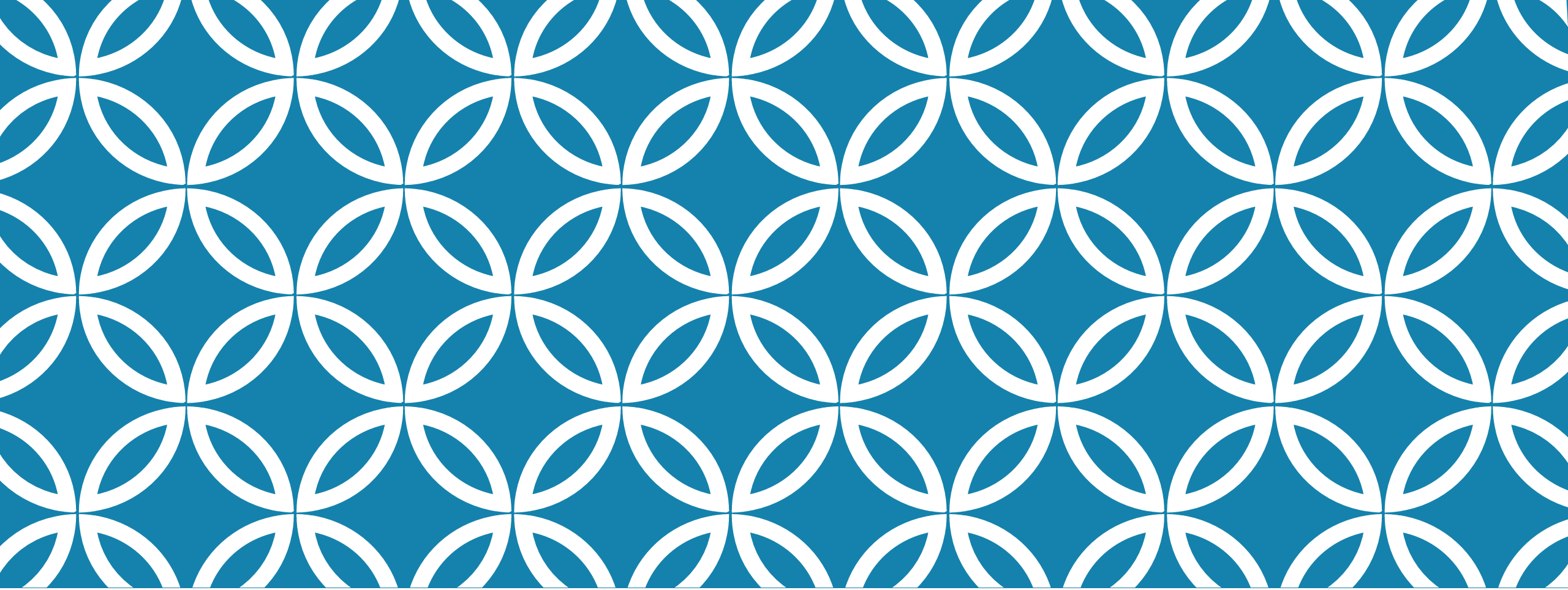


# INTERACTIVE WEB APPS WITH SHINY

Katarzyna M Tyc, PhD

*--- Guest Lecturer ---*
Virginia Commonwealth University
09/25/2023

# REFERENCES

Material covered in these slides is based on the following resources:

ShinyApp gallery: https://shiny.rstudio.com/gallery/

Official Shiny tutorials: https://shiny.rstudio.com/tutorial/written-tutorial/lesson1/

Materials from previous years: https://bios524-r-2021.netlify.app/class/10-class/

Other resources: https://debruine.github.io/shinyintro/index.html

Example source: https://www.r-bloggers.com/2019/12/r-shiny-for-beginners-annotated-starter-code/

# PREREQUISITES

Basic familiarity with R and RStudio:

- Data import
- Data processing
- Data visualization
- If/else statements
- …

```
library(ggplot2)

pets <- read.csv("pets.csv")

dv <- sample(c("score", "weight"), 1)

if (dv == "score") {
  g <- ggplot(pets, aes(pet, score, fill = country))
} else if (dv == "weight") {
  g <- ggplot(pets, aes(pet, weight, fill = country))
}

g + geom_violin(alpha = 0.5)
```

Source: https://debruine.github.io/shinyintro/index.html

```
install.packages("shiny")
```

# LECTURE OUTLINE

Setting up a folder to *host* your Shiny app

Source code structure of a Shiny app

Adding content into User Interface (UI)

Introducing interactive elements to control the app (a.k.a. widgets)

