NYC DATA SCIENCE ACADEMY

# Introduction to Shiny 1

## Outline of today's class

- Shiny introduction
- Design the User-interface
- Control Widgets
- Build reactive output
- Use datatable in Shiny Apps

## Outline of today's class

- Shiny introduction
  - Design the User-interface
  - Control Widgets
  - Build reactive output
  - Use datatable in Shiny Apps

## What is Shiny

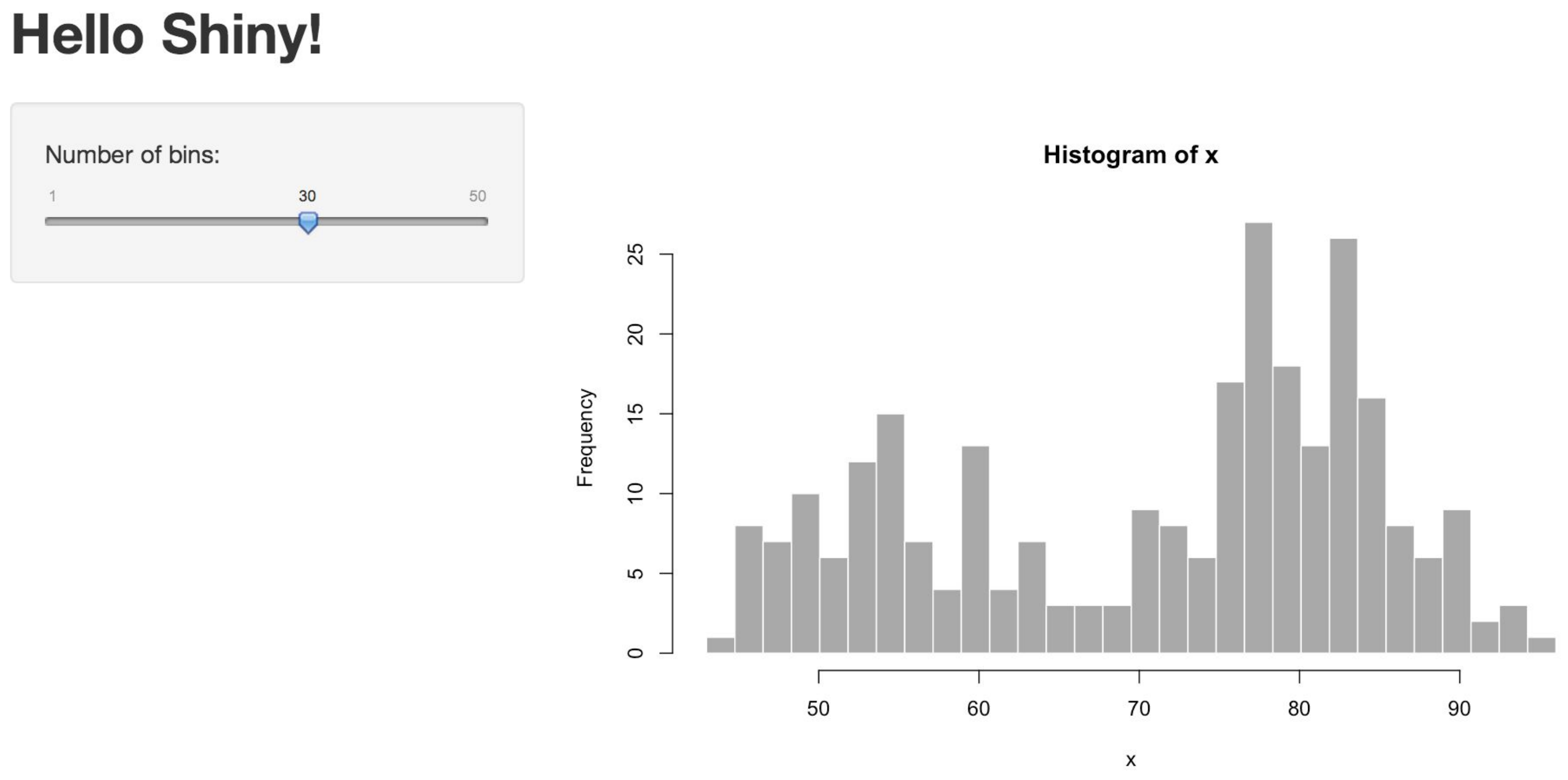
- A web application framework for R.
- Turn your analyses into interactive web applications.
- NO HTML, CSS, or JavaScript knowledge required.

## Shiny Features

- Shiny makes it super simple for R users to turn analyses into interactive web applications that anyone can use.
- Let your users choose input parameters using friendly controls like sliders, dropdowns, and text fields.
- Easily incorporate any number of outputs like plots, tables, and summaries.
- If you have some experience with R, you're just minutes away from combining the statistical power of R with the simplicity of a web page.

## Example: Hello Shiny!

```r
library(shiny)
runExample("01_hello")
```



## Structure of a Shiny App

- Shiny apps have two components:
  - A user-interface script
    - Store in ui.R script
    - Control the layout and appearance
  - A server script
    - Store in server.R script
    - Instructions computer needs to build app

## Structure of a Shiny App

- User-interface script
  - Defined in a source script named ui.R
  - Controls the layout and appearance of your app

## ui.R Script for Hello Shiny Example

```r
shinyUI(fluidPage(
  # Application title
  titlePanel("Hello Shiny!"),
  # Sidebar with a slider input for the number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
                  value = 30)
    ),
    # Show a plot of the generated distribution
    mainPanel(
      plotOutput("distPlot")
    )
  )
))
```

## Structure of a Shiny App

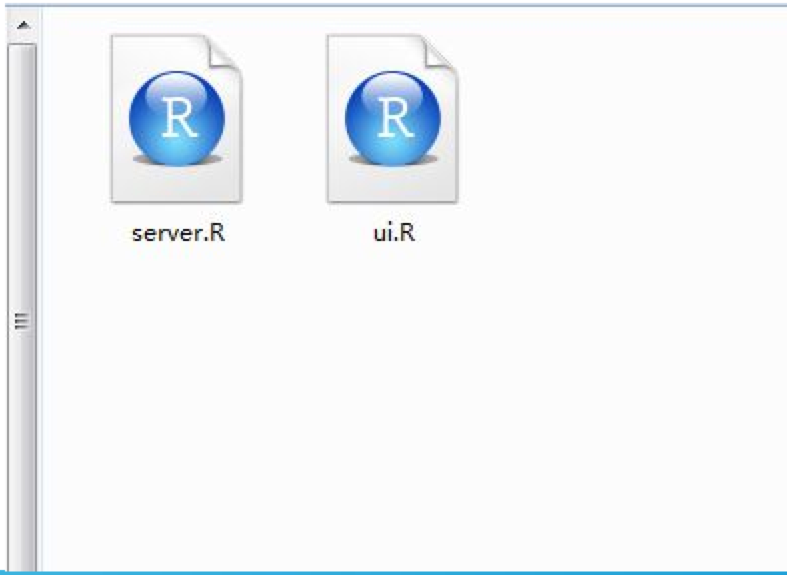
- Server script
  - Defined in a source script named server.R.
  - Contains instructions that your computer needs to build your app.

## server.R Script for Hello Shiny Example

```
shinyServer(function(input, output) {
  # Expression that generates a histogram. The expression is
  # wrapped in a call to renderPlot to indicate that:
  #
  #  1) It is "reactive" and therefore should re-execute
automatically
  #     when inputs change
  #  2) Its output type is a plot
  output$distPlot <- renderPlot({
    x    <- faithful[, 2]  # Old Faithful Geyser data
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })
})
```

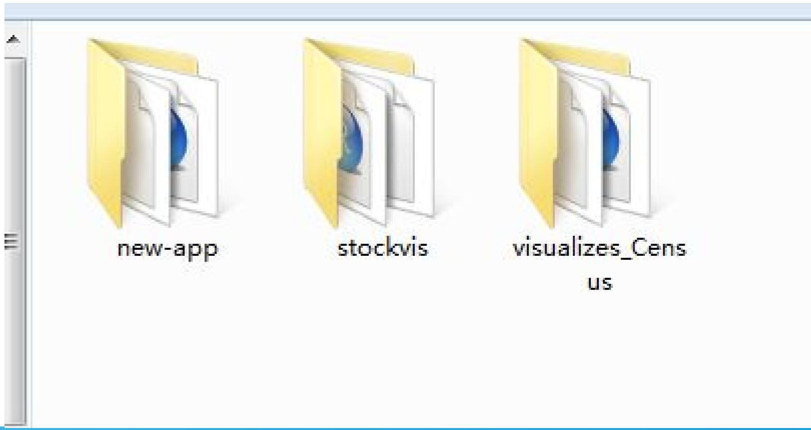
## Running an App

- Every Shiny app has the same structure: two R scripts saved together in a directory.
- At a minimum, a Shiny app has ui.R and server.R files.



## Running an App

- Create a Shiny app by making a new directory and saving a ui.R and server.R file inside it.
- Each app will need its own unique directory.



## Running an App

- Run a Shiny app by giving the name of its directory to the function runApp.
- For example :
  - ➢ If your Shiny app is in a directory called my_app, run it with the following code:

```
library(shiny)
runApp("~/my_app")
```

  - ➢ The first argument of runApp is the app's directory.

## Running an App

❖ More examples without coding

```
library(shiny)
runExample("01_hello") # 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
...
```

## Outline of today's class

- Shiny introduction
- **Design the User-interface**
- Control Widgets
- Build reactive output
- Use datatable in Shiny Apps

## Create an Empty App

- Edit the scripts to match the ones below:
  - ui.R

```
shinyUI(fluidPage(
))
```

  - server.R

```
shinyServer(function(input, output) {
})
```

- This code is the bare minimum needed to create a Shiny app.
- The result is an empty app with a blank user-interface, an appropriate starting point for this lesson.

