Web apps in R with Shiny

2019-08-29

apps/goog-index/app.R

Web apps in R

Reactivity

Web apps in R

Reactivity

Design and User Interface (UI)

Web apps in R

Reactivity

Design and User Interface (UI)

Dashboards

Your Turn 1

Open a new Shiny file (file > New File > Shiny Web App)

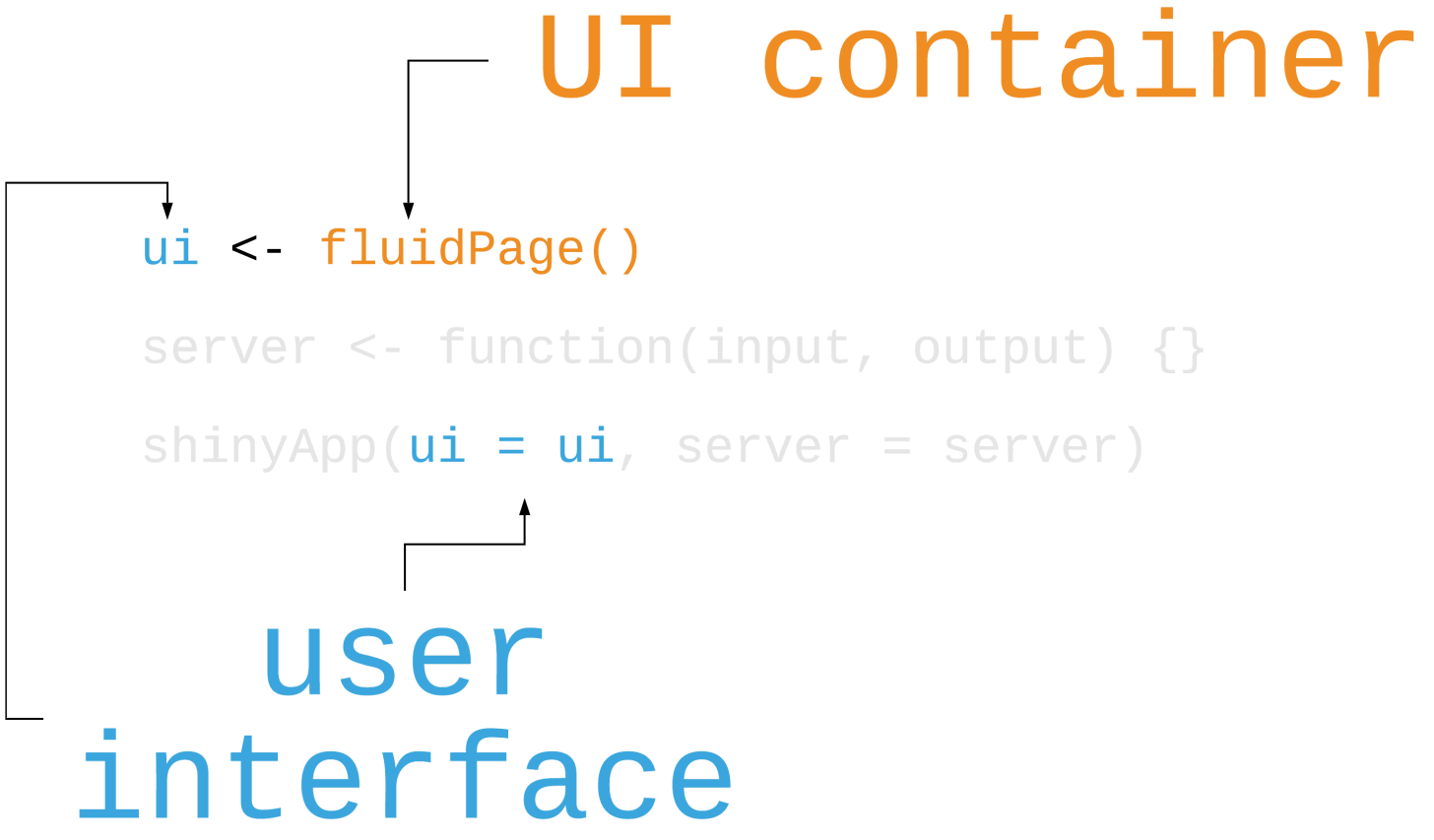
Run the app

Stop the app from running

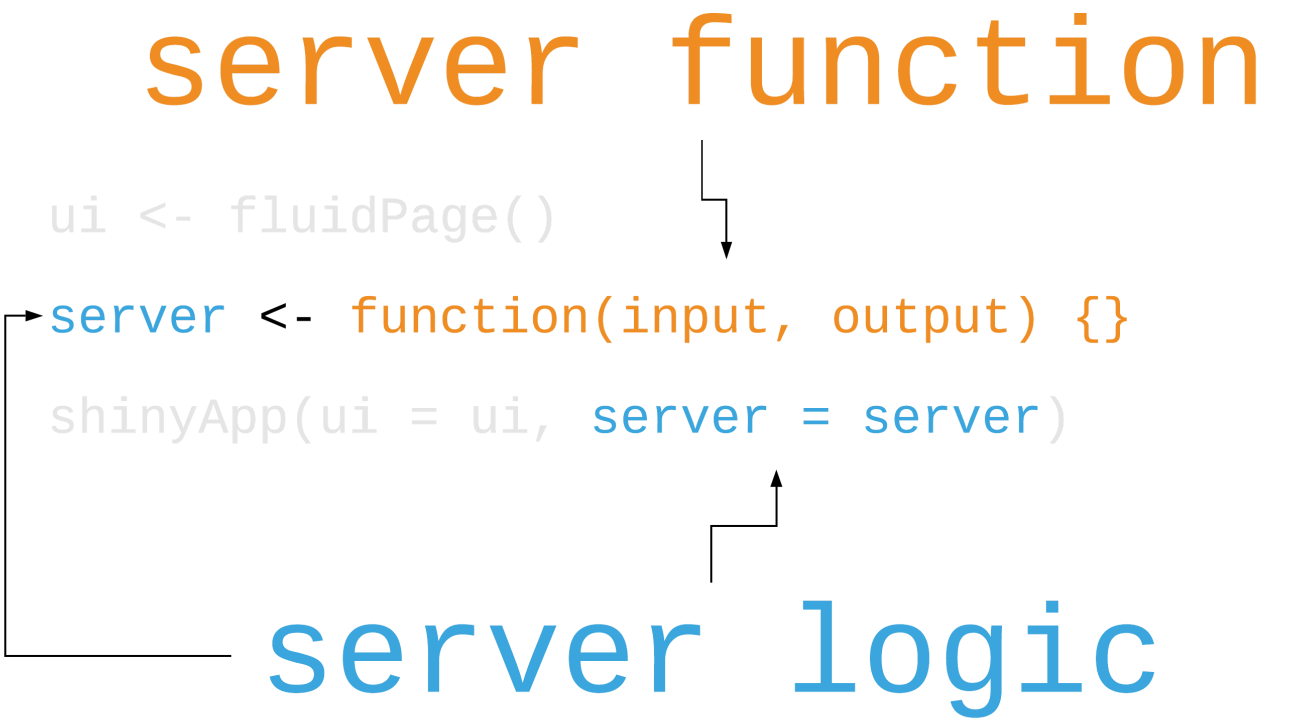
Anatomy of a shiny app

```
ui <- fluidPage()

server <- function(input, output) {}

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

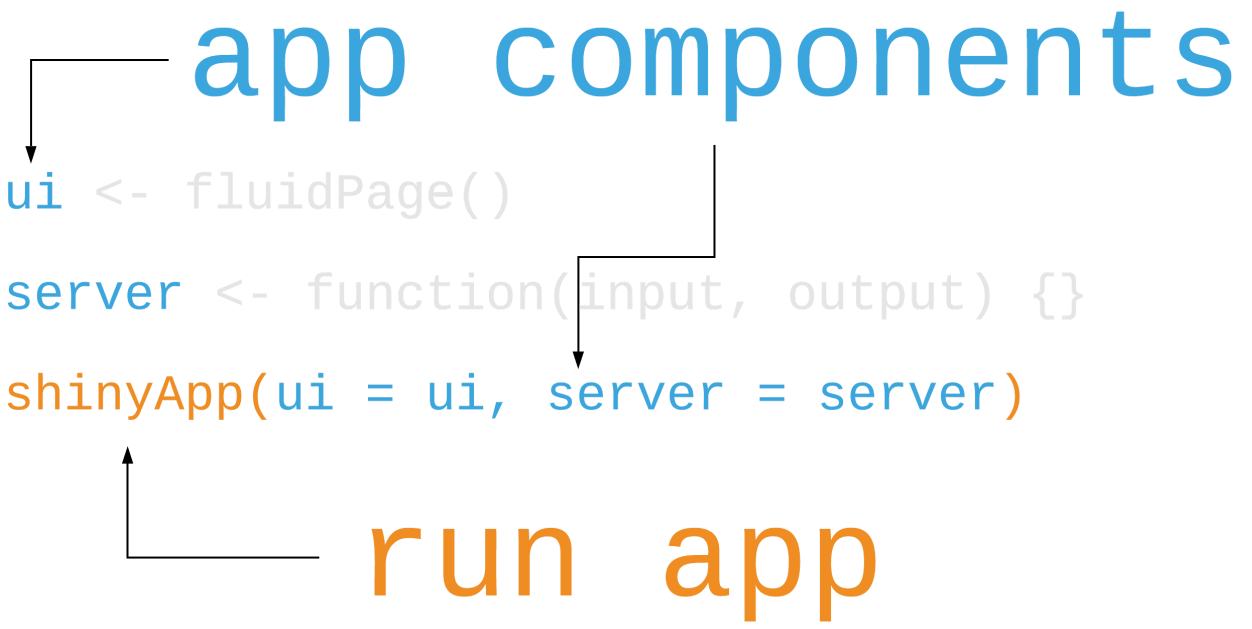
Anatomy of a shiny app



Anatomy of a shiny app



Anatomy of a shiny app



new data alert!

movies

	title	title_type	genre	runtime	mpaa_rating	stud
1	Filly Brown	Feature Film	Drama	80	R	Indo
2	The Dish	Feature Film	Drama	101	PG-13	Warr
3	Waiting for Guffman	Feature Film	Comedy	84	R	Sony
4	The Age of Innocence	Feature Film	Drama	139	PG	Colu