Connecting widgets to reactive output

# EXAMPLE SHINYAPPS

runExample("01_hello", display.mode = "normal") # *a histogram*

runExample("02_text" ) # *tables and data frames*

runExample("03_reactivity") # *a reactive expression*

runExample("04_mpg") # *global variables*

runExample("05_sliders") # *slider bars*

runExample("06_tabsets") # *tabbed panels*

runExample("07_widgets") # *help text and submit buttons*

runExample("08_html") # *Shiny app built from HTML*

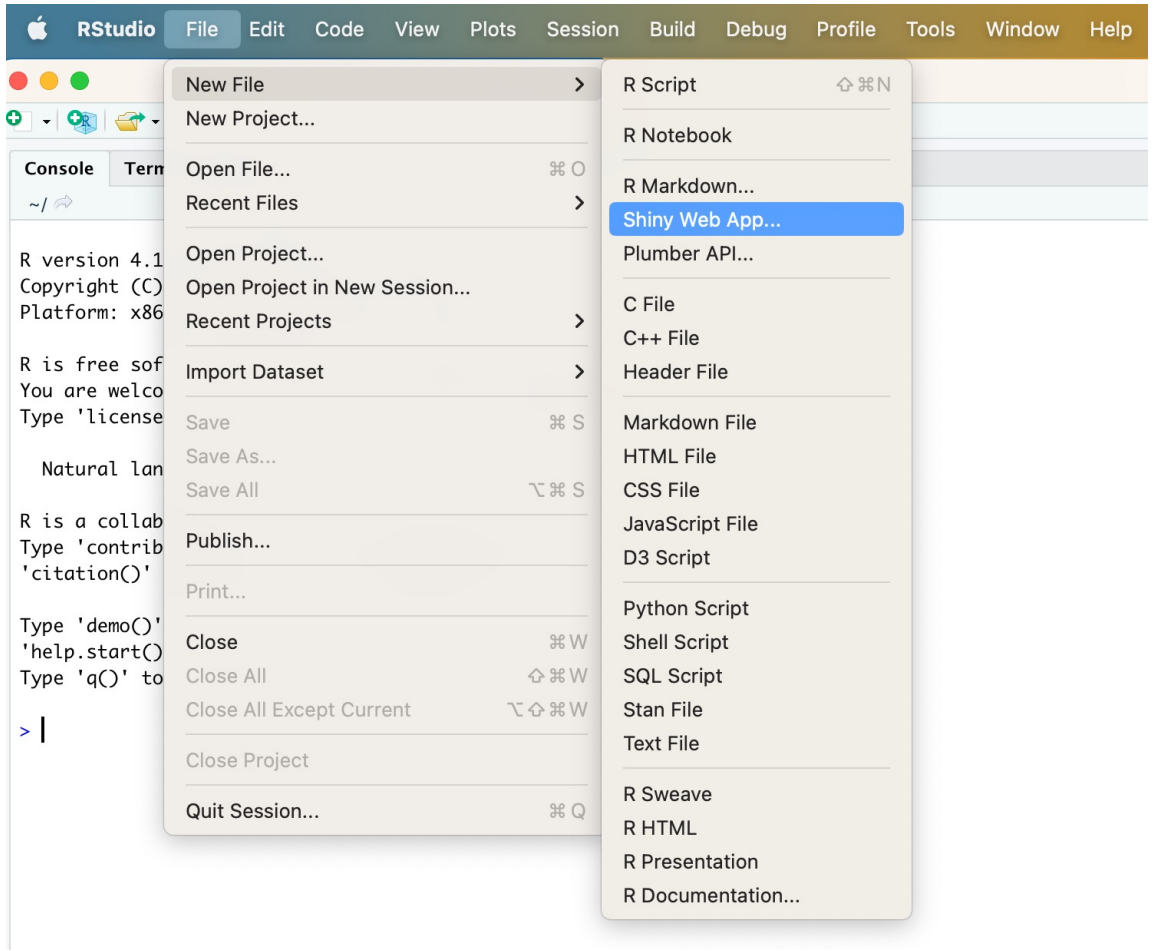
runExample("09_upload") # *file upload wizard*

runExample("10_download") # *file download wizard*

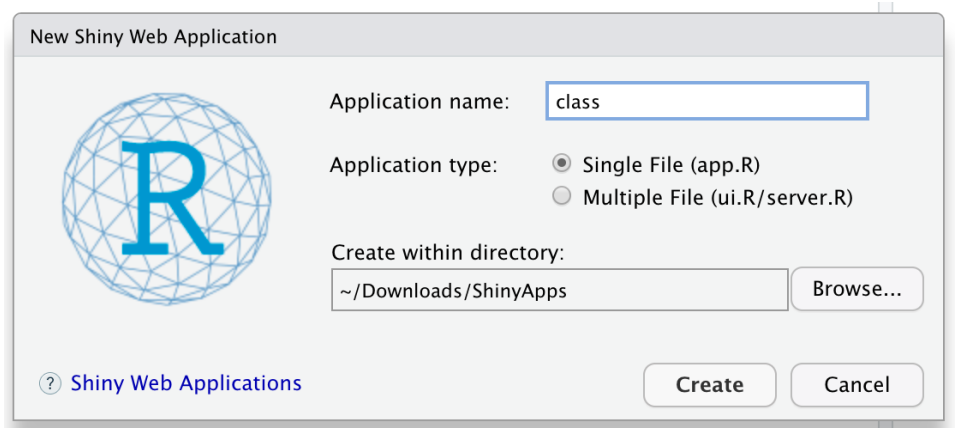
runExample("11_timer") # *an automated timer*

# YOUR FIRST DEMO

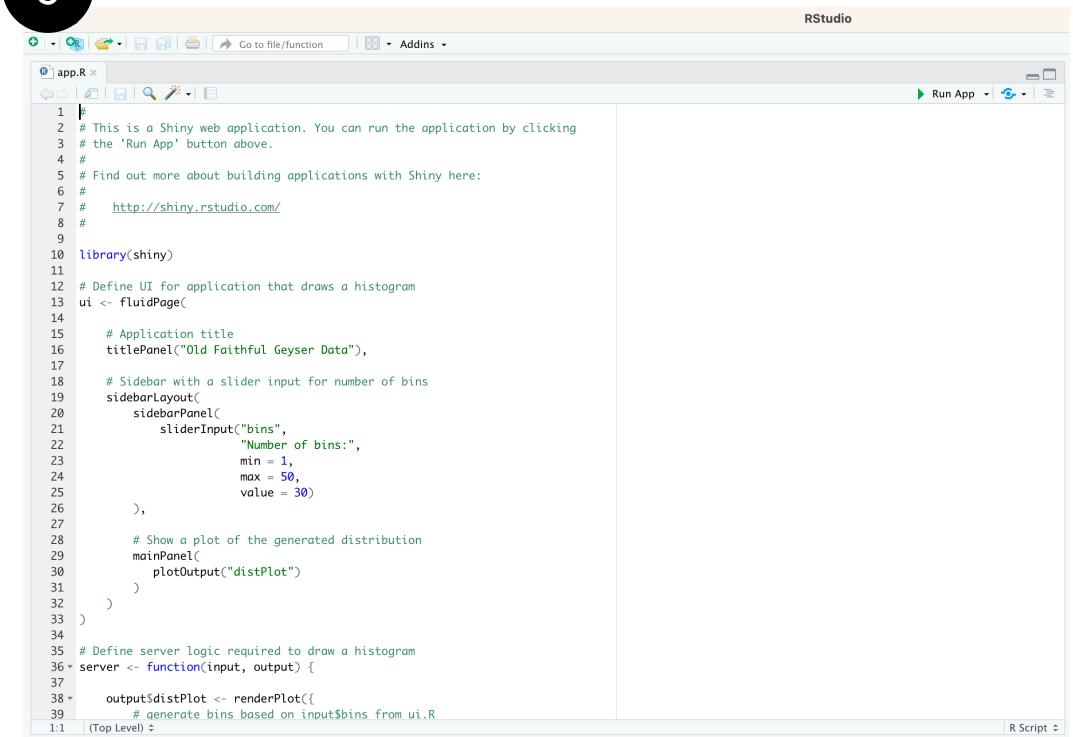
1




2




3




Refer to https://debruine.github.io/shinyintro/index.html for practicing!