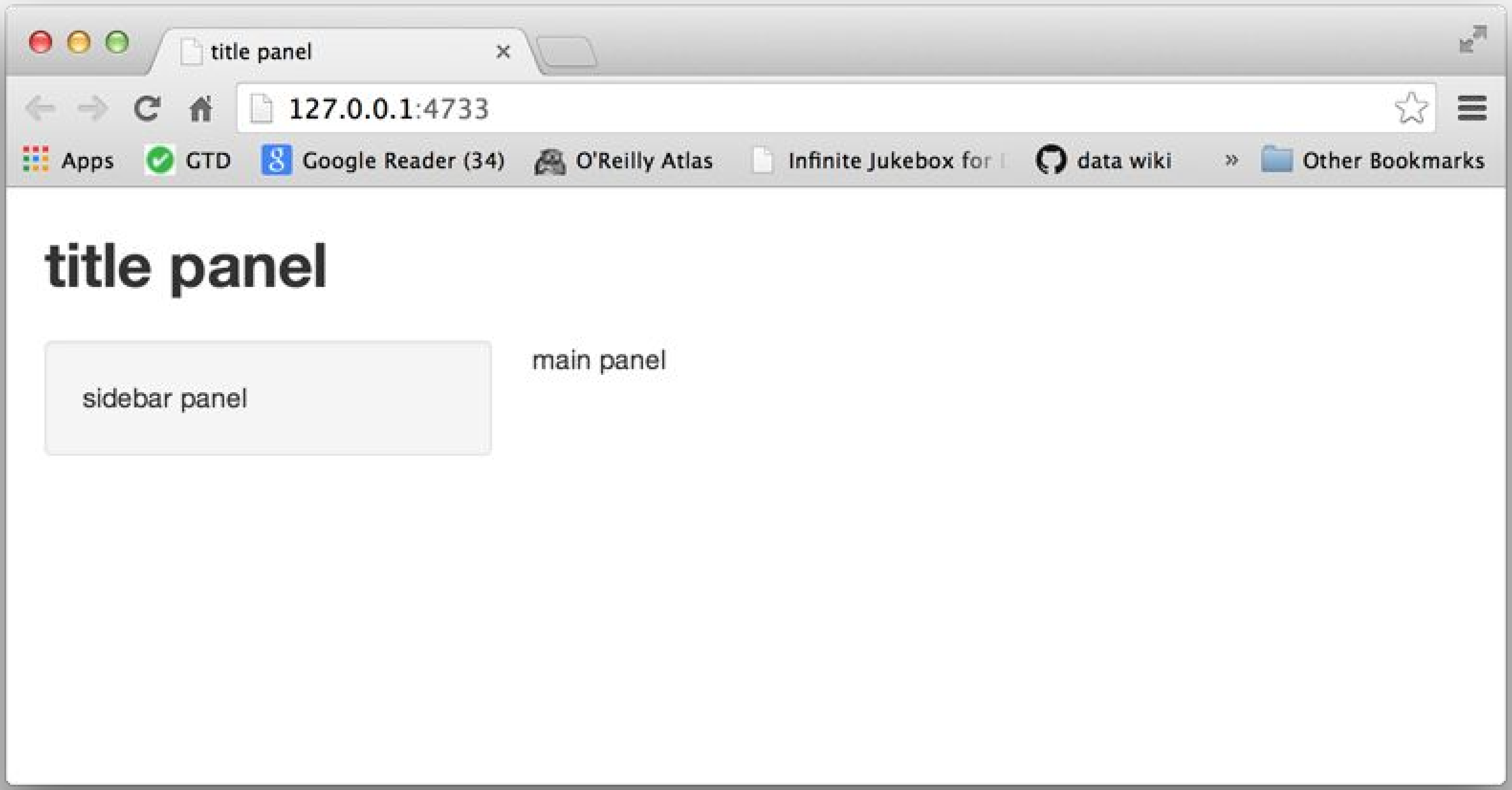
## Layout

- ui.R scripts use the function fluidPage to create a display that automatically adjusts to the dimensions of your user's browser window.
- Lay out your app by placing elements in the function fluidPage .

```
# ui.R
shinyUI(fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel( "sidebar panel"),
    mainPanel("main panel")
  )
))
```

## Layout

- It creates a UI with 3 titles:



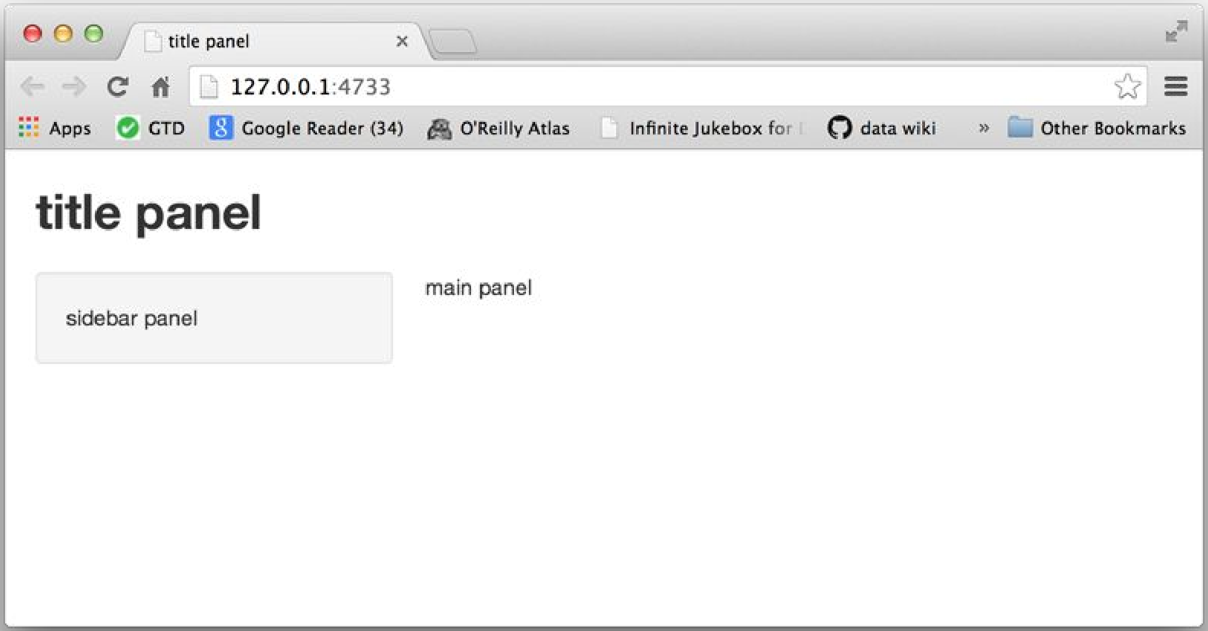
title panel

sidebar panel

main panel

## Layout

- Add elements to fluidPage
  - titlePanel and sidebarLayout
    - The two most popular elements to add to fluidPage.
    - Create a basic Shiny app with a sidebar.



title panel

sidebar panel

main panel

## Layout

- titlePanel show the head title.
- sidebarLayout always takes two arguments:
  - sidebarPanel function output
    - default:left
  - mainPanel function output

```
# ui.R
shinyUI(fluidPage(
  titlePanel("title panel"),
## Title  of the shiny.
  sidebarLayout(
    sidebarPanel( "sidebar panel"), ## SideBar panel title
    mainPanel("main panel")         ## Main panel title
  )
))
```

## Layout

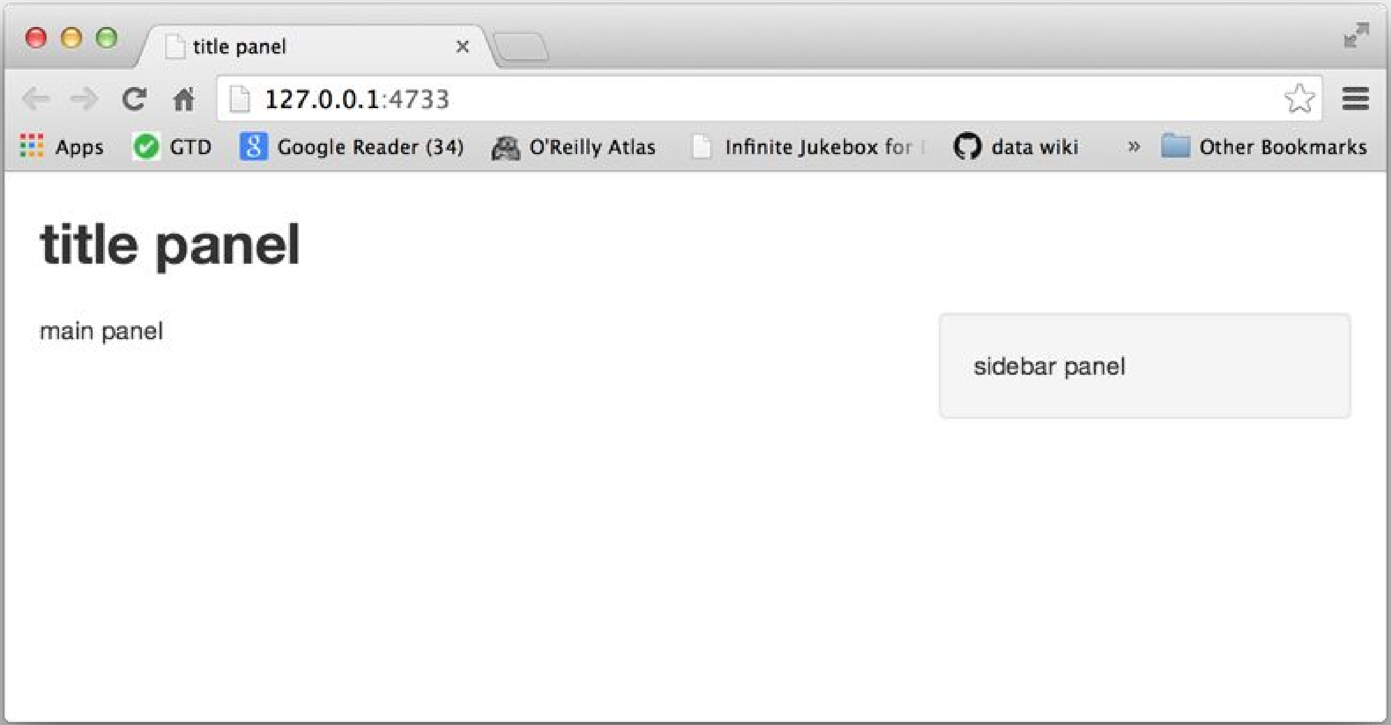
- Switch sidebar Panel to right
  - Add argument: position="right"

```
shinyUI(fluidPage(
  titlePanel("title panel"),

  sidebarLayout(position = "right",
    sidebarPanel( "sidebar panel"),
    mainPanel("main panel")
  )
))
```