5	Malevolence	Feature Film	Horror	90	R	Anch
6	Old Partner	Documentary	Documentary	78	Unrated	Shca
7	Lady Jane	Feature Film	Drama	142	PG-13	Para
8	Mad Dog Time	Feature Film	Drama	93	R	MGM
9	Beauty Is Embarrassing	Documentary	Documentary	88	Unrated	Inde
10	The Snowtown Murders	Feature Film	Drama	119	Unrated	IFC F
11	Superman II	Feature Film	Action & Adventure	127	PG	Warr
12	Leap of Faith	Feature Film	Drama	108	PG-13	Para
13	The Royal Tenenbaums	Feature Film	Comedy	110	R	Buer
14	School for Scoundrels	Feature Film	Comedy	100	PG-13	MGM
15	Rhinestone	Feature Film	Comedy	111	PG	20th
16	Burn After Reading	Feature Film	Drama	96	R	Focu
17	The Doors	Feature Film	Drama	140	R	Sony
18	The Wood	Feature Film	Drama	106	R	Para
19	Jason X	Feature Film	Horror	91	R	New
20	Dragon Wars	Feature Film	Drama	90	PG-13	Sony
21	Fallen	Feature Film	Drama	124	R	Warr
22	The Gleaners and I	Documentary	Documentary	82	Unrated	Zeitg

Where does it come from?

`movies.Rdata`

How can I use it?