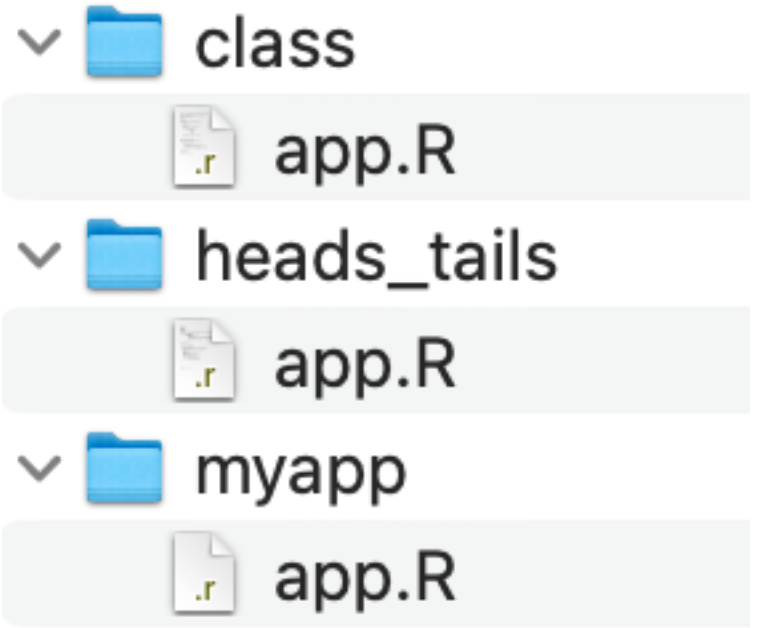
# DIRECTORY OF A SHINY APP

Shiny apps are contained in a single script called **app.R**

Once you save it in a directory, you can run the app by running **runApp()**

Example:

```
library(shiny)

runApp("~/Downloads/ShinyApps/class/")
runApp("~/Downloads/ShinyApps/heads_tails/")
runApp("~/Downloads/ShinyApps/myapp/")

display.mode = "showcase" # allows to see the app code
```

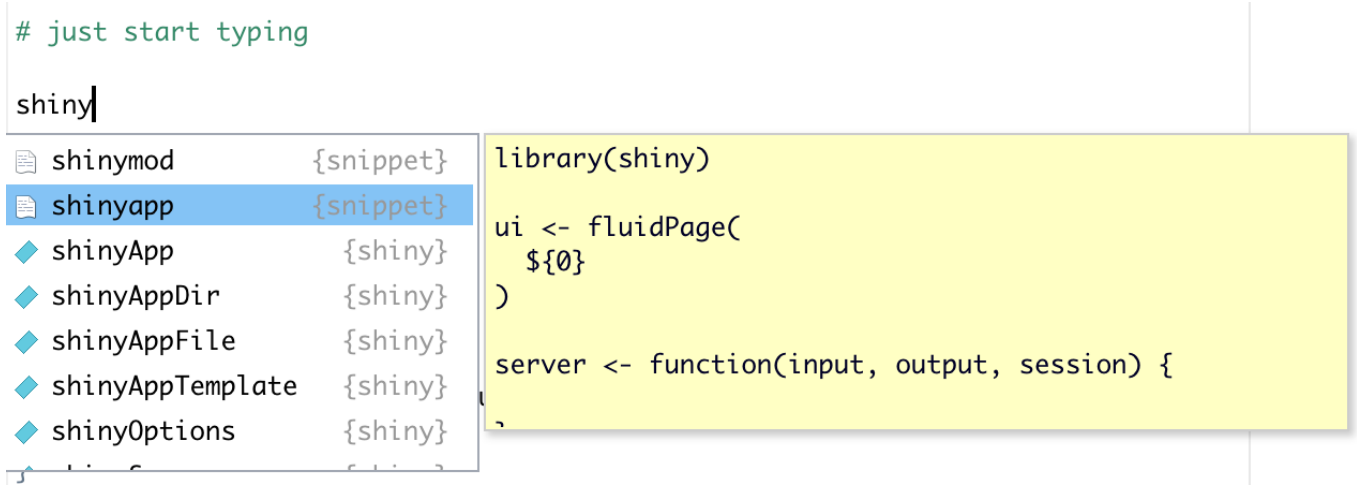


# STRUCTURE OF A SHINY APP

```
library(shiny)

# See above for the definitions of ui and server
ui <- ...

server <- ...

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

Example.

shinyapp {snippet}



# CREATE YOUR SHINY APP

```
library(shiny)

#  Define UI ----
ui <- fluidPage( )

# Define server logic ----
server <- function(input, output) { }

# Run the app ----
shinyApp(ui = ui, server = server)
```

# ADD SOME LAYOUT

```
 ui <- fluidPage(
titlePanel("title panel"),
sidebarLayout(
        sidebarPanel("sidebar panel"),
        mainPanel("main panel") )
)
```

# ADD SOME MORE LAYOUT

```
ui <- fluidPage(
titlePanel("My Shiny App"),
sidebarLayout(
        sidebarPanel(),
        mainPanel(
                h1("First level title", align = "center"),
                h2("Second level title"),
                h3("Third level title"),
                h4("Fourth level title"),
                h5("Fifth level title"),
                h6("Sixth level title") ) ) )
```

# TEXT FORMATTING

