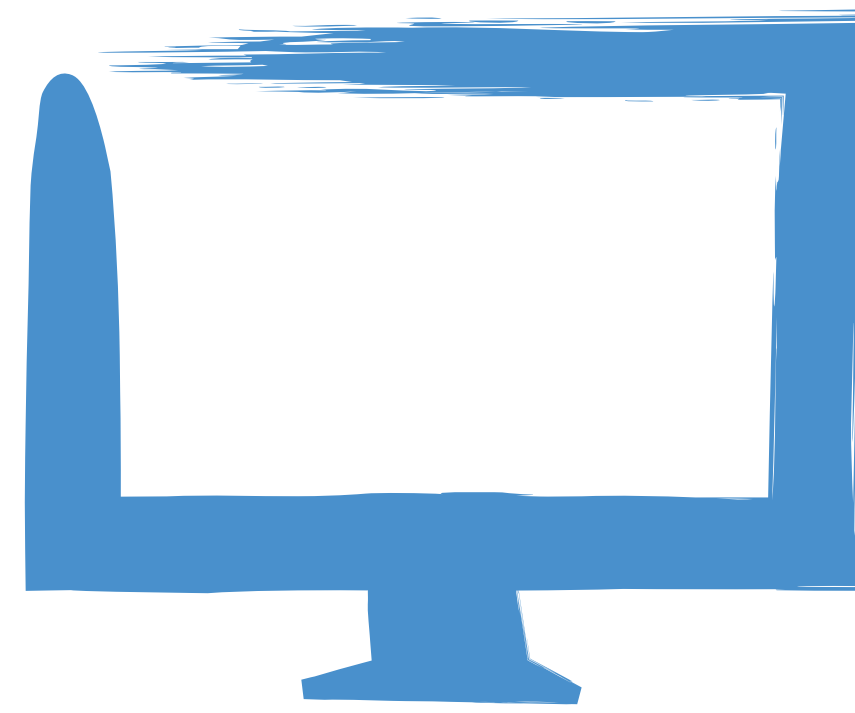
```
library(shiny)
library(thematic)
```

```
# Call thematic_shiny() prior to launching the app, to change
# R plot theming defaults for all the plots generated in the app
thematic_shiny()
```

```
ui ← ...
server ← ...
```

```
shinyApp(ui, server)
```

Open **02-building-ui** > **10-ui.R** and follow along



Other external packages

- shinythemes (Bootstrap 3 themes)

The image displays four overlapping screenshots of Shiny applications, each using a different theme from the shinythemes package:

- Darkly:** A dark-themed interface with a dark blue header and dark grey content area.
- United:** A light-themed interface with a white background and a prominent orange header.
- Flatly:** A light-themed interface with a dark blue header and a light grey content area.
- Navbar 1:** A light-themed interface with a dark blue header and a white content area.

Each theme screenshot shows the same set of UI components:

- File input:** A "Browse..." button and a "No file selected" message.
- Text input:** A text box containing the word "general".
- Slider input:** A slider with a range from 1 to 100, currently set to 30.
- Default action button:** A button labeled "Search".
- actionButton with CSS class:** A button labeled "Action button".

The Navbar 1 theme screenshot also includes a table and a verbatim text output:

speed	dist
4.00	2.00
4.00	10.00
7.00	4.00
7.00	22.00

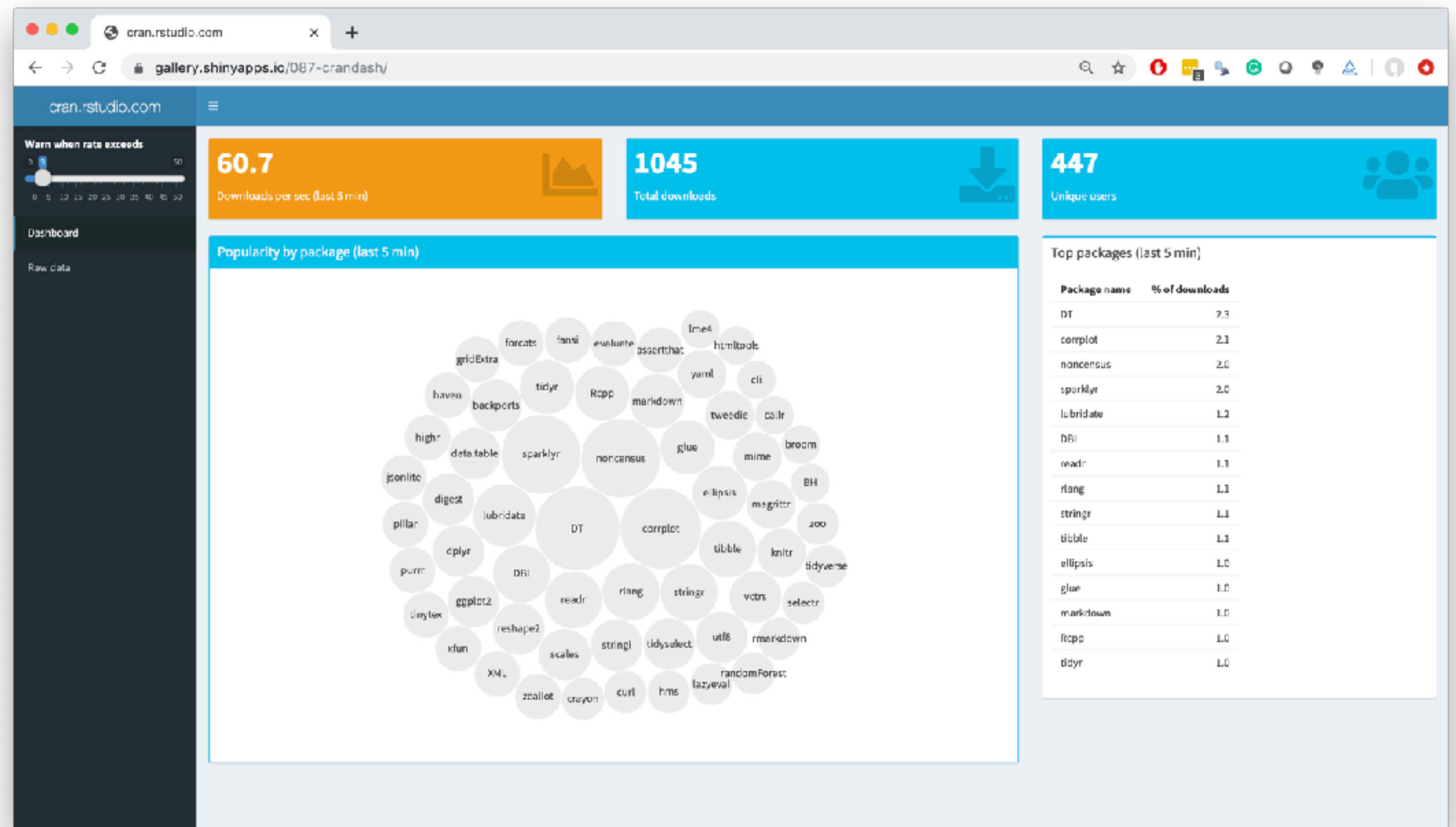
Verbatim text output: `general, 30, NULL`

Below the Navbar 1 screenshot, the following headers are visible:

- Header 4
- Header 5
- Header 3

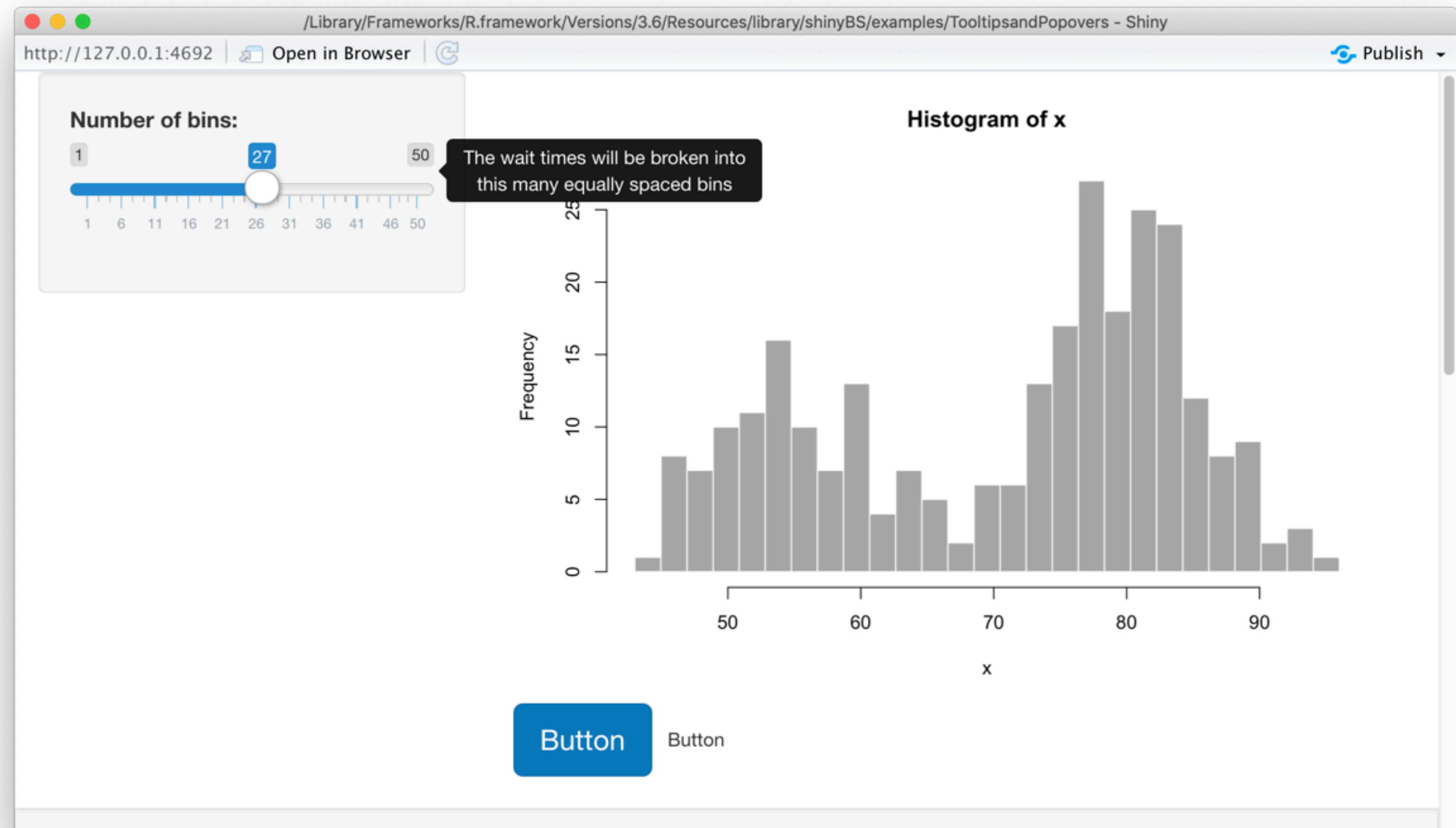
Other external packages

- ▶ shinythemes (Bootstrap 3 themes)
- ▶ shinydashboard



