Shiny

Part II

Laurent R. Bergé
B_xSE, University of Bordeaux
Fall 2023

UI modifications

Basic

Dynamically changing the user inputs

- you can seamlessy modify the UI from the server using the `updateTYPE` family of functions.

Dynamically changing the user inputs

- you can seamlessy modify the UI from the server using the `updateTYPE` family of functions.

```r
# ui.R
selectInput("select_variable",
            "Select the variable:",
            "Not yet defined"),
```

```r
# server.R
updateSelectInput(session,
                  "select_variable",
                  choices = names(faithful))
```

Exercise

- let the user choose which variables to represent using radio buttons

CSS

UI modifications

CSS

Javascript

One app = One function

Adding custom CSS

Since we're producing HTML files, you can use whatever "web-stuff" you want.

In particular you can use CSS to style anything you want in the page.

Adding custom CSS

Since we're producing HTML files, you can use whatever "web-stuff" you want.

In particular you can use CSS to style anything you want in the page.

How to

There are three main ways to include custom CSS:

1. using the style argument of HTML elements
2. including CSS directly in the header (head tag)
3. writing CSS in a separate script and including it

Adding CSS

the `style` argument

- when including a CSS element, simply add CSS code in the `style` argument

Code

```
tags$div("My name is Bond.",
         style = "font-family: 'Courier New';
                  color: Brown;
                  font-weight: bold;")
```

Result

My name is bond

Adding CSS

the `style` argument

- when including a CSS element, simply add CSS code in the `style` argument

Code

```
tags$div("My name is Bond.",
         style = "font-family: 'Courier New';
                  color: Brown;
                  font-weight: bold;")
```

Result

My name is bond

When?

Only useful for non-repeated minor styling.

Adding CSS

in the header

Add CSS directly in the HTML header.

Code

```
fluidPage(
  tags$head(
    tags$style("
    p {
      background-color: #ffe;
      font-family: Roboto Slab;
      font-size: 2rem;
    }
    ")
  ),
  p("Hello World")
)
```

Result

Hello World

Adding CSS
in the header

Add CSS directly in the HTML header.

Code

```
fluidPage(
  tags$head(
    tags$style("
    p {
      background-color: #ffe;
      font-family: Roboto Slab;
      font-size: 2rem;
    }
    ")
  ),
  p("Hello World")
)
```

Result

Hello World

When?

When you have only a handful of styling to do.

Adding CSS
in a separate file

- write the CSS code in a file located in the `www/` folder★
- import the CSS code by creating a link to this file in the header:

```
fluidPage(
  tags$head(
    tags$link(rel = "stylesheet",
              type = "text/css",
              href = "my-style.css")
  ),
  p("Hello World")
)
```

Note that there is no 'www/' here!

★: the `www/` folder should be located where the app files are. Create it if it does not exist.

Adding CSS
in a separate file

- using custom CSS in a separate file should be the way to go for most projects, unless your styling is super tiny and fit the two previous cases

Adding CSS
in a separate file

- using custom CSS in a separate file should be the way to go for most projects, unless your styling is super tiny and fit the two previous cases

NOTA

If your app does not update when you modify the CSS code, this is normal (this file is not tracked).

To apply the modifications:

1. open the app on a web browser
2. force reload with `Ctrl + F5`

Exercise

Using our distribution app:

- change the font to `Fira Sans`★
- use the background style of the body defined at this link
- add an icon before the tab name which is currently active (see next slides for help)

★: See Google Fonts for many available fonts and how to import them.

CSS tips: `before` and `after`

Assume this HTML:

```
<p class = "hi">
  Hello
</p>
```

Question:

Can you add the word “World” after “Hello” just with CSS?

CSS tips: `before` and `after`

Assume this HTML:

```
<p class = "hi">
  Hello
</p>
```

Question:

Can you add the word “World” after “Hello” just with CSS?

Yes! With the `::after` selector.

CSS ::after

Code

```
/* CSS code */
p.hi::after {
  content: " World"
}
```

Result

HTML

```
<p class = "hi">
  Hello
  ::after == $0
</p>
```

Displays as

Hello World

Back to the exercise

For the previous exercise, you only need to find the right selector!

Javascript

UI modifications

CSS

Javascript

One app = One function

Adding JS

Two ways to include JS code:

1. add the code in the header directly
2. write the code in a separate script

Adding JS
in the header

Write the code in the header using `tags$script`:

```
tags$head(
  tags$script(HTML("
  document.body.style.backgroundColor = 'AliceBlue';
  "))
)
```

When?

When your code base is tiny and to the point. Not the place for functions.

Where to place JS code?

Note that you can place JS code anywhere on the page: you don't need to place it in the `head`.

However, it's usually good practice to do so since the code is more tractable and easier to maintain.

Adding JS

in a separate file

Like for CSS stylesheets, you can write your JS code in a separate file located in the `www/` folder. Please note:

- the path of the file should be relative to the `www/` folder
- like for direct inclusion, you use the `script` tag, but this time you use the `src` attribute

```
tags$head(
  tags$script(src = "my-script.js")
)
```

Adding JS in a separate file

This should be norm and will make your life easy:

- you can use a dedicated editor to handle the JS★
- easier to track changes
- easier to maintain
- only way to handle a growing code base

★: It will spot syntax errors, provide autocompletion and contextual documentation, etc.

Why using JS in Shiny?

- javascript can manipulate anything on a webpage, in any arbitrary way: you will need it when you want to implement advanced behaviors, in particular interactivity with the user
- many tools that you use (in particular in shiny itself) use JS behind the scenes

JS how: buttons and clicks

Attach a function to a button using the `onclick` attribute.