```
ui <- fluidPage(
titlePanel("My Shiny App"),
sidebarLayout( sidebarPanel(),
        mainPanel(
        p("p creates a paragraph of text."),
        strong("strong() makes bold text."),
        em("em() creates italicized (i.e, emphasized) text."),
        br(),
        code("code displays your text similar to computer code"),
        div("div creates segments of text with a similar style. This division of text is all blue
because I passed the argument 'style = color:blue' to div", style = "color:blue")  ) ) )
```

# CONTROL WIDGETS

Buttons

Checkbox

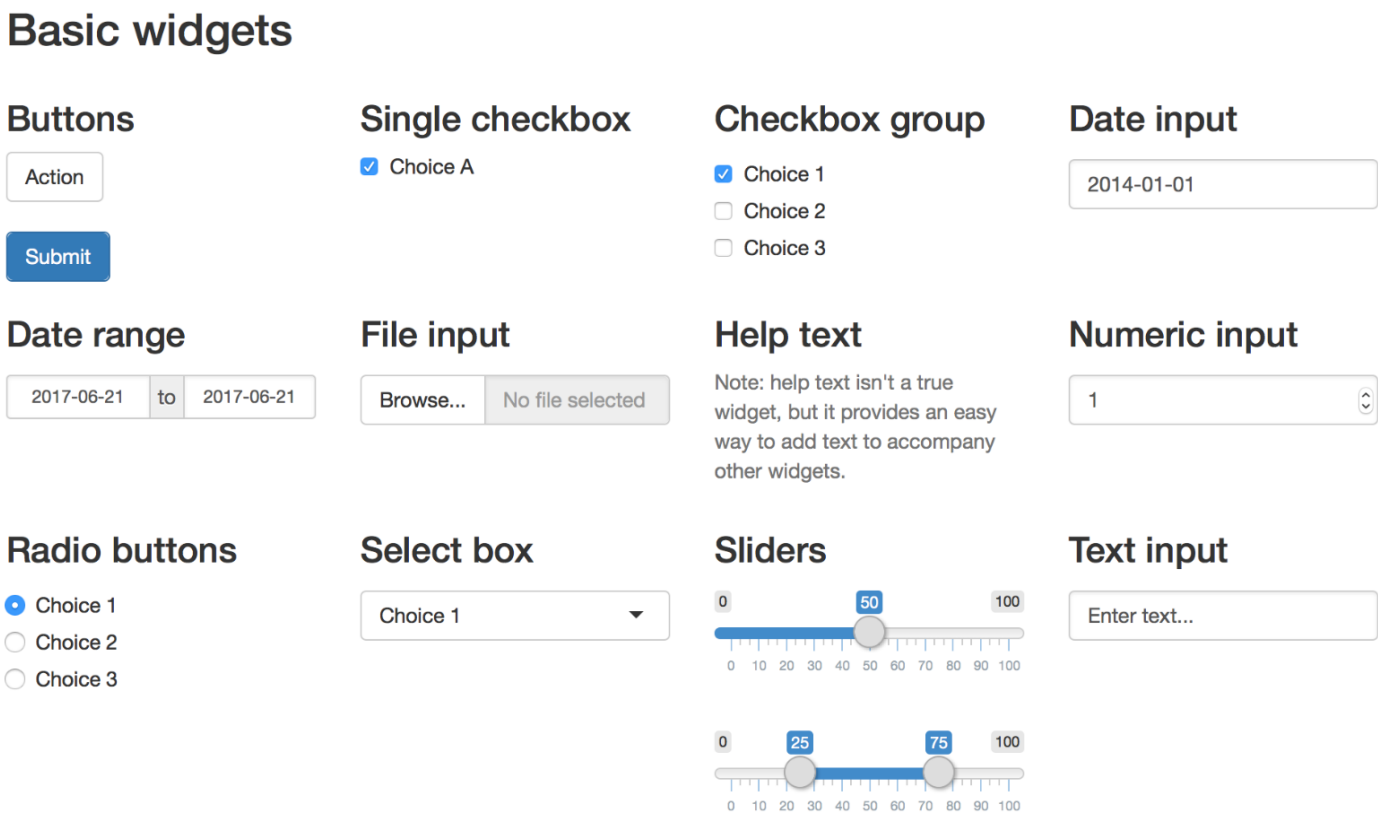
File input

Select box

Sliders

Text or numeric input

…




https://shiny.rstudio.com/tutorial/written-tutorial/lesson3/

# ADD SOME WIDGETS

```
sidebarPanel(
        helpText("some help text"),
        selectInput(inputId = "sample",
                label = "Select sample:", choices = c("a","b","c")),
        sliderInput(inputId = "cutoff",
                label = "Select a threshold:", min = 0, max = 255, value = 10)
    )
```

# DISPLAY REACTIVE OUTPUT

| Output function | Creates |
|---|---|
| `dataTableOutput` | DataTable |
| `htmlOutput` | raw HTML |
| `imageOutput` | image |
| `plotOutput` | plot |
| `tableOutput` | table |
| `textOutput` | text |
| `uiOutput` | raw HTML |
| `verbatimTextOutput` | text |

Step 1: Add an R object to the UI

```
mainPanel( textOutput("selected_cutoff") )
```

Step 2: Provide R code to build the object (happens inside `server`).

```
server <- function(input, output) {
        output$selected_cutoff <- renderText({ paste("You have selected", input$cutoff) })
        }
```

# HEADS AND TAILS

```
#  Define UI ----
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(sliderInput(inputId = "n", label = "No of flips:", min = 10, max = 1000, value = 10),
                 sliderInput(inputId = "prob", label = "Success rate:", min = 0, max = 1, value = 0.5)
    ),
    mainPanel( plotOutput(outputId = "bars") )
  ))
```

# HEADS AND TAILS: OUTPUT

**Testing first:**

```
rbinom(n=25, size = 1, prob =0.5)
barplot(table(rbinom(n=25, size = 1, prob =0.5)))
```

**Wrap it in the output:**

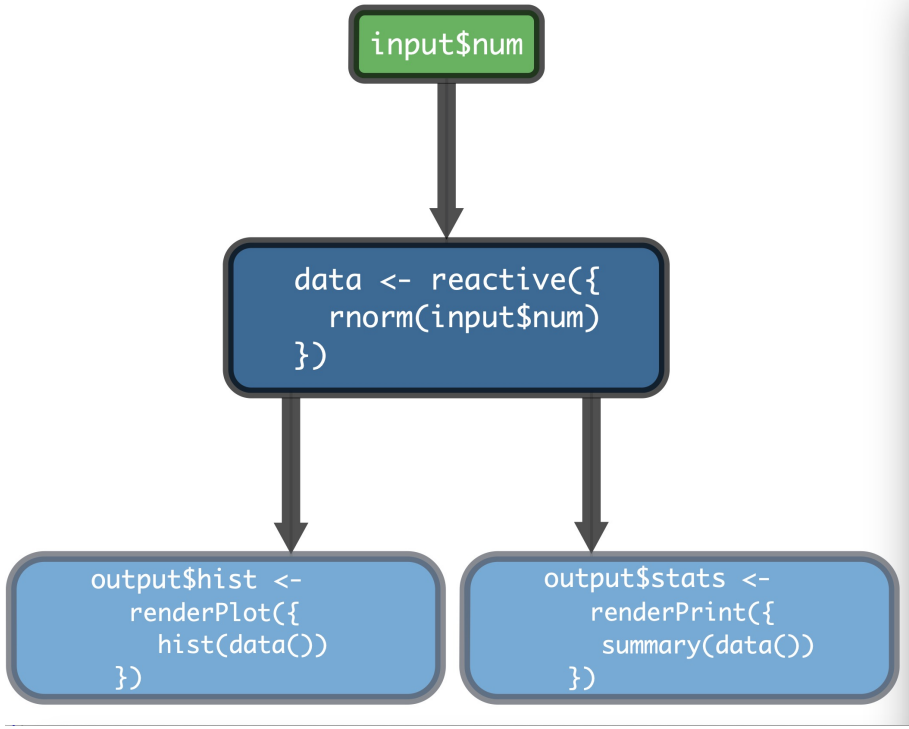
```
output$bars <- renderPlot({   barplot(table(rbinom(n=25, size = 1, prob =0.5)))    })
```

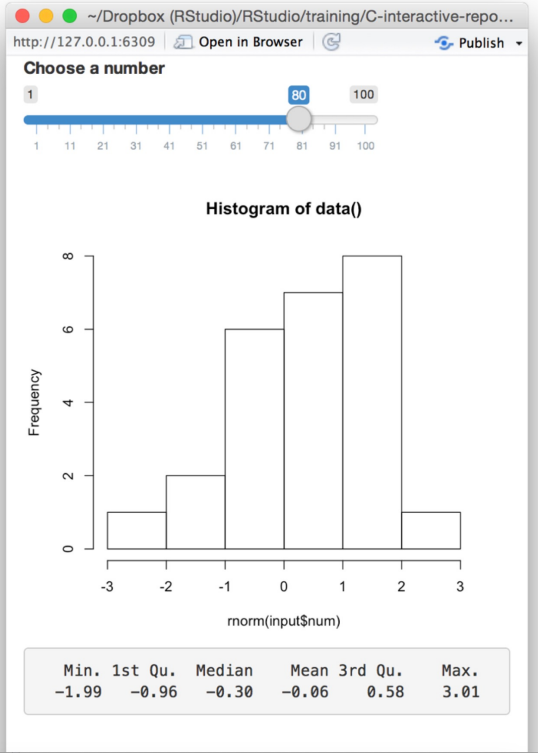
**Connect to dynamic input:**