```
load("movies.Rdata")
View(movies)
```

this loads it in your global environment

Your Turn 2

Open apps/movies_01.R

Try to identify the components of the app

Run the app

Stop the app

movies_01.R

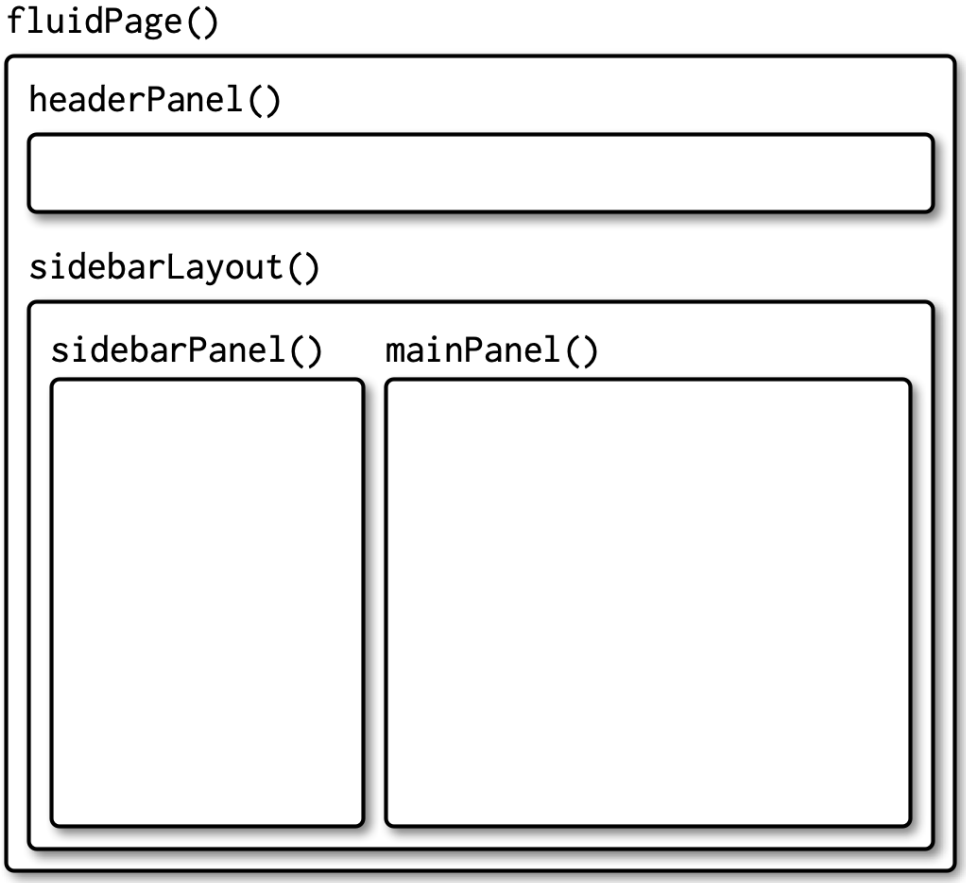


Image by Hadley Wickham

Sidebar layouts