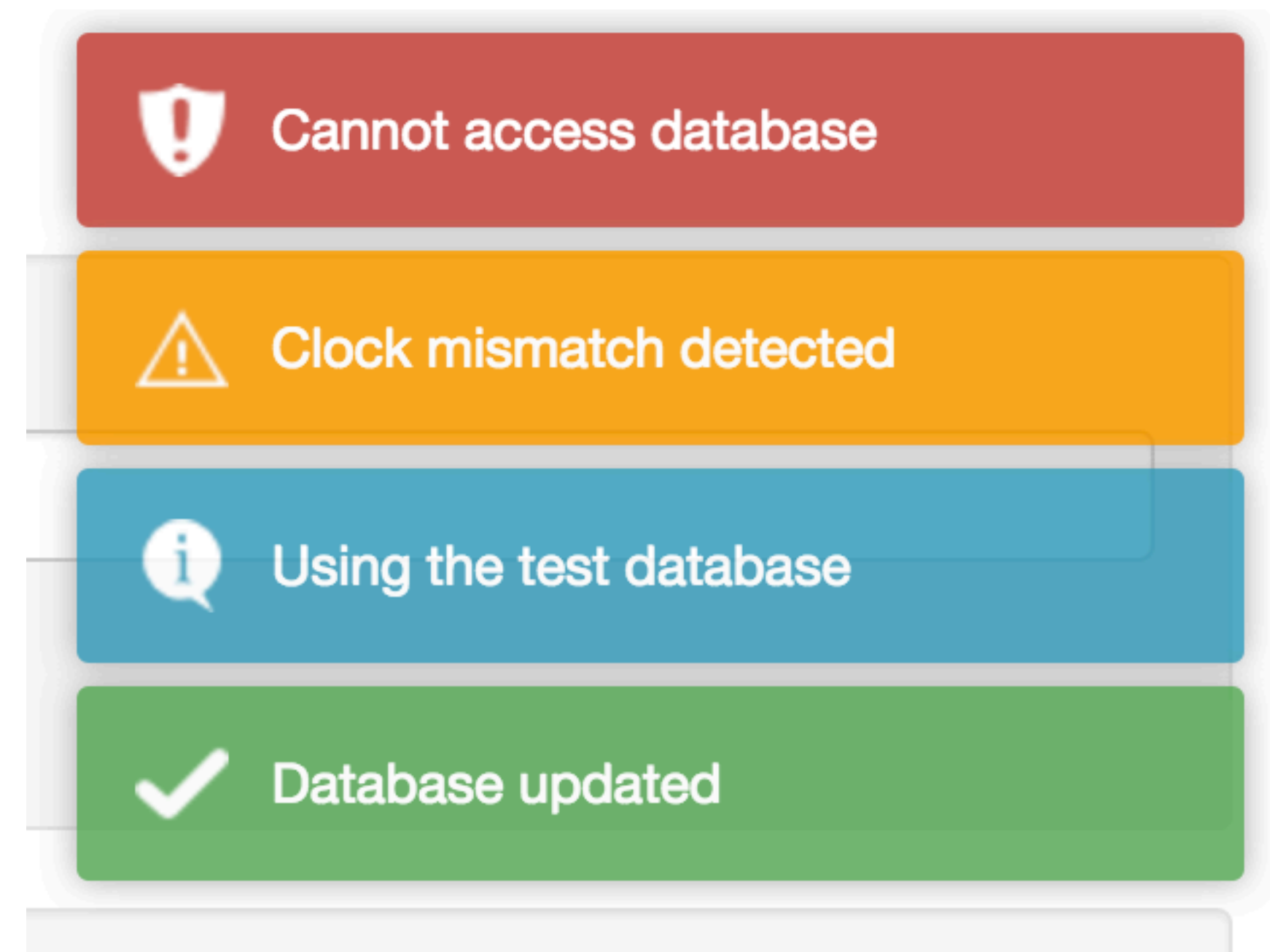
Other external packages

- ▶ shinythemes (Bootstrap 3 themes)
- ▶ shinydashboard
- ▶ shinyBS (@ebailey78)