```
output$bars <- renderPlot({  barplot(table(rbinom(n = input$n,size = 1,prob = input$prob)))
})
```

# REACTIVE EXPRESSION

```
data <- reactive(table(rbinom(input$n, 1, input$prob)))


output$bars <- renderPlot({
    barplot(data())
})
```



# ADD REACTIVE EXPRESSION

```
# modify mainPanel in the ui:
mainPanel(plotOutput(outputId = "bars"),
          plotOutput(outputId = "hist"))

# and update the server:
server <- function(input, output) {
   data <- reactive(table(rbinom(input$n, input$size, input$prob)))
   output$bars <- renderPlot({  barplot(data()) })
   output$hist <- renderPlot({ hist(data())  })
 }
```

# EXAMPLE: MORE TESTING OF REACTIVE EXPRESSION

```
library(shiny)

ui <- fluidPage(

  sidebarLayout(

    sidebarPanel(sliderInput(inputId = "n", label = "No. of coin flips", min = 10, max = 1000, value = 10),

      sliderInput(inputId = "prob", label = "Success of rate", min = 0, max = 1, value = 0.5) ),

   mainPanel( plotOutput(outputId = "xxxxx"),

              plotOutput(outputId = "aaa"),

              plotOutput(outputId = "ccccc") ) )

)

server <- function(input, output, session) {

  call_it_something <- reactive(table(rbinom(n=input$n, size = 1, prob = input$prob))) # values are drawn just once and saved in this reactive expression

  output$xxxxx <- renderPlot({   barplot(call_it_something() )  })

   output$aaa <- renderPlot({    barplot(call_it_something() )   })

   output$ccccc <- renderPlot({barplot(table(rbinom(n=input$n, size = 1, prob = input$prob)))   }) # since we do not use reactive expression, values will be drawn on the fly and the result will
be different from the two above

}

shinyApp(ui, server)
```

# Example from: https://www.r-bloggers.com/2019/12/r-shiny-for-beginners-annotated-starter-code/

# HEADS AND TAILS: ESTHETICS

```
output$bars <- renderPlot({
      flips <- tibble(flips = rbinom(input$n, 1, input$prob)) %>%
      mutate(flips = if_else(flips == 1, "Heads", "Tails"))
    flips %>%
      count(flips) %>%
      ggplot(aes(flips, n, fill = flips)) +
      geom_col() +
      geom_label(aes(flips, n, label = n), size = 5) +
      theme(legend.position = "none",
            axis.text = element_text(size = 15)) +
      labs(x = "", y = "") +
      ggtitle(str_c("Results of ", input$n,
                    " flips with Heads probability ",
                    sprintf("%.2f", input$prob)))
    })
```