## Layout

❖ Switch sidebar Panel to right

➢ Add argument: position="right"



title panel

main panel

sidebar panel

## Layout

- More layout design
  - Grid Layout
  - Tabsets
  - Navlists
  - Navbar Pages
  - etc. http://shiny.rstudio.com/articles/layout-guide.html

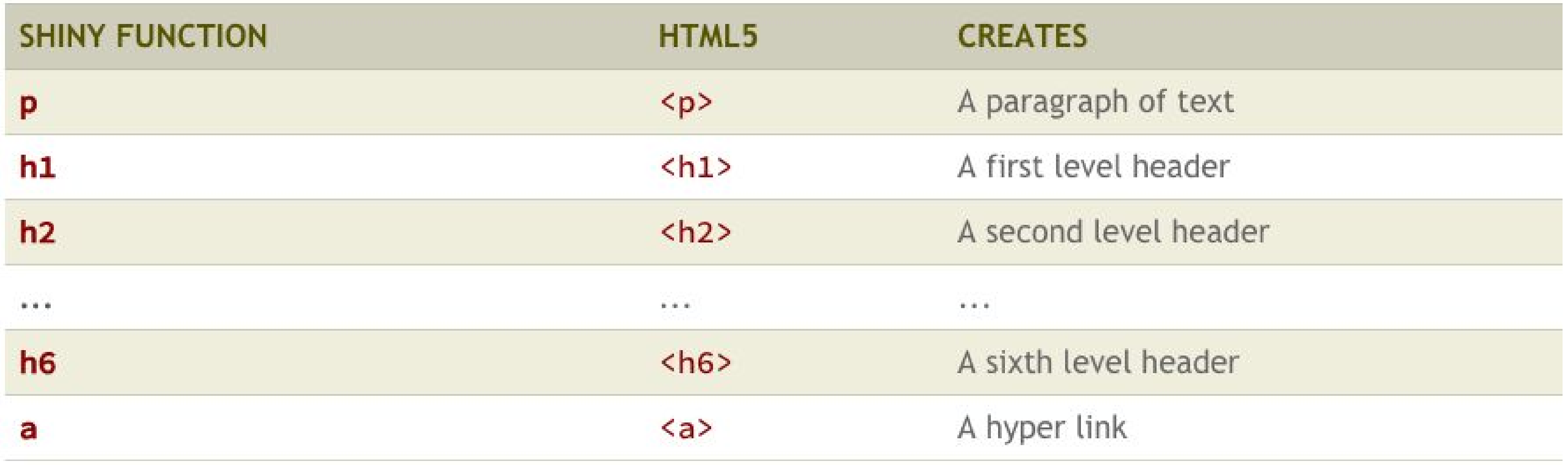
## HTML Content

- ❖ Add content to your Shiny app by placing it inside a *Panel function.
- ❖ For example:
  - ➢ Add the character string to the sidebarPanel function

```
# ui.R

shinyUI(fluidPage(
  titlePanel("title panel"),

  sidebarLayout(
    sidebarPanel( "sidebar panel"),
    mainPanel("main panel")
  )
))
```

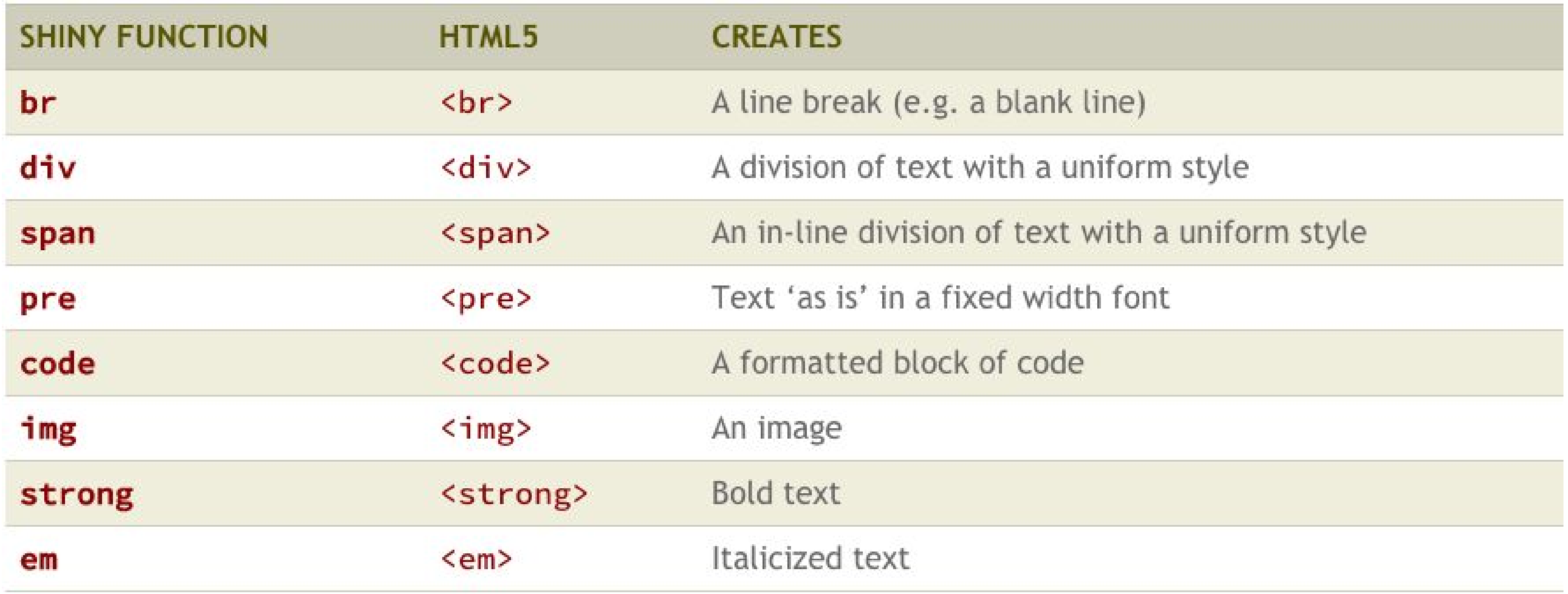
## HTML Content

- Function for HTML5
  - Use one of Shiny's HTML tag functions to add more advanced content.
  - These functions parallel common HTML5 tags.



| SHINY FUNCTION | HTML5 | CREATES |
|---|---|---|
| p | `<p>` | A paragraph of text |
| h1 | `<h1>` | A first level header |
| h2 | `<h2>` | A second level header |
| ... | ... | ... |
| h6 | `<h6>` | A sixth level header |
| a | `<a>` | A hyper link |

## HTML Content



| SHINY FUNCTION | HTML5 | CREATES |
|---|---|---|
| br | <br> | A line break (e.g. a blank line) |
| div | <div> | A division of text with a uniform style |
| span | <span> | An in-line division of text with a uniform style |
| pre | <pre> | Text ‘as is’ in a fixed width font |
| code | <code> | A formatted block of code |
| img | <img> | An image |
| strong | <strong> | Bold text |
| em | <em> | Italicized text |

NYC DATA SCIENCE ACADEMY

## Headers

- To create a header element:
  - Select a header function (e.g., h1 or h5)
  - Give it the text you want to see in the header

```
library(shiny)
h1("my title")
```

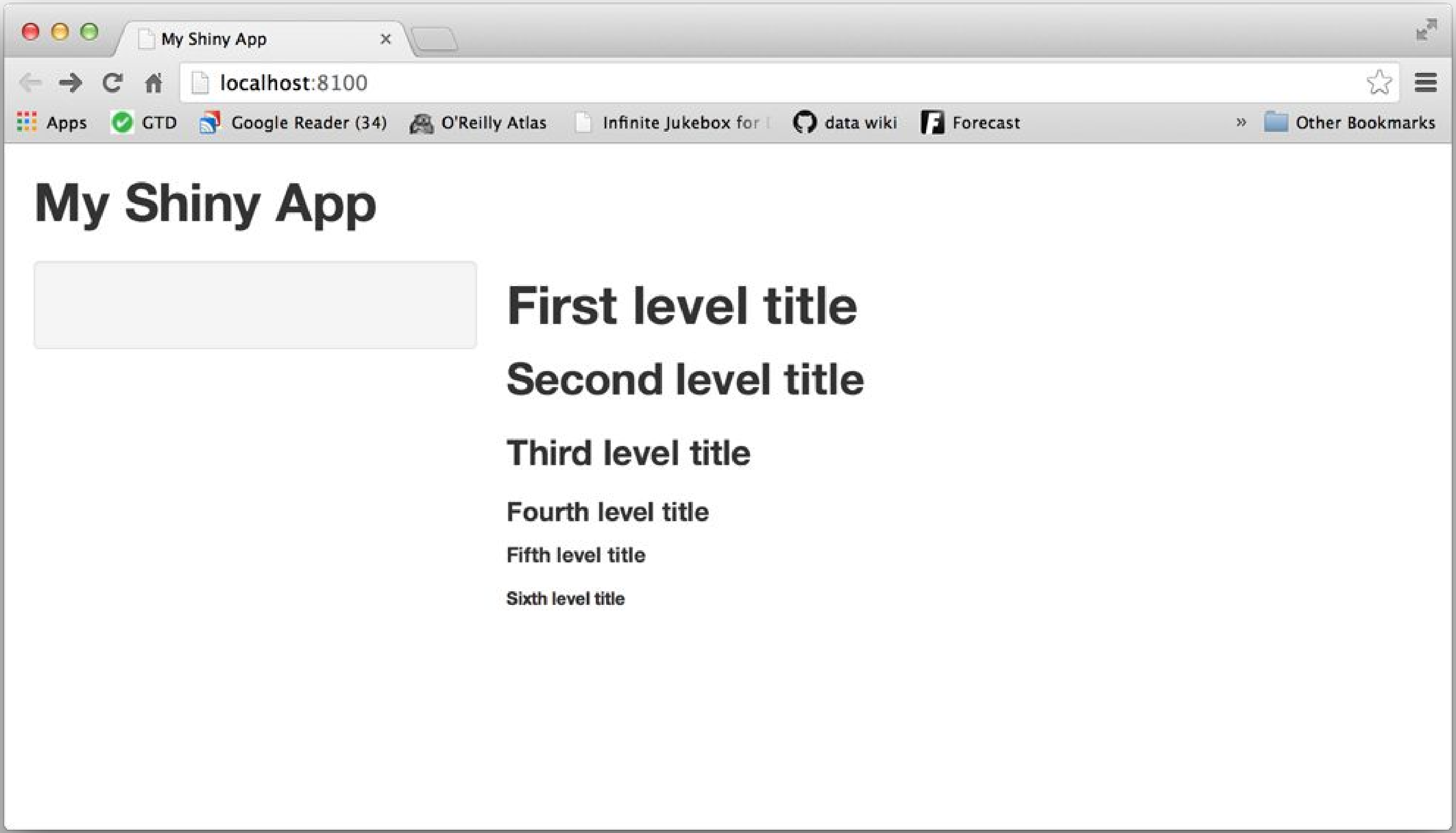
**my title**

## Headers

- Pass h1("my title") to titlePanel, sidebarPanel, or mainPanel
- Put the code in your ui.R and runApp().

```r
# ui.R
shinyUI(fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      h1("First level title"),
      h2("Second level title"),
      h3("Third level title"),
      h4("Fourth level title"),
      h5("Fifth level title"),
      h6("Sixth level title")
    )
  )
))
```