```
ui <- fluidPage(
  headerPanel(),
  sidebarLayout(
    sidebarPanel(
      # Inputs
    ),
    mainPanel(
      # Outputs
    )
  )
)
```

Sidebar inputs

```
sidebarPanel(
  selectInput(
    inputId = "y",
    label = "Y-axis:",
    choices = c("..."),
    selected = "audience_score"
  ),
  selectInput(
    inputId = "x",
    label = "X-axis:",
    choices = c("..."),
    selected = "critics_score"
  )
)
```

Sidebar inputs

```
sidebarPanel(
  selectInput(
    inputId = "y",
    label = "Y-axis:",
    choices = c("..."),
    selected = "audience_score"
  ),
  selectInput(
    inputId = "x",
    label = "X-axis:",
    choices = c("..."),
    selected = "critics_score"
  )
)
```

Sidebar inputs

```
sidebarPanel(
  selectInput(
    inputId = "y",
    label = "Y-axis:",
    choices = c("..."),
    selected = "audience_score"
  ),
  selectInput(
    inputId = "x",
    label = "X-axis:",
    choices = c("..."),
    selected = "critics_score"
  )
)
```

Main panel outputs

```
mainPanel(
  plotOutput(outputId = "scatterplot")
)
```

Main panel outputs

```
mainPanel(
  plotOutput(outputId = "scatterplot")
)
```

Server

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(
      data = movies,
      aes_string(x = input$x, y = input$y)
    ) +
      geom_point()
  })
}
```

Server

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(
      data = movies,
      aes_string(x = input$x, y = input$y)
    ) +
      geom_point()
  })
}
```

Main panel outputs

```
mainPanel(
  plotOutput(outputId = "scatterplot")
)
```

Server

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(
      data = movies,
      aes_string(x = input$x, y = input$y)
    ) +
      geom_point()
  })
}
```

Server

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(
      data = movies,
      aes_string(x = input$x, y = input$y)
    ) +
      geom_point()
  })
}
```

Sidebar inputs

```
sidebarPanel(
  selectInput(
    inputId = "y",
    label = "Y-axis:",
    choices = c("..."),
    selected = "audience_score"
  ),
  selectInput(
    inputId = "x",
    label = "X-axis:",
    choices = c("..."),
    selected = "critics_score"
  )
)
```

Server