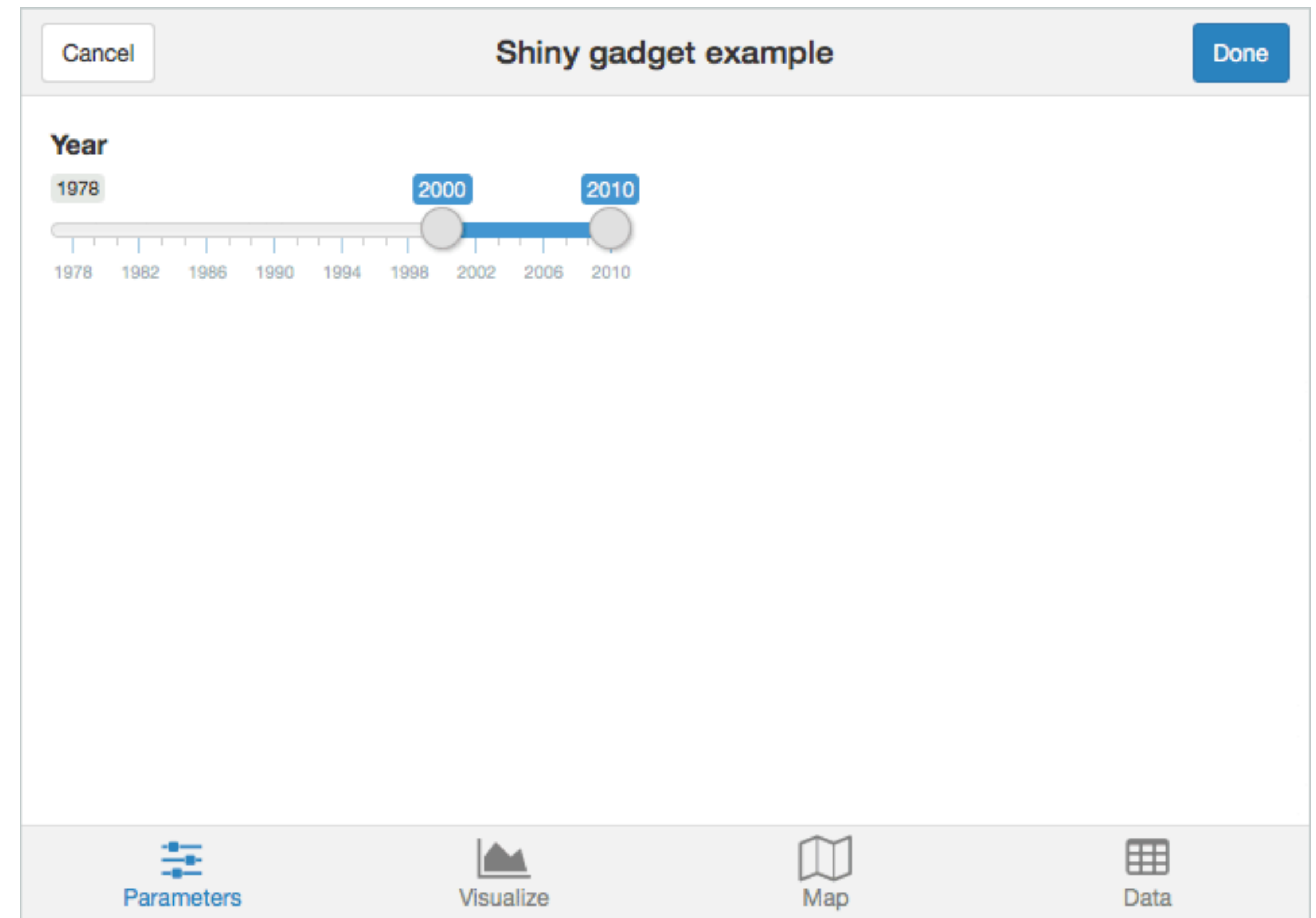
Other external packages

- ▶ shinythemes (Bootstrap 3 themes)
- ▶ shinydashboard
- ▶ shinyBS (@ebailey78)
- ▶ shinytoastr (@gaborcsardi)



Other external packages

- ▶ shinythemes (Bootstrap 3 themes)
- ▶ shinydashboard
- ▶ shinyBS (@ebailey78)
- ▶ shinytoastr (@gaborcsardi)
- ▶ miniUI (for mobile devices or Shiny Gadgets)



Other external packages

- ▶ shinythemes (Bootstrap 3 themes)
- ▶ shinydashboard
- ▶ shinyBS (@ebailey78)
- ▶ shinytoastr (@gaborcsardi)
- ▶ miniUI (for mobile devices or Shiny Gadgets)
- ▶ shinyjs (@daattali, perform many UI-related JavaScript operations from R)