## Headers



My Shiny App

First level title

Second level title

Third level title

Fourth level title

Fifth level title

Sixth level title

## Headers

❖ align = "center" can be used to make the title place center

```
h2("My Title",align="center")
```

**My Title**

```
h3("h3 title in the center.",align="center")
```

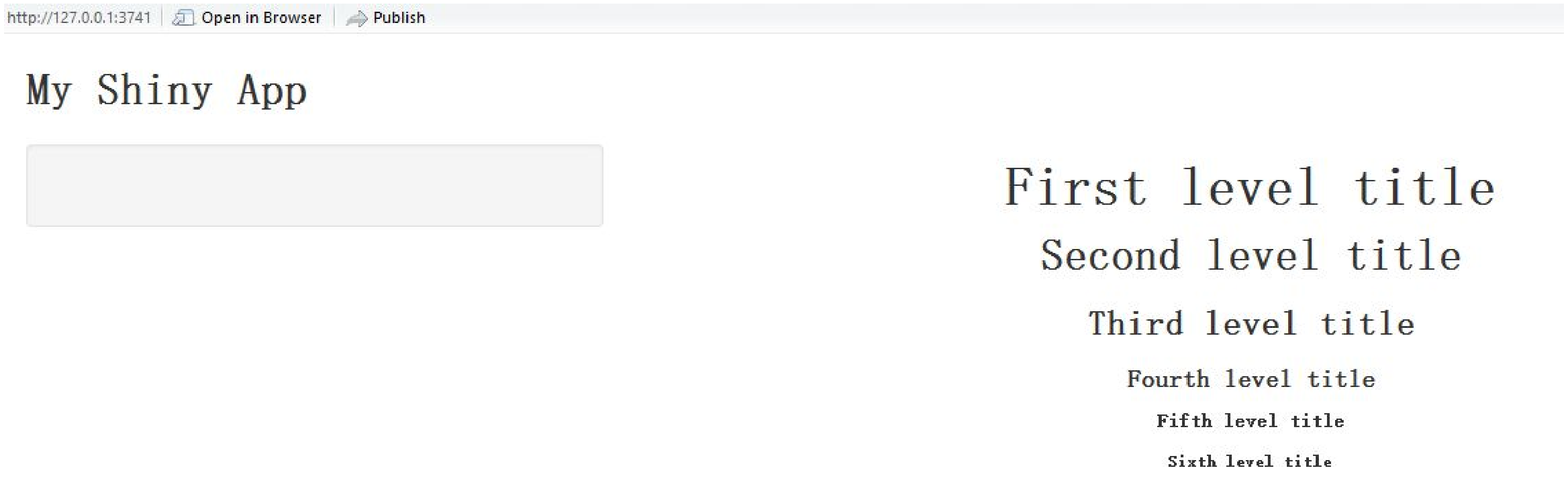
**h3 title in the center.**

## Headers: Example

- align = "center" can be used to make the title place center
  - Put the code in your ui.R and runApp().

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

## Headers: Example



My Shiny App

First level title

Second level title

Third level title

Fourth level title

Fifth level title

Sixth level title

## Formatted Text

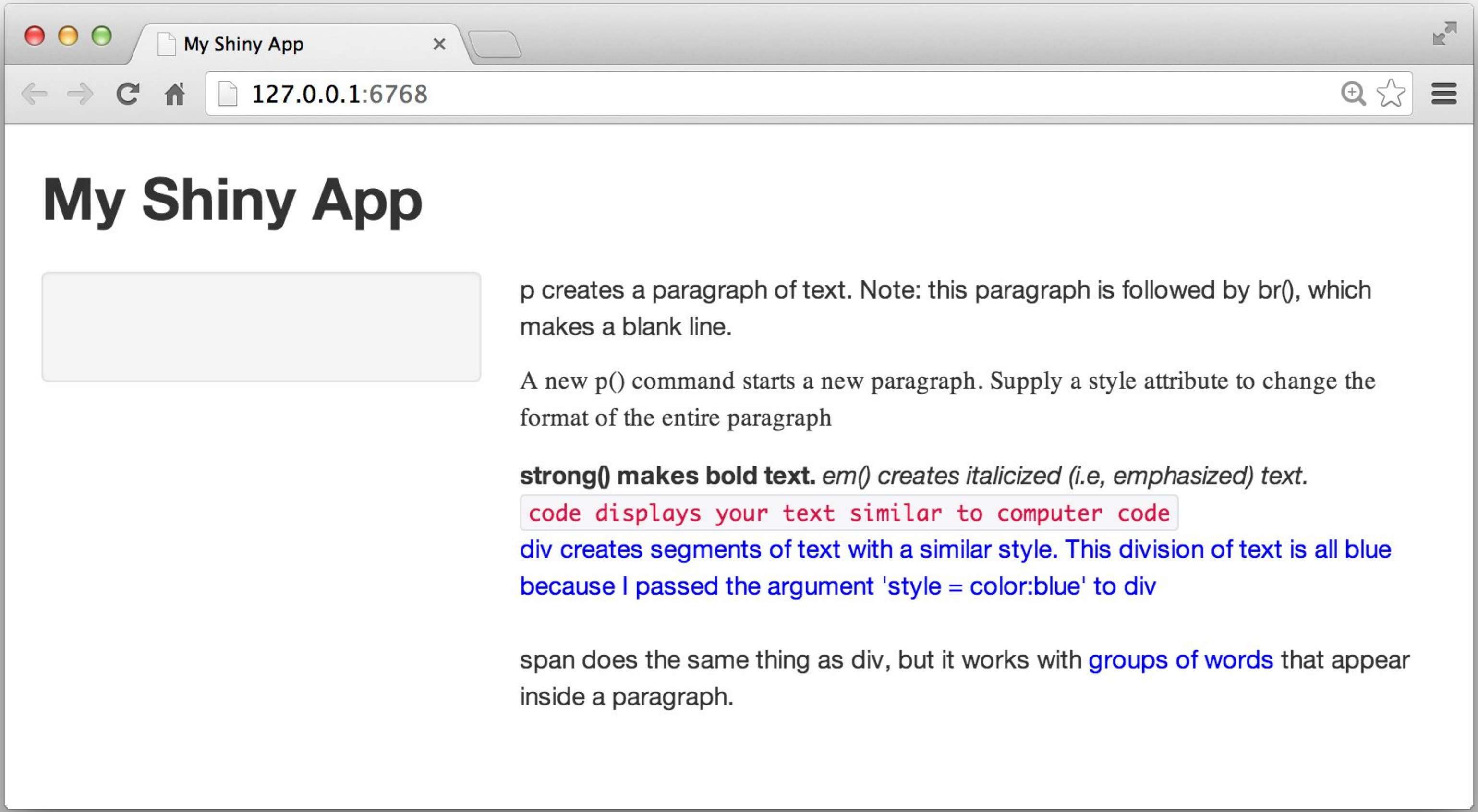
- Maybe you want:
  - some content bold
  - Need a line break
  - Put script into special type
  - Different color
- Time to use other functions

## Formatted Text

❖ Put the code in your ui.R and runApp().

```
shinyUI(fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout( sidebarPanel(),
    mainPanel(
      p("p creates a paragraph of text. Note: this paragraph is followed by br(),
        which makes a blank line."),
      p("A new p() command starts a new paragraph. Supply a style attribute to change
        the format of the entire paragraph",
        style = "font-family: 'times'; font-si16pt"),
      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"),
      br(),
      p("span does the same thing as div, but it works with",
        span("groups of words", style = "color:blue"),
        "that appear inside a paragraph.")
    )
  )
))
```

## Formatted Text



# My Shiny App

p creates a paragraph of text. Note: this paragraph is followed by br(), which makes a blank line.

A new p() command starts a new paragraph. Supply a style attribute to change the format of the entire paragraph

**strong() makes bold text.** *em() creates italicized (i.e, emphasized) text.*
`code displays your text similar to computer code`
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

span does the same thing as div, but it works with groups of words that appear inside a paragraph.