```
server <- function(input, output) {
  output$scatterplot <- renderPlot({
    ggplot(
      data = movies,
      aes_string(x = input$x, y = input$y)
    ) +
      geom_point()
  })
}
```

Run the app

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

Your Turn 3

Add new select menu to color the points. Use the following arguments: inputId = "z", label = "Color by:", choices = c("title_type", "genre", "mpaa_rating", "critics_rating", "audience_rating"), selected = "mpaa_rating"

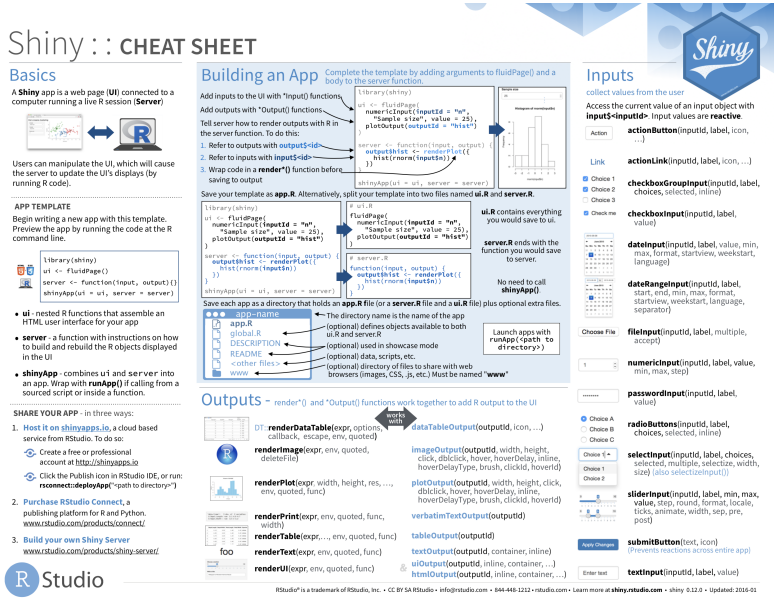
Use this variable in the aesthetics of the ggplot function as the color argument

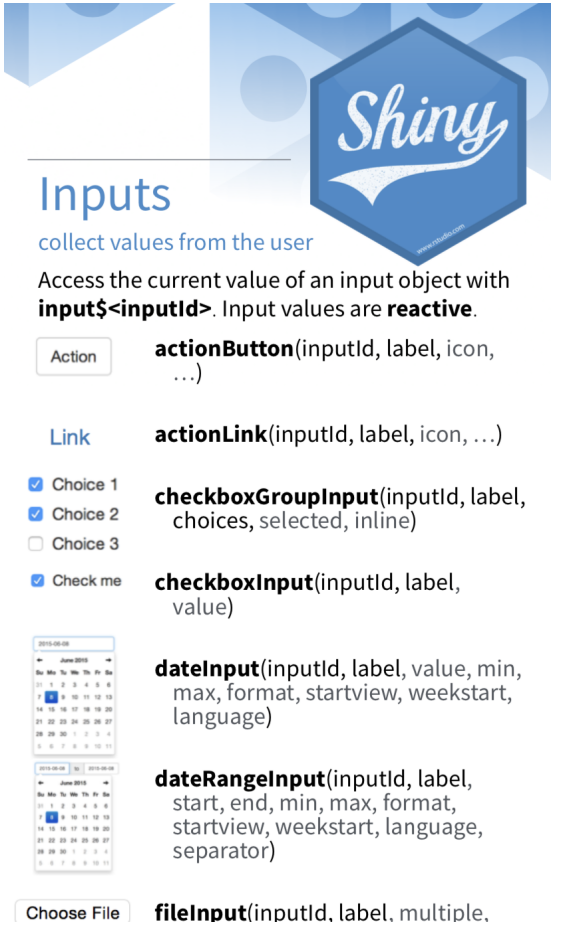
Run the app in the Viewer Pane

Your Turn 3 (solution: movies_02.R)

```
# in sidebarPanel()
selectInput(
  inputId = "z",
  label = "Color by:",
  choices = c("..."), # truncated
  selected = "mpaa_rating"
)
```