Function	Description
<code>show / hide / toggle</code>	Display or hide an element (optionally with an animation).
<code>hidden</code>	Initialize a Shiny tag as invisible (can be shown later with <code>show</code>).
<code>enable / disable / toggleState</code>	Enable or disable an input element, such as a button or a text input.
<code>disabled</code>	Initialize a Shiny input as disabled.
<code>reset</code>	Reset a Shiny input widget back to its original value.
<code>delay</code>	Execute R code (including any <code>shinyjs</code> functions) after a specified amount of time.
<code>alert</code>	Show a message to the user.
<code>html</code>	Change the text/HTML of an element.
<code>onclick</code>	Run R code when a specific element is clicked. Was originally intended for running a <code>shinyjs</code> function when clicked, though any R code can be used.
<code>onevent</code>	Similar to <code>onclick</code> , but can be used with many other events (for example, listen for a key press, mouse hover, etc).
<code>addClass / removeClass / toggleClass</code>	add or remove a CSS class from an element.
<code>runjs</code>	Run arbitrary JavaScript code.



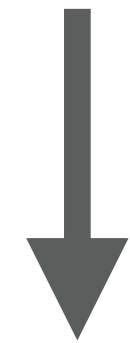
Tag objects and UI functions

An API for Composing HTML

- ▶ When Shiny was born, it came with a sub-package for composing HTML.
- ▶ These functions were so useful, they were extracted out into a separate package: **htmltools**.
- ▶ Now used by rmarkdown and htmlwidgets as well.

HTML basics

```
<a href="https://www.rstudio.com">RStudio</a>
```



[RStudio](https://www.rstudio.com)

HTML basics

```
<a href="https://www.rstudio.com">RStudio</a>
```

End tag

Start tag

Child content

Anatomy of a tag

Attribute name

```
<a href="https://www.rstudio.com">RStudio</a>
```

Tag name

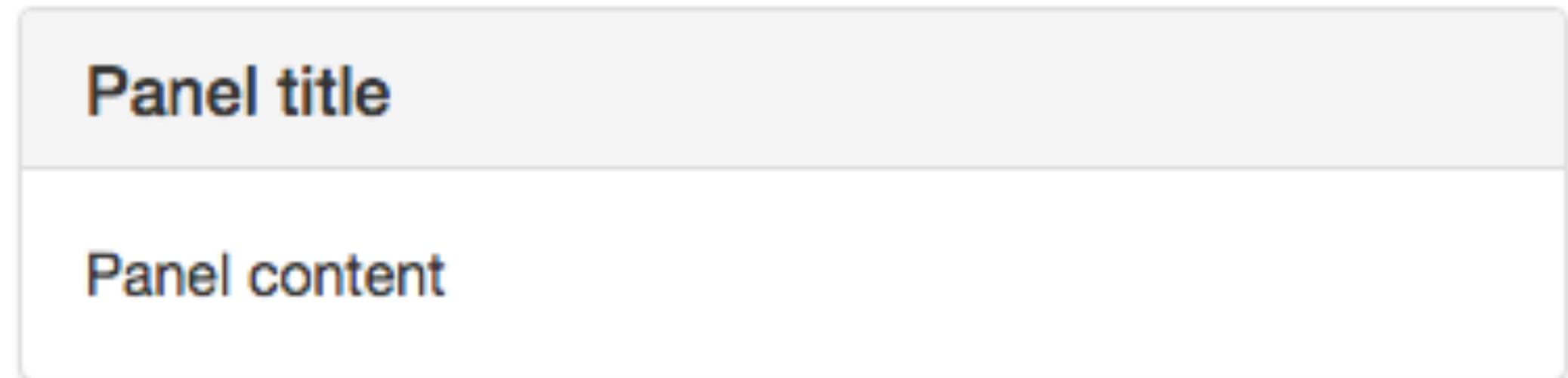
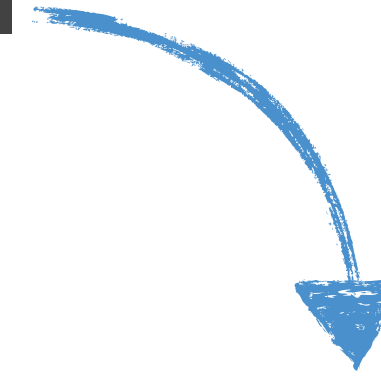
Attribute value

Creates an **anchor** whose **hyperlink reference** is the URL `https://www.rstudio.com`.