Let's create a function that changes the background of a button when clicked.

```
<script>
  // we define the function that will be run by the click
  function changeColor() {
    let my_btn = document.getElementById("id-btn")
    my_btn.style.backgroundColor = 'AliceBlue'
  }
</script>

<button id = "id-btn" onclick = "changeColor()">
  That's a button.
</button>
```

JS how: trigger functions when events occur

You can attach a function to any event occurring in the webpage using event listeners.

Let's recreate the previous example with an event listener. Plus: let's do it on a `div` and not a `button`.

```
<div id = "div-btn">
  This div works like a button. Click me.
</div>

<script>
  // we define the function that will be run by the click
  function changeColor() {
    this.style.backgroundColor = 'AliceBlue'
  }

  // we attach a function to a click event on the div
  let div_btn = document.getElementById("div-btn")
  div_btn.addEventListener("click", changeColor)
</script>
```

JS and event listeners

In the previous example:

- I could attach a click event on a `div` even though it's not a button!
- I needed to access the object (using `document.getElementById('div-btn')`) to attach it the "click" event
- in the function, I could use `this` to refer to the current object from which the function was fired (not possible with the `onclick`)

Useful event listeners

Ususally you want to apply your javascript on the page once it's loaded. Use the following code:

```
window.addEventListener("load", function_to_run)
```

... with the function `function_to_run` containing all the necessary code that will be applied to the full web page once it's loaded.

Exercise

- on the top right corner of each *saved graph*, add a button to delete the graph (i.e. remove it from the webpage)

Steps:

1. embed the graph in a `div` which will also contain a button, folllowing the graph
2. assign an unique id to the `div` container and add the following argument `style = position: relative`
3. assign the class `btn-delete` to the button and style it with CSS (add `position: absolute; top: 0px; right: 0px;`)
4. using `onclick`, attach a function removing the div

Alternative exercise

- replicate the previous exercise without using JS directly but using shiny's `removeUI` function

One app = One function

UI modifications

CSS

Javascript

One app = One function

Running the app

So far, to run the app we needed to:

- go to a file from the app: either `ui.R` or `server.R`
- click on `Launch App`

Q: Could we run the app from the console?

Running the app

So far, to run the app we needed to:

- go to a file from the app: either `ui.R` or `server.R`
- click on `Launch App`

Q: Could we run the app from the console?

A: Nope!

Writing app-functions

To launch an app from the console, run:

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

... with `ui` a variable containing the UI and `server` a variable containing the server-side function.

App-function: minimal example

```
min_ui = fluidPage(
  titlePanel("My simple app")
)

min_server = function(input, output, session){
  # nothing
}

shinyApp(min_ui, min_server)
```

App-function: problem with the previous approach

... and solution

You need to write `shinyApp(min_ui, min_server)` to call your app, that's not really handy.

App-function: problem with the previous approach

... and solution

You need to write `shinyApp(min_ui, min_server)` to call your app, that's not really handy.

Solution

Embedd it in a function.

```
min_app = function(){
  shinyApp(min_ui, min_server)
}
```

App-function: issues so far

There are two main problems with the previous app-function (`min_app`):

1. it does not accept arguments and really... how to pass arguments to the shiny app???
2. since now the call is independent from a file location: how can we make the app look after the CSS and the JS in the `www/` folder???

App-function: passing arguments

You can use R `options` to pass arguments:

```
greet_ui = fluidPage(textOutput("text_hello"))

greet_server = function(input, output, session){
  output$text_hello = renderText(getOption("greet_text"))
}

greet_app = function(name){
  options(greet_text = paste0("Hello ", name))
  shinyApp(greet_ui, greet_server)
}

greet_app("Anna-Lisa")
```

App-function: finding files in `www/`

To make shiny access files located in a `www/` folder (typically CSS and JS files, but can also be images):

- run `shiny::addResourcePath("www", "path_to_www"))` somewhere in the code
- this will give the app access to the `www/` folder
- beware, contrary to before, now `www/` must appear in the file path

App using `www/` files: example

R code

```
app_ui = fluidPage(
  tags$head(
    tags$link(rel = "stylesheet",
              type = "text/css",
              href = "www/my-style.css")
  ),

  titlePanel("Basic App, with style")
)

app_server = function(input, output, session) {
    # nothing
}

my_app = function() {
  my_www = file.path("./shiny/app-fun/www")
  shiny::addResourcePath("www", my_www)
  on.exit(shiny::removeResourcePath("www"))

  shiny::shinyApp(app_ui, app_server)
}
```

CSS code

(location: "`./shiny/app-fun/www/my-style.css`")

```
@import url('https://fonts.googleapis.com/css2?family=
Silkscreen&display=swap');

body {
  background-color: aliceblue;
  font-family: 'Silkscreen', sans-serif;
}
```

Result

Commment on the previous code

```
my_app  =  function()  {
    my_www  =  file.path("./shiny/app-fun/www")
    shiny::addResourcePath("www",  my_www)
   on.exit(shiny::removeResourcePath("www"))

    shiny::shinyApp(app_ui,  app_server)
}
```

The highlighted line ensures the resource is removed once we leave the function (i.e. when the app is closed). It will work even if the function is stopped from running, as is the case with an app.

This is to avoid possible conflicts when running multiple apps.

Conclusion

That's it folks!

Although it's just an introduction, it should be enough to help you make a fancy shiny app!