## Images

- Put a image in UI
- Download an image from here:
  - http://shiny.rstudio.com/tutorial/lesson2/www/bigorb.png
- Use the function img()

## Images

- To insert an image, give the img() function the name of your image file as the src argument

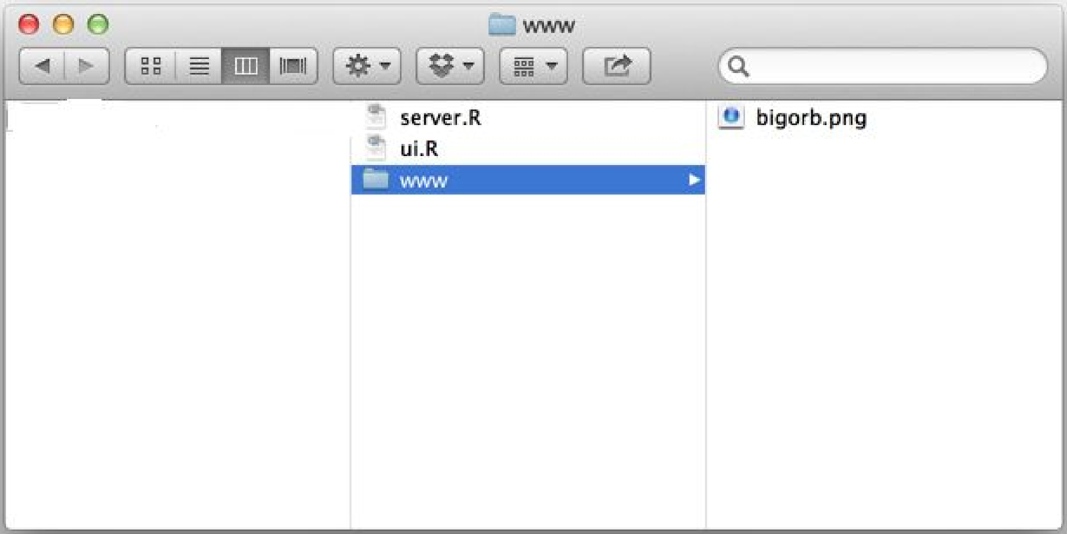
```
img(src = "my_image.png")
```

- Must spell out this argument since img passes your input to an HTML tag, and src is what the tag expects.
- Change the size by:

```
img(src = "my_image.png",height = 72, width = 72)
```

## Images

- The image must be in a folder named www in the same directory as the ui.R script.
  - Put the image into folder www
  - So if you want to use an image named bigorb.png, your www directory should look like this one:

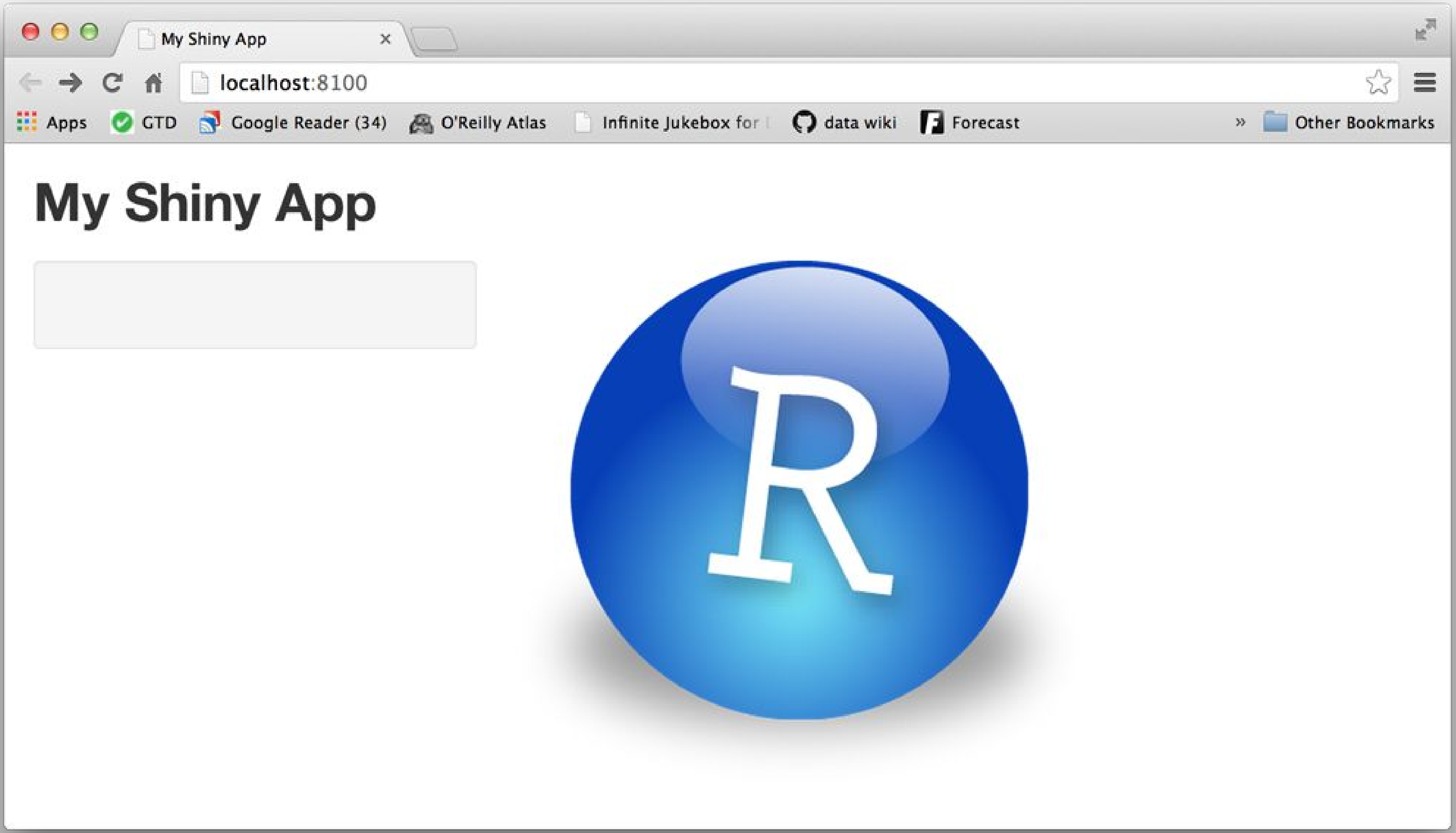


## Images

- Put images into *Panel
  - Put the code in your ui.R and runApp().

```
# ui.R
shinyUI(fluidPage(
  titlePanel("My Shiny App"),
  sidebarLayout(
    sidebarPanel(),
    mainPanel(
      img(src="bigorb.png", height = 400, width = 400)
    )
  )
))
```

## Images


My Shiny App
localhost:8100
Apps
GTD
Google Reader (34)
O'Reilly Atlas
Infinite Jukebox for
data wiki
Forecast
Other Bookmarks
My Shiny App

## Summary

- In this section we've learned
  - How to build a ui.R.
  - How to add different titles.
  - How to write different types of content.
  - How to add an image to your UI.

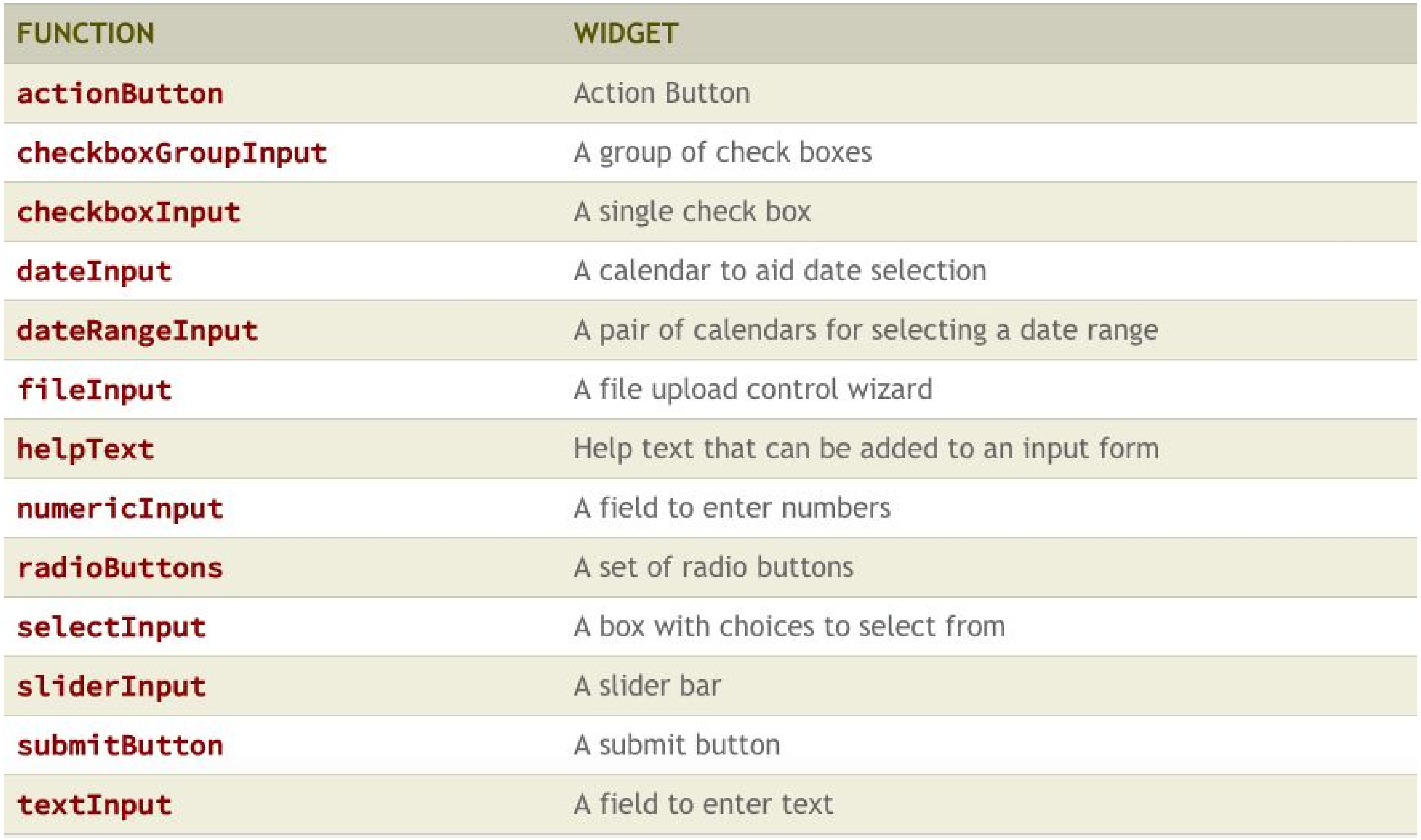
## Outline of today's class

- ❖ Shiny introduction
- ❖ Design the User-interface
- ❖ **Control Widgets**
- ❖ Build reactive output
- ❖ Use datatable in Shiny Apps