```
library(dplyr)
library(ggplot2)
library(stringr)
library(tibble)
```

# (1) SHARE YOUR APP VIA GITHUB

Host your code on GitHub: https://github.com/rstudio/shiny_example/ (repository must be public) and run your app from within R using runGitHub() or runUrl().

**Example:**

shiny::runGitHub("shiny_example", "rstudio")

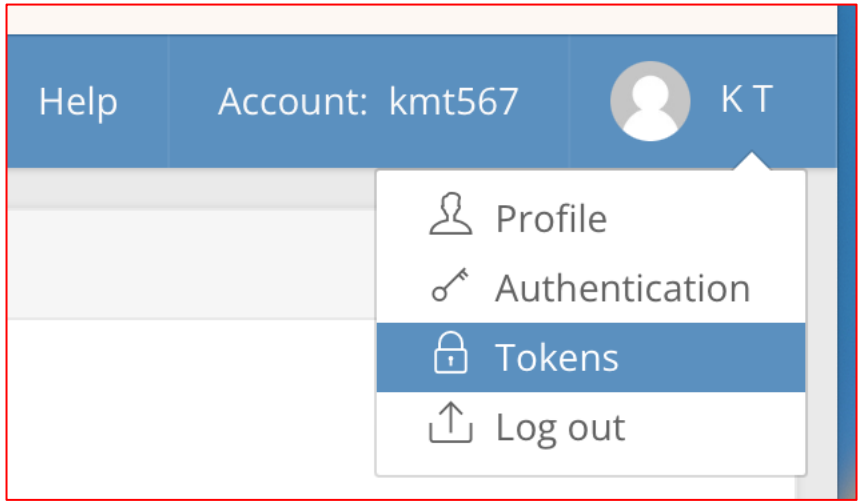
shiny::runUrl('https://github.com/rstudio/shiny_example/archive/main.tar.gz')

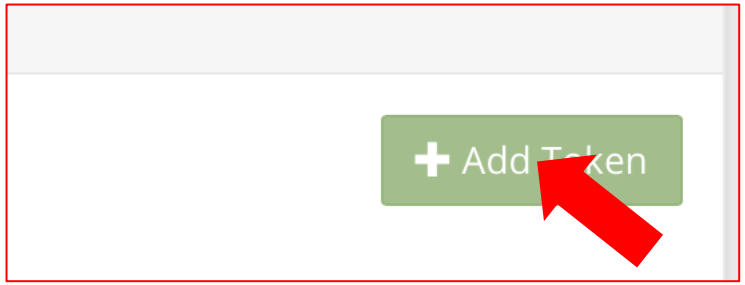
# Hosted on my GitHub account: https://github.com/kmt555/MyShinyApps

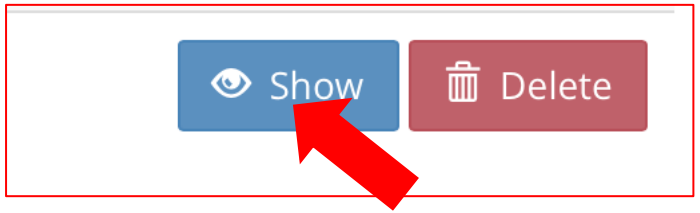