```
# in server <- function(input, output) {}
output$scatterplot <- renderPlot({
  ggplot(
    data = movies,
    aes_string(x = input$x, y = input$y, color = input$z)
  ) +
    geom_point()
})
```


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) {}
shinyApp(ui = ui, server = server)
ui - nested R functions that assemble an HTML user interface for your app
server - a function with instructions on how to build and rebuild the R objects displayed in the UI
shinyApp - combines ui and server into an app. Wrap with runApp() if calling from a sourced script or inside a function.
SHARE YOUR APP - in three ways:
1. Host it on shinyapps.io, a cloud based service from RStudio. To do so:
Create a free or professional account at http://shinyapps.io
Click the Publish icon in RStudio IDE, or run: rsconnect::deployApp("<path to directory>")
2. Purchase RStudio Connect, a publishing platform for R and Python. www.rstudio.com/products/connect/
3. Build your own Shiny Server www.rstudio.com/products/shiny-server/
RStudio
Building an App
Complete the template by adding arguments to fluidPage() and a body to the server function.
Add inputs to the UI with *Input() functions
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().
Save each app as a directory that holds an app.R file (or a server.R file and a ui.R file) plus optional extra files.
app-name
app.R
global.R
DESCRIPTION
README
<other files>
www
The directory name is the name of the app
(optional) defines objects available to both ui.R and server.R
(optional) used in showcase mode
(optional) data, scripts, etc.
(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "www"
Launch apps with runApp('<path to directory>')
Outputs - render*() and *Output() functions work together to add R output to the UI
works with
DT::renderDataTable(expr, options, callback, escape, env, quoted)
dataTableOutput(outputId, icon, ...)
renderImage(expr, env, quoted, deleteFile)
imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPlot(expr, width, height, res, ..., env, quoted, func)
plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)
renderPrint(expr, env, quoted, func, width)
verbatimTextOutput(outputId)
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, ...)
Inputs
collect values from the user
Access the current value of an input object with input$<inputId>. 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, language, separator)
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)
submitButton(text, icon) (Prevents reactions across entire app)
textInput(inputId, label, value)
RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at shiny.rstudio.com • shiny 0.12.0 • Updated: 2016-01


Inputs
collect values from the user
Access the current value of an input object with input$<inputId>. Input values are reactive.
Action
actionButton(inputId, label, icon, ...)
Link
actionLink(inputId, label, icon, ...)
Choice 1
Choice 2
Choice 3
checkboxGroupInput(inputId, label, choices, selected, inline)
Check me
checkboxInput(inputId, label, value)
dateInput(inputId, label, value, min, max, format, startview, weekstart, language)
dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)
Choose File
fileInput(inputId, label, multiple,

Your Turn 4

Add a slider input to control the alpha level of the scatterplot points. Don't forget to label it!

Set min **to 0 and** max **to 1. Choose a default for** value

Use the value from this input in the plot

Run the app

Your Turn 4 (solution: movies_03.R)

```
# in sidebarPanel()
sliderInput(
  inputId = "alpha",
  label = "Alpha:",
  min = 0,
  max = 1,
  value = 0.5
)
```

```
# in server <- function(input, output) {}
output$scatterplot <- renderPlot({
  ggplot(
    data = movies,
    aes_string(x = input$x, y = input$y, color = input$z)
  ) +
    geom_point(alpha = input$alpha)
})
```

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){}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines `ui` and `server` into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

SHARE YOUR APP - in three ways:

1. **Host it on shinyapps.io**, a cloud based service from RStudio. To do so:
   - Create a free or professional account at http://shinyapps.io
   - Click the Publish icon in RStudio IDE, or run: **rsconnect::deployApp("<path to directory>")**
2. **Purchase RStudio Connect**, a publishing platform for R and Python. www.rstudio.com/products/connect/
3. **Build your own Shiny Server** www.rstudio.com/products/shiny-server/

RStudio

Building an App

Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with *input() functions*

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

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

ui.R contains everything you would save to ui.