## What’s a Widget?

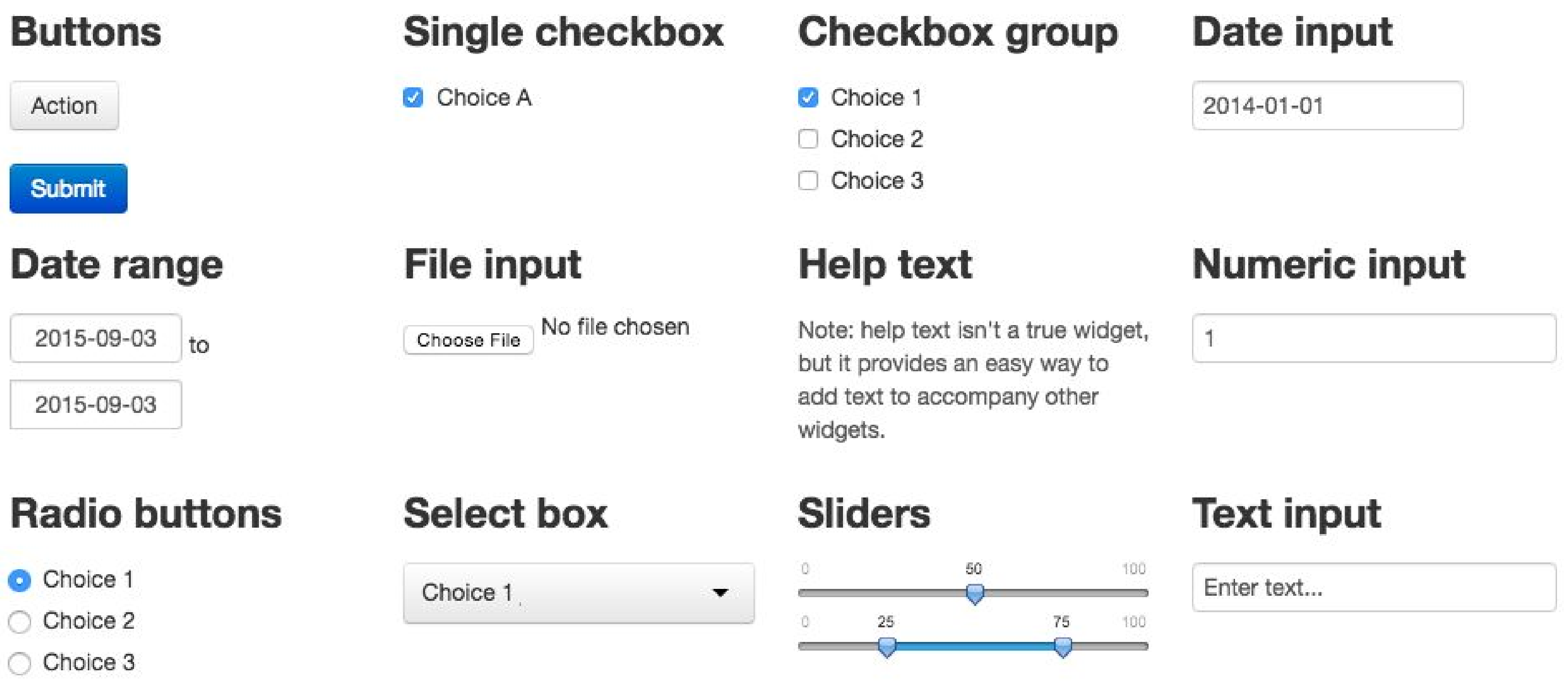
- A web element that users can interact with.
  - Widgets provide a way for users to send messages to the Shiny app.
- Shiny widgets collect a value from user.
  - When a user changes the widget, the value will change as well.
- These widgets come from the Twitter Bootstrap project, a popular open source framework for building user-interfaces.

## The Standard Shiny Widgets



| FUNCTION | WIDGET |
|---|---|
| `actionButton` | Action Button |
| `checkboxGroupInput` | A group of check boxes |
| `checkboxInput` | A single check box |
| `dateInput` | A calendar to aid date selection |
| `dateRangeInput` | A pair of calendars for selecting a date range |
| `fileInput` | A file upload control wizard |
| `helpText` | Help text that can be added to an input form |
| `numericInput` | A field to enter numbers |
| `radioButtons` | A set of radio buttons |
| `selectInput` | A box with choices to select from |
| `sliderInput` | A slider bar |
| `submitButton` | A submit button |
| `textInput` | A field to enter text |

## The Standard Shiny Widgets



Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

## Adding Widgets

- Add a widget to your app
  - place a widget function in sidebarPanel or mainPanel in your ui.R file.
- Each widget function requires several arguments.
  - The first two arguments for each widget are:
    - A name for the widget.
    - label.

## Adding Widgets

- Example : actionButton

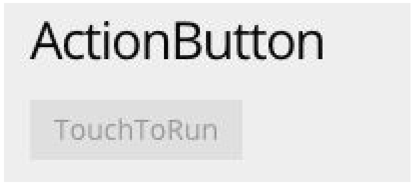
```
actionButton(inputId, label, icon = NULL, ...)
```

- inputId:
  - Specifies the input slot that will be used to access the value.
- label:
  - The contents of the button or link-usually a text label, but you could also use any other HTML, like an image.
- icon:
  - An optional icon to appear on the button

## Adding Widgets

- Example : actionButton
  - Action Button may look like this:

```
sidebarPanel(h3("ActionButton"),
       actionButton(1, "TouchToRun")
   )
```



## Adding Widgets

- Example : checkboxGroupInput

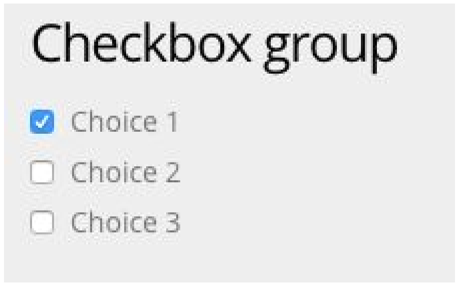
```
checkboxGroupInput(inputId, label, choices, selected = NULL)
```

- inputId:
  - Input variable to assign the control's value to.
- label:
  - Display label for the control.
- choices:
  - List of values to show checkboxes for. If elements of the list are named then the name rather than the value is displayed to the user.
- selected: The values that should be initially selected, if any.

## Adding Widgets

- Example : checkboxGroupInput
  - Checkbox Group Input may look as this:

```
sidebarPanel(checkboxGroupInput("checkGroup",
              label = h3("Checkbox group"),
              choices = list("Choice 1" = 1,
                             "Choice 2" = 2,
                             "Choice 3" = 3),selected = 1))
```


Checkbox group

☑ Choice 1
☐ Choice 2
☐ Choice 3

## Adding Widgets

- Example : checkboxInput

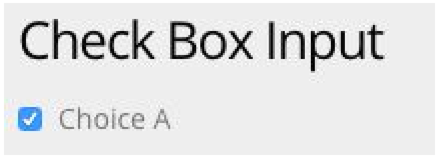
```
checkboxInput(inputId, label, value = FALSE)
```

- inputId
  - Input variable to assign the control's value to.
- label
  - Display label for the control.
- value
  - Initial value (TRUE or FALSE).

## Adding Widgets

- Example : checkboxInput
    - Checkbox Input may look as this:

```
checkboxInput("checkbox", label = "Choice A", value =TRUE)
```


Check Box Input

☑ Choice A

## Adding Widgets

- Example : dateInput

```
dateInput(inputId, label, value = NULL, min = NULL, max = NULL,
  format = "yyyy-mm-dd", startview = "month", weekstart = 0,
  language = "en")
```

- value:
  - The starting date. Either a Date object, or a string in yyyy-mm-dd format.
- min/max:
  - The minimum/maximum allowed date. Either a Date object, or a string in yyyy-mm-dd format.
- format (Defaults: "yyyy-mm-dd"):
  - The format of the date to display in the browser.

## Adding Widgets

- Example : dateInput

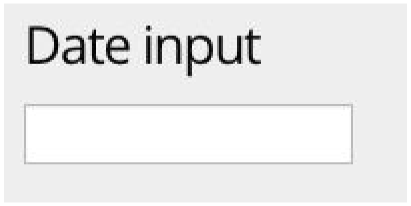
```
dateInput(inputId, label, value = NULL, min = NULL, max = NULL,
  format = "yyyy-mm-dd", startview = "month", weekstart = 0,
  language = "en")
```

- startview:
  - The date range shown when the input object is first clicked. "month" , "year", or "decade".
- weekstart:
  - Which day is the start of the week.
- language:
  - The language used for month and day names.

## Adding Widgets

❖ Example : dateInput

```
dateInput("date",
          label = h3("Date input"))
```


Date input

## Adding Widgets

- Example : dateRangeInput

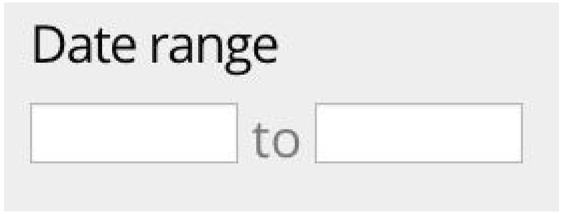
```
dateRangeInput(inputId, label, start = NULL,
end = NULL,separator = " to ",...)
```

- start:
  - The initial start date. Either a Date object, or a string in yyyy-mm-dd format.
- end:
  - The initial end date. Either a Date object, or a string in yyyy-mm-dd format.
- separator:
  - String to display between the start and end input boxes.
- args are same as dateInput

## Adding Widgets

❖ Example : dateRangeInput

```
dateRangeInput("dates", label = h3("Date range"))
```



## Adding Widgets

❖ Example : fileInput

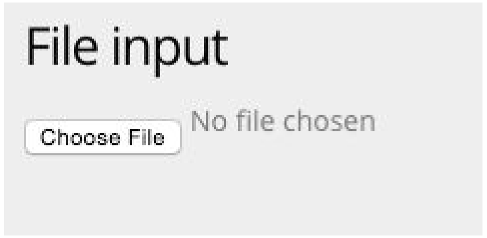
```
fileInput(inputId, label, multiple = FALSE, accept = NULL)
```

- multiple:
  - Whether the user should be allowed to select and upload multiple files at once.
- accept:
  - A character vector of MIME types; gives the browser a hint of what kind of files the server is expecting.