shiny::runGitHub(repo = "MyShinyApps", username = "kmt555", ref ="main")

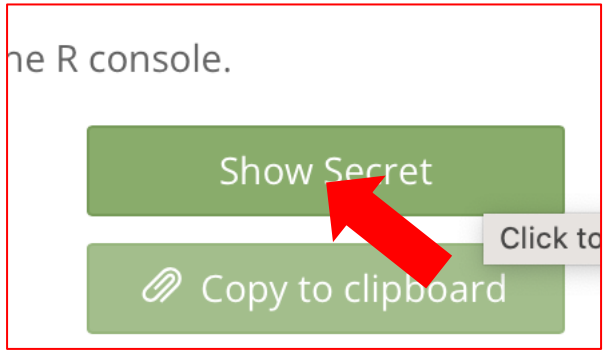
# (2) SHARE YOUR APP WITH SHINYAPPS.IO

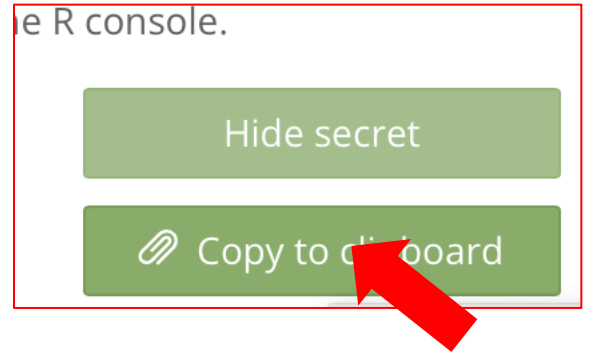
1. Create an account on shinyapps.io and collect a “token”

2. 


3. 

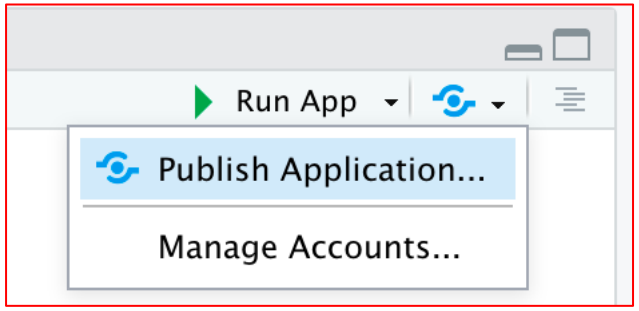

4. 


5. 

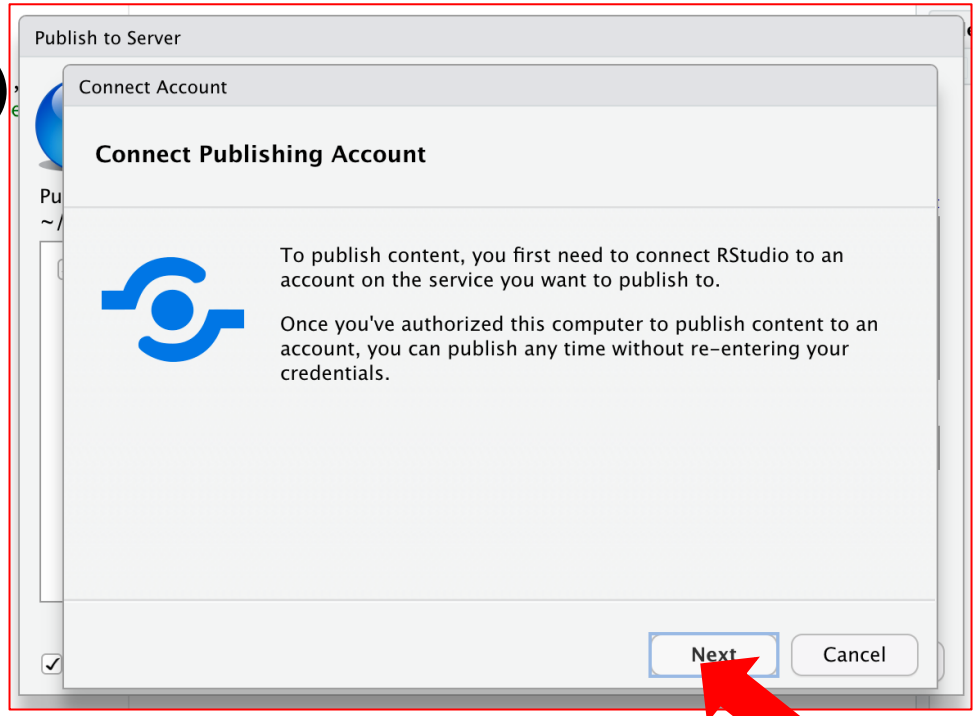

6. 

# CONNECT YOUR RSTUDIO WITH SHINYAPPS.IO

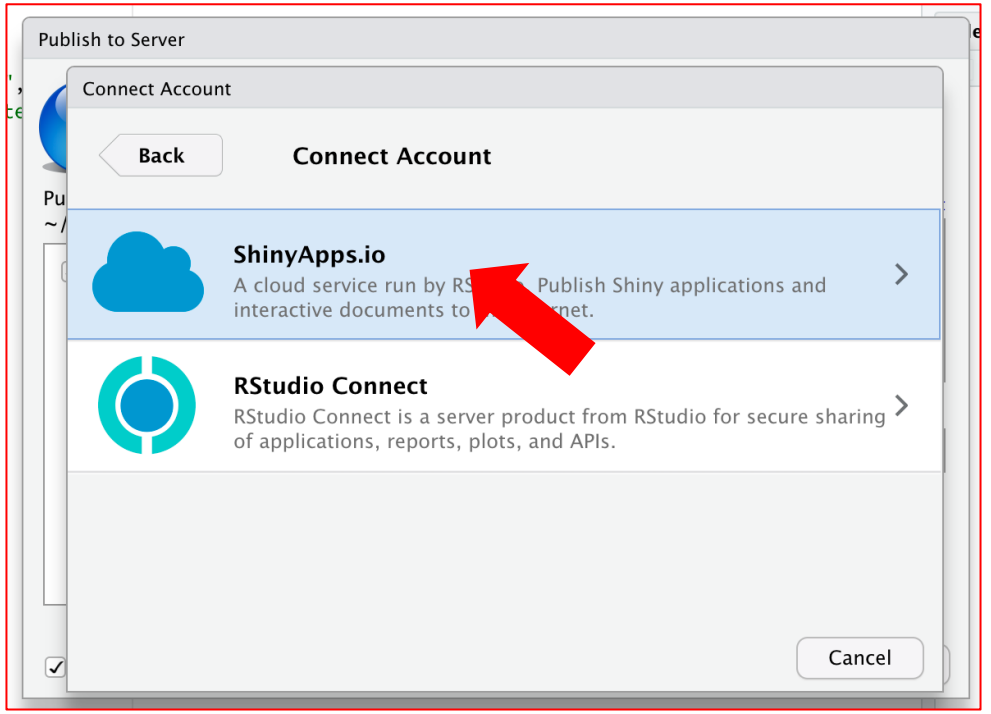
7




8

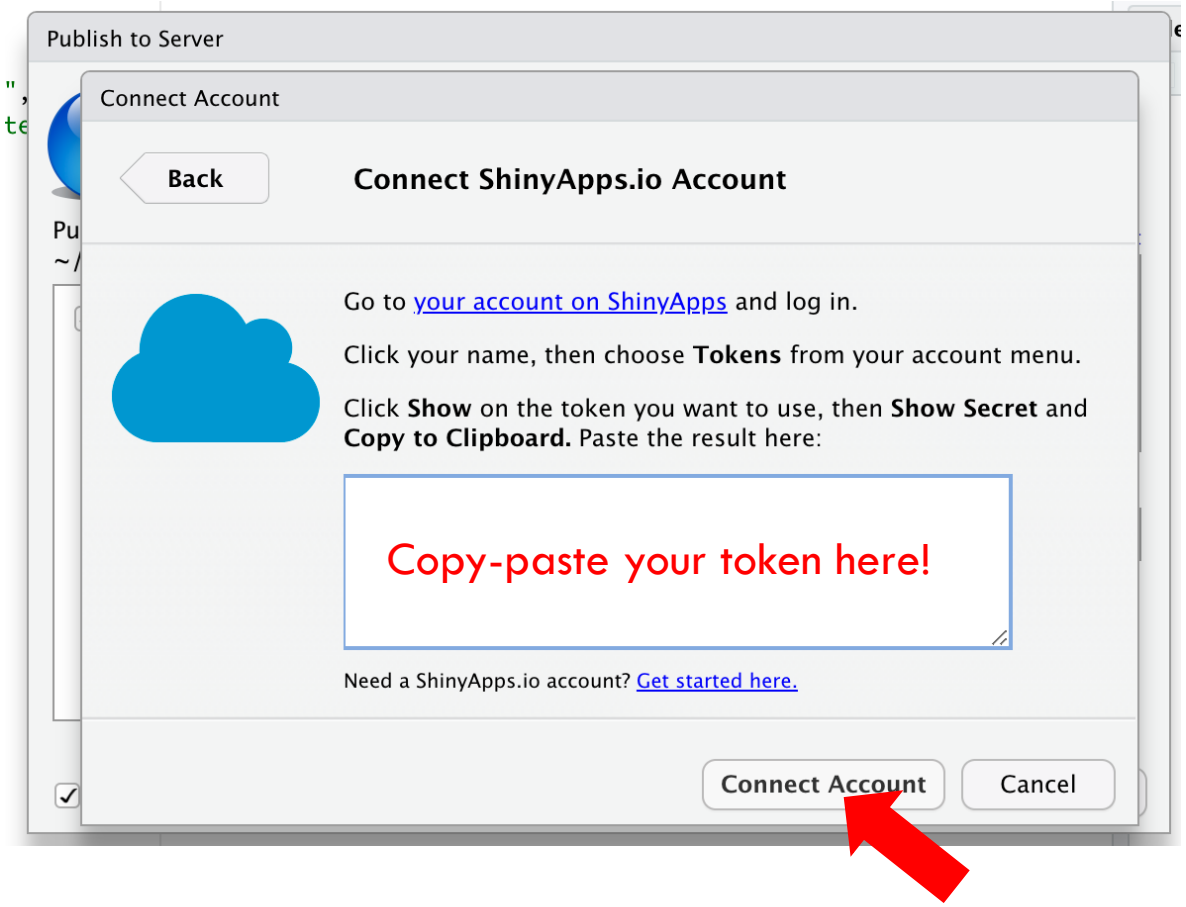



9

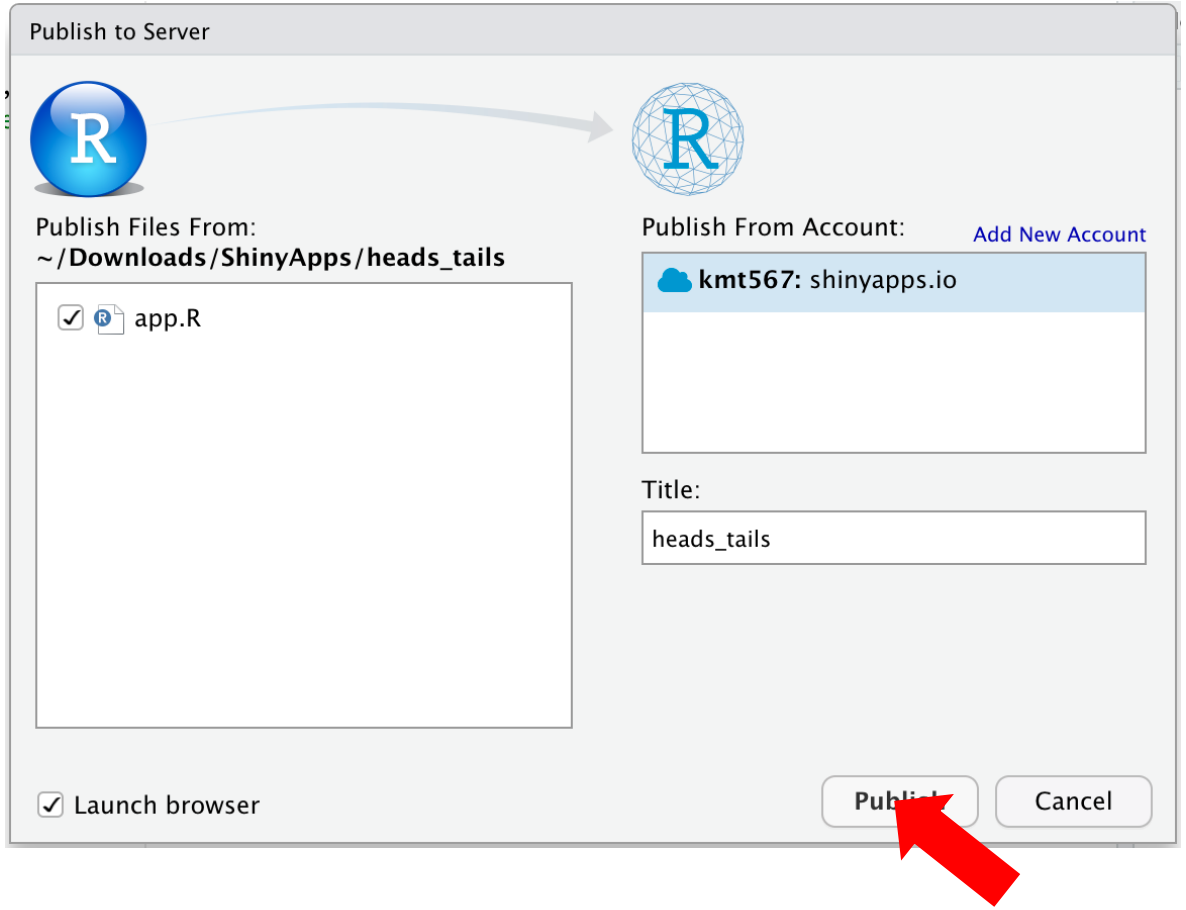


# SHINYAPPS.IO . . .

10




11



https://shiny.rstudio.com/articles/action-buttons.html

# ACTION BUTTONS