```
# server.R
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
```

server.R ends with the function you would save to server.

No need to call **shinyApp()**

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

```
app-name
  app.R
  global.R
  DESCRIPTION
  README
  <other files>
  www
```

- The directory name is the name of the app
- (optional) defines objects available to both ui.R and server.R
- (optional) used in showcase mode
- (optional) data, scripts, etc.
- (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp('<path to directory>')`

Outputs - render*() and *Output() functions work together to add R output to the UI

works with

DT::**renderDataTable**(expr, options, callback, escape, env, quoted) — **dataTableOutput**(outputId, icon, ...)

renderImage(expr, env, quoted, deleteFile) — **imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

renderPlot(expr, width, height, res, ..., env, quoted, func) — **plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

renderPrint(expr, env, quoted, func, width) — **verbatimTextOutput**(outputId)

renderTable(expr, ..., env, quoted, func) — **tableOutput**(outputId)

renderText(expr, env, quoted, func) — **textOutput**(outputId, container, inline)

renderUI(expr, env, quoted, func) & **htmlOutput**(outputId, inline, container, ...) — **uiOutput**(outputId, inline, container, ...)

Inputs

collect values from the user

Access the current value of an input object with **input$<inputId>**. 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, language, separator)

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)

submitButton(text, icon) (Prevents reactions across entire app)

textInput(inputId, label, value)

Outputs - render*() and *Output() functions work together to add

works with

DT::**renderDataTable**(expr, options, callback, escape, env, quoted) — **dataTableOutpu**

renderImage(expr, env, quoted, deleteFile) — **imageOutput**(ou click, dblclick, h hoverDelayTyp

renderPlot(expr, width, height, res, ..., env, quoted, func) — **plotOutput**(outp dblclick, hover, hoverDelayTyp

renderPrint(expr, env, quoted, func, width) — **verbatimTextOu**

renderTable(expr,..., env, quoted, func) — **tableOutput**(out

renderText(expr, env, quoted, func) — **textOutput**(outp

renderUI(expr, env, quoted, func) & **uiOutput**(output **htmlOutput**(outp

Your Turn 5

Add a new output in server **using** DT::renderDataTable()**. Inside of the render function, create a data table with** DT::datatable()

Set data = movies[, 1:7]**,** options = list(pageLength = 10)**, and** rownames = FALSE

Add the output to mainPanel() **in** ui **using** DT::dataTableOutput()

Run the app

Your Turn 5 (solution: movies_04.R)

```r
# in mainPanel()
DT::dataTableOutput(outputId = "moviestable")
```

```r
# in server <- function(input, output) {}
output$moviestable <- DT::renderDataTable({
  DT::datatable(
    data = movies[, 1:7],
    options = list(pageLength = 10),
    rownames = FALSE
  )
})
```

Your Turn 6

Add a title to your app with headerPanel()

Make the input choices nicer by making the vector named, e.g. choices = c("IMDB rating" = "imdb_rating", ...)

Clean up your axes titles with:

str_to_title() **to change to title case**

str_replace_all() **to replace _ with** " "

Your Turn 6 (solution: movies_05.R)

```
# in fluidPage()
headerPanel("Movie browser")
```

```
# in sidebarPanel()
selectInput(
  ...,
  choices = c(
    "IMDB rating" = "imdb_rating",
    "IMDB number of votes" = "imdb_num_votes",
    "Critics Score" = "critics_score",
    "Audience Score" = "audience_score",
    "Runtime" = "runtime"
  )
)
```

Your Turn 6 (solution: movies_05.R)

```
# in server <- function(input, output) {}
output$scatterplot <- renderPlot({
  ggplot(
    data = movies,
    aes_string(x = input$x, y = input$y, color = input$z)) +
    geom_point(alpha = input$alpha) +
    labs(
      x = str_to_title(str_replace_all(input$x, "_", " ")),
      y = str_to_title(str_replace_all(input$y, "_", " ")),
      color = str_to_title(str_replace_all(input$z, "_", " "))
    )
})
```

Directory Structure

```
|--name_of_app
    |-- app.R
```

```
|--name_of_app
    |-- ui.R
    |-- server.R
    |-- global.R
    |-- www
        |-- image.png
```

Sharing your app

Sharing your app

shinyapps.io

Sharing your app

shinyapps.io

Shiny Server

Sharing your app

shinyapps.io

Shiny Server

RStudio Connect or Shiny Server Pro

Your Turn 7

Create folder called movies_app

Move any of the (working) app files into this folder, along with movies.Rdata

Go to http://shinyapps.io. Sign up for an account. Follow the instructions to deply your app.

Resources

Shiny Website: A collection of articles on Shiny

Mastering Shiny: A Work-in-progress book from Hadley Wickham