## Adding Widgets

❖ Example : fileInput

```
fileInput("file", label = h3("File input"))
```


File input

## Adding Widgets

- Example : helpText

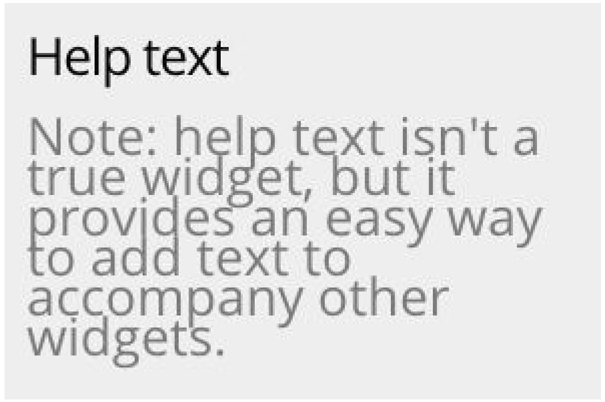
```
helpText(...)
```

  - One or more help text strings (or other inline HTML elements)

## Adding Widgets

❖ Example : helpText

```
sidebarPanel(
  h3("Help text"),
      helpText("Note: help text isn't a true widget,",
        "but it provides an easy way to add text to",
        "accompany other widgets."))
```


Help text

Note: help text isn't a true widget, but it provides an easy way to add text to accompany other widgets.

## Adding Widgets

- Example : numericInput

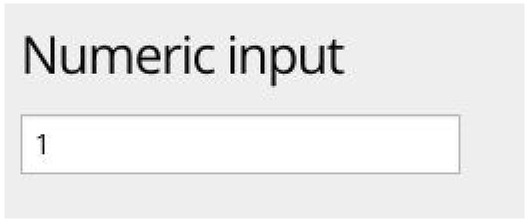
```
numericInput(inputId, label, value, min = NA,max = NA,step = NA)
```

- value:
  - Initial value
- min:
  - Minimum allowed value
- max:
  - Maximum allowed value
- step:
  - Interval to use when stepping between min and max

## Adding Widgets

❖ Example : numericInput

```
numericInput("num",
        label = h3("Numeric input"),
        value = 1)
```



## Adding Widgets

- Example : radioButtons

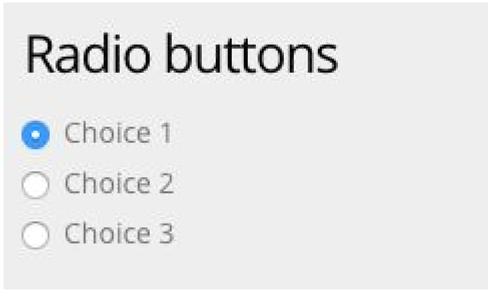
```
radioButtons(inputId, label, choices, selected = NULL)
```

- choices:
  - List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user)
- selected:
  - The initially selected value (if not specified then defaults to the first value)

## Adding Widgets

❖ Example : radioButtons

```
radioButtons("radio", label = h3("Radio buttons"),
        choices = list("Choice 1" = 1, "Choice 2" = 2,
                       "Choice 3" = 3),selected = 1)
```



Radio buttons

- [✓ Choice 1]
- Choice 2
- Choice 3

## Adding Widgets

- Example : selectInput

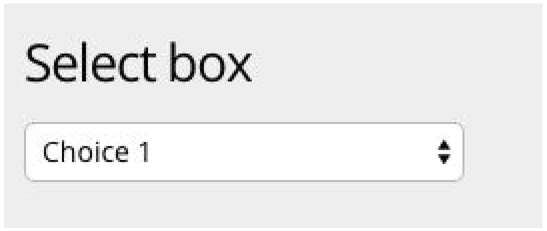
```
selectInput(inputId, label, choices, selected = NULL,
multiple = FALSE, selectize = TRUE, ...)
```

- choices:
  - List of values to select from.
- selected:
  - The initially selected value (or multiple values if multiple = TRUE).
- multiple:
  - Is selection of multiple items allowed?
- selectize:
  - Whether to use selectize.js or not

## Adding Widgets

❖ Example : selectInput

```
selectInput("select", label = h3("Select box"),
        choices = list("Choice 1" = 1, "Choice 2" = 2,
                       "Choice 3" = 3), selected = 1)
```



## Adding Widgets

- Example : sliderInput

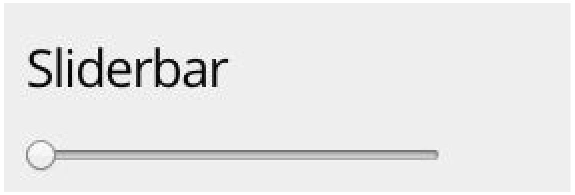
```
sliderInput(inputId, label, min, max, value, step = NULL,
       round = FALSE,...)
```

- min/max:
  - The minimum/maximum value (inclusive) that can be selected.
- value:
  - The initial value of the slider.
- step:
  - Specifies the interval between each selectable value on the slider (NULL means no restriction).
- round:
  - TRUE to round all values to the nearest integer; FALSE if no rounding is desired; or an integer to round to that number of digits .

## Adding Widgets

❖ Example : sliderInput

```
  sidebarPanel(
      sliderInput("obs","Sliderbar", 0, 1000, 0)
   )
```



## Adding Widgets

- Example : submitButton

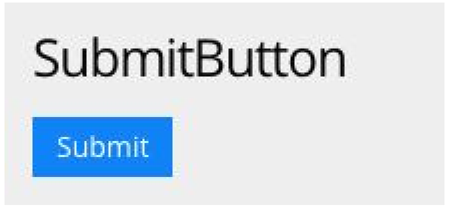
```
submitButton(text = "Apply Changes", icon = NULL, ...)
```

- text:
  - Button caption
- icon:
  - Optional icon to appear on the button

## Adding Widgets

❖ Example : submitButton

```
submitButton("Submit")
```



## Adding Widgets

- Example : textInput

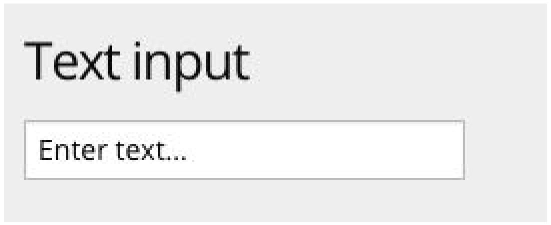
```
textInput(inputId, label, value = "", ...)
```

- inputId:
  - Input variable to assign the control's value to
- label:
  - Display label for the control
- value:
  - Initial value

## Adding Widgets

❖ Example : textInput

```
textInput("text", label = h3("Text input"),
        value = "Enter text...")
```



Text input

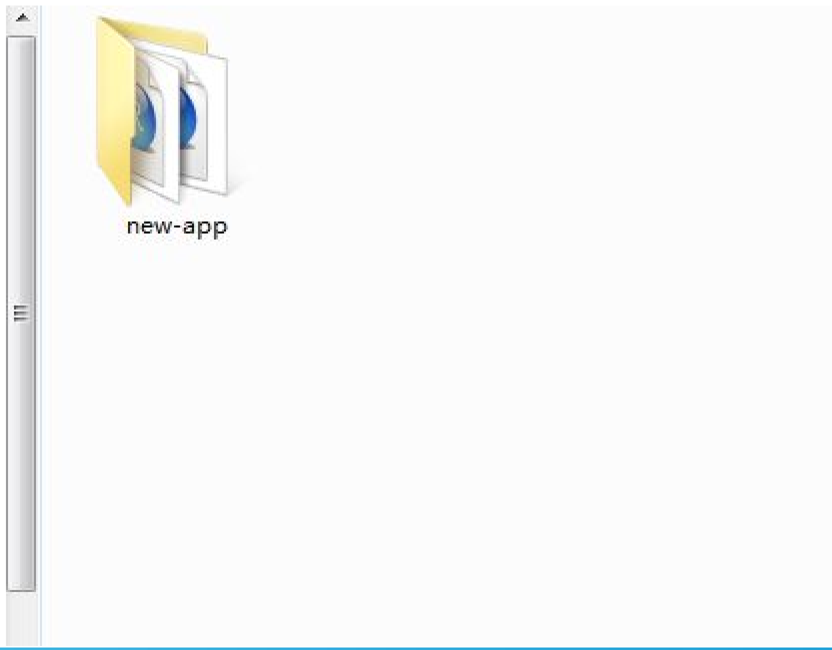
Enter text...

## Outline of today's class

- Shiny introduction
- Design the User-interface
- Control Widgets
- **Build reactive output**
- Use datatable in Shiny Apps

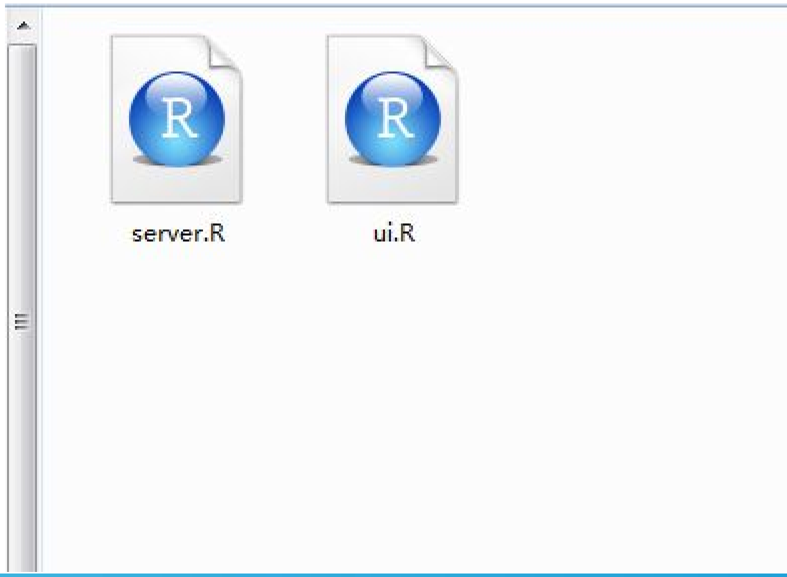
## Preparation

- Create a folder in your working directory named new_app.
- Save the ui.R and server.R files that you make in this section in new_app.