```
library(shiny)

ui <- fluidPage(
actionButton(inputId = "clicks", label = "Click me") )

server <- function(input, output) {
observeEvent(input$clicks, { print(as.numeric(input$clicks)) })
}

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

# USE ACTION BUTTONS TO DELAY REACTIONS

```
library(shiny)
ui <- fluidPage(
sliderInput(inputId = "num", label = "Choose a number", value = 25, min = 1, max =
100),
plotOutput("hist") )

server <- function(input, output) {

output$hist <- renderPlot({ hist(rnorm(input$num))
}) }

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

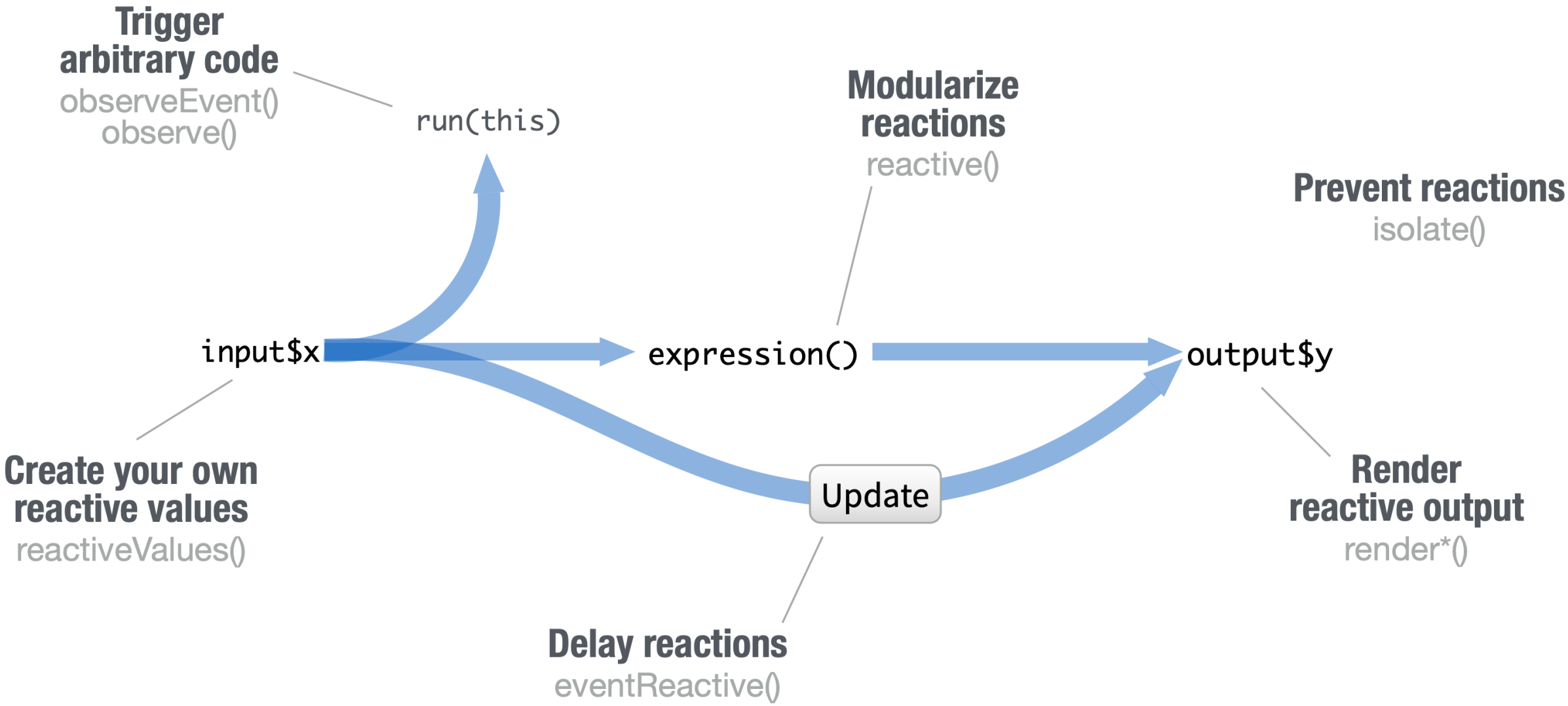
# ADD BUTTON

```
library(shiny)
ui <- fluidPage(
sliderInput(inputId = "num", label = "Choose a number", value = 25, min = 1, max =
100),
actionButton(inputId = "go", label = "Update"), plotOutput("hist") )

server <- function(input, output) {
data <- eventReactive(input$go, { })
output$hist <- renderPlot({ hist(rnorm(input$num))
}) }

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

```
server <- function(input, output) {
data <- eventReactive(input$go, { rnorm(input$num) })
output$hist <- renderPlot({ hist(data())
})
}
```

# REACTIVEVALUES()

```
library(shiny)

ui <- fluidPage(

       actionButton(inputId = "norm", label = "Normal"),
       actionButton(inputId = "unif", label = "Uniform"),
       plotOutput("hist") )

server <- function(input, output) {

       rv <- reactiveValues(data = rnorm(100))
       observeEvent(input$norm, { rv$data <- rnorm(100) })
       observeEvent(input$unif, { rv$data <- runif(100) })

       output$hist <- renderPlot({ hist(rv$data) }) }

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


Trigger
arbitrary code
observeEvent()
observe()
run(this)
Modularize
reactions
reactive()
Prevent reactions
isolate()
input$x
expression()
output$y
Update
Create your own
reactive values
reactiveValues()
Render
reactive output
render*()
Delay reactions
eventReactive()