Anatomy of a tag

- ▶ Text can contain tags
- ▶ Tags can optionally contain text and/or other tags
- ▶ Each start tag can have zero or more attributes

```
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">Panel title</h3>
  </div>
  <div class="panel-body">
    Panel content
  </div>
</div>
```



Looks like R, means HTML

```
<div class="panel panel-default">
  <div class="panel-heading">
    <h3 class="panel-title">
      Panel title
    </h3>
  </div>
  <div class="panel-body">
    Panel content
  </div>
</div>
```

```
01 div(class="panel panel-default",
02     div(class="panel-heading",
03         h3(class="panel-title",
04             "Panel title",
05         )
06     ),
07     div(class="panel-body",
08         "Panel content"
09     )
10 )
```

Using tag functions

- ▶ All tag functions behave the same way:
 - ▶ Call the function to create a tag object
 - ▶ Named arguments become attributes
 - ▶ Unnamed arguments become children
- ▶ Many common tags are exported as functions by `htmltools` and `shiny` (`p`, `h1-h6`, `a`, `br`, `div`, `span`, `img`).
- ▶ All other tags can be accessed via the `tags` object.
- ▶ If you have lots of HTML to write, you can use the `withTags` function — it makes the `tags$` prefix optional.

```
<li>Item 1</li>
```



```
tags$li("Item 1")
```

```
withTags(  
  ul(  
    li("Item 1"),  
    li("Item 2")  
  )  
)
```

Tag attributes

- ▶ Any valid HTML attribute name can be used (use quotes if the name has dashes, e.g.
"data-toggle" = "dropdown")
- ▶ Valid tag attribute values are:
 - ▶ NULL (omit the attribute)
 - ▶ NA (the attribute should be included with no value)
 - ▶ Single-element character vector (or something to be coerced to character)

```
01 tags$input(type = "checkbox",  
02           disabled = if (disabled) NA # else NULL  
03           )
```

Tag children

- Valid tag children are:
 - Tag objects
 - Single-element character vectors (treated as text)
 - NULL (silently ignored)
 - Raw HTML (see `?htmltools::HTML`)
 - Lists of valid tag children

Using tags

- Tags are made using normal R functions that take normal parameters and return normal values! You can do R-like things to them:

```
tags$ul(lapply(1:10, tags$li))
```

- Print tag objects at the console to see their HTML source
 - Call `print(x, browse = TRUE)` to see their rendered view instead
 - Use `htmltools::browsable()` to make an object show its rendered view when printed, by default
 - If your top-level object is a list, you'll need to wrap in `tagList(...)` to get the right behaviour at the console (or in an R Markdown doc)

Your turn

- ▶ Open `02-building-ui` > `11-ui.R`.
- ▶ Replace `includeHTML("youtube_thumbnail.html")` with the equivalent `htmltools` tag objects. **Hint:** Take a look inside `youtube_thumbnail.html`. Also, run `includeHTML("youtube_thumbnail.html")` in the console and take a look at the raw HTML code it generates.
- ▶ **Stretch goal:** If you get that working, take the next step and define an R function that takes a YouTube URL, a title, and a description, and returns a thumbnail frame like the one you created.

10_m 00_s



Solution

Solution to the previous exercise

> **02-building-ui** > **12-ui.R**

> **02-building-ui** > **13-ui.R**





Using raw html

- ▶ Incorporate tiny amounts of HTML using inline string literals wrapped in `HTML()`
 - ▶ `div(HTML("This is HTML"))`
- ▶ For chunks of (static) HTML, use `includeHTML` (or similar `includeCSS`, `includeScript`)
 - ▶ `div(includeHTML("file.html"))`
- ▶ Or go the other way, with the HTML Templates feature: start with HTML, and embed R expressions that yield tag objects
 - ▶ Read more at shiny.rstudio.com/articles/templates.html

Building user interfaces



Mine Çetinkaya-Rundel

@minebocek 
mine-cetinkaya-rundel 
mine@rstudio.com 