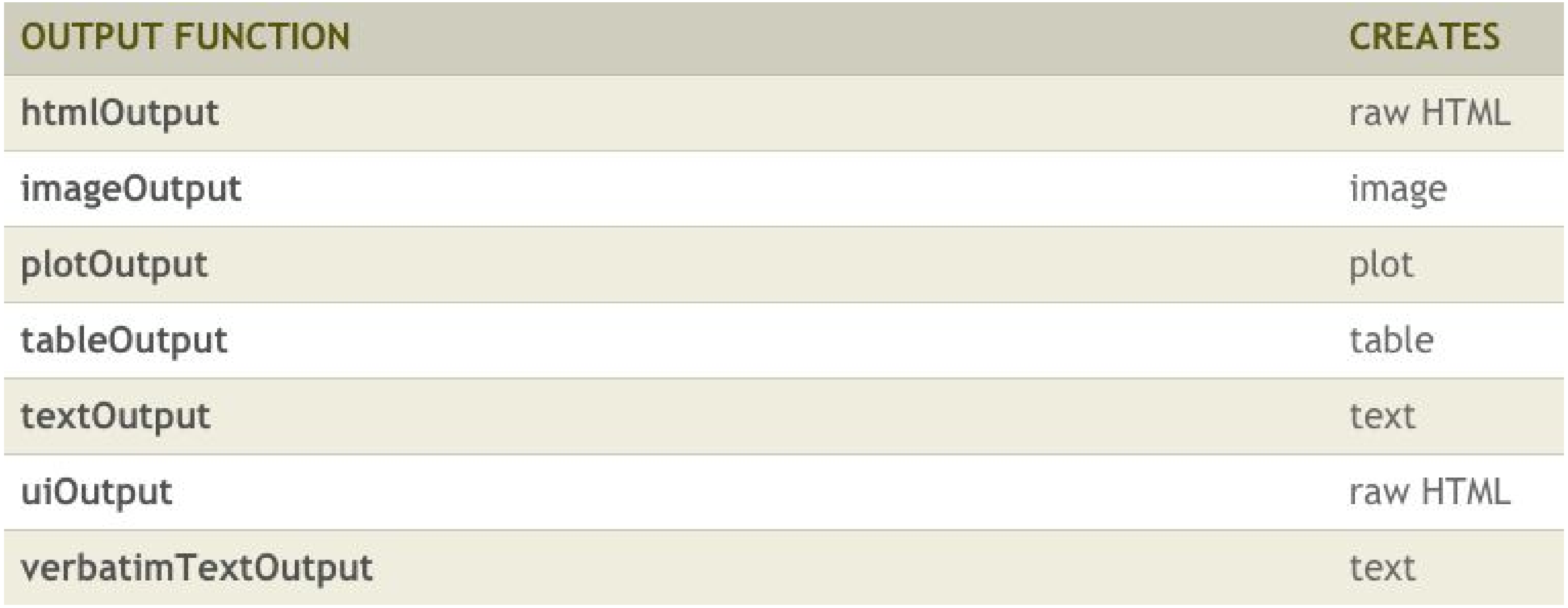
## Two Steps

- You can create reactive output with a two step process.
  - Add an R object to your user-interface with ui.R.
  - Tell Shiny how to build the object in server.R.
  - The object will be reactive if the code that builds it calls a widget value.



## Create UI

- There are several output functions for creating different type of outputs.



| OUTPUT FUNCTION | CREATES |
|---|---|
| htmlOutput | raw HTML |
| imageOutput | image |
| plotOutput | plot |
| tableOutput | table |
| textOutput | text |
| uiOutput | raw HTML |
| verbatimTextOutput | text |

- Place the output function inside sidebarPanel or mainPanel in the ui.R script.

## Add R Objects to ui.R

- Example for ui.R
  - Use helpText, selectInput, sliderInput to input values

```
shinyUI(fluidPage(
  titlePanel("RATE ME!"),
    sidebarLayout(
      sidebarPanel(
        helpText("What do you think about this app?"),
        selectInput("var",
                  label = "Choose one to display",
                  choices = c("awesome", "fantastic",
                     "admirable", "wonderful"),
                   selected = "awesome"),
        sliderInput("range",
                  label = "Percent:",
                  min = 90, max = 100, value = 100)
    ),
```

## Add R Objects to ui.R

❖ Example for ui.R

```
mainPanel(
      textOutput("text1")
    )
  )
))
```

- ➢ Use textOutput in mainPanel to show where to place the output.
- ➢ Each of the *Output functions require a single argument:
- ➢ a character string that Shiny will use as the name of your reactive element.
- ➢ Your users will not see this name, but you will use it later.

## Build the Object in server.R

- ❖ Placing a function *Output in ui.R tells Shiny where to display your object.
- ❖ Provide R code to build the object in server.R.
- ❖ Place the R code in the unnamed function that appears inside shinyServer in your server.R script.

## Build the Object in server.R

- Code in the unnamed function that appears inside shinyServer.

```
# server.R

shinyServer(function(input, output) {

     output$text1 <- renderText({
          "You have selected this"
     })

  }
)
```

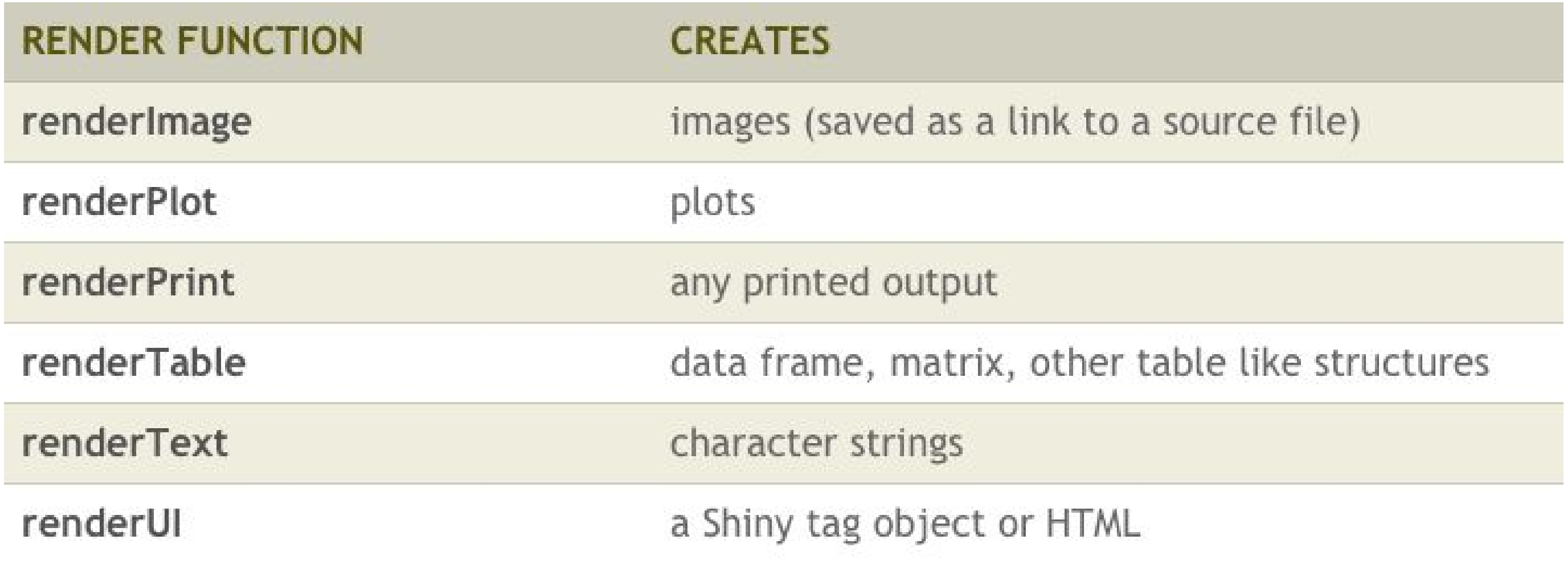
## Build the Object in server.R

- Make sure the element name match the name of the reactive element that you created in ui.R.
  - e.g. output$text1 matches textOutput("text1") in ui.R.

```
# server.R

shinyServer(function(input, output) {

     output$text1 <- renderText({
          "You have selected this"
     })

  }
)
```

## render* Function

- ❖ Each entry to output should contain the output of one of Shiny's render* functions.

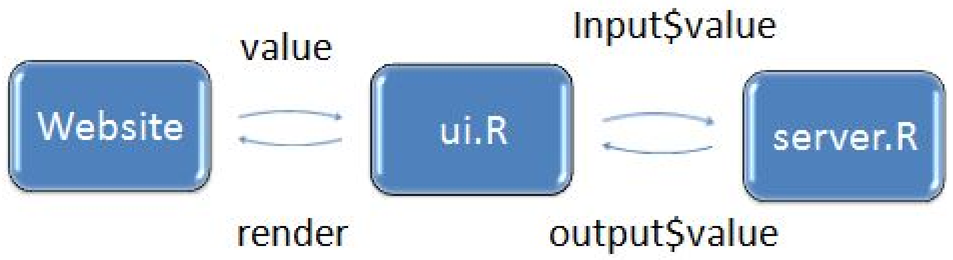


| RENDER FUNCTION | CREATES |
|---|---|
| renderImage | images (saved as a link to a source file) |
| renderPlot | plots |
| renderPrint | any printed output |
| renderTable | data frame, matrix, other table like structures |
| renderText | character strings |
| renderUI | a Shiny tag object or HTML |

- ❖ Each render* function takes a single argument: an R expression surrounded by braces, {}.

## Make Your Text Reactive

- Make your text reactive by asking Shiny to call a widget value.
- Use input and output in ui.R



- output stores instructions for building the R objects in your app.
- input stores the current values of all of the widgets in your app.

## Make Your Text Reactive

- Example : build a reactive text
- Assumed our app has two widgets: var and range.

```
# server.R

shinyServer(function(input, output) {

  output$text1 <- renderText({
    paste("This app is",input$range,"%",input$var,"!!!")
  })
  }
)
```

- Use runApp() to show your app

## Outline of today's class

- Shiny introduction
- Design the User-interface
- Control Widgets
- Build reactive output
- **Use datatable in Shiny Apps**

## Use Datatable in Shiny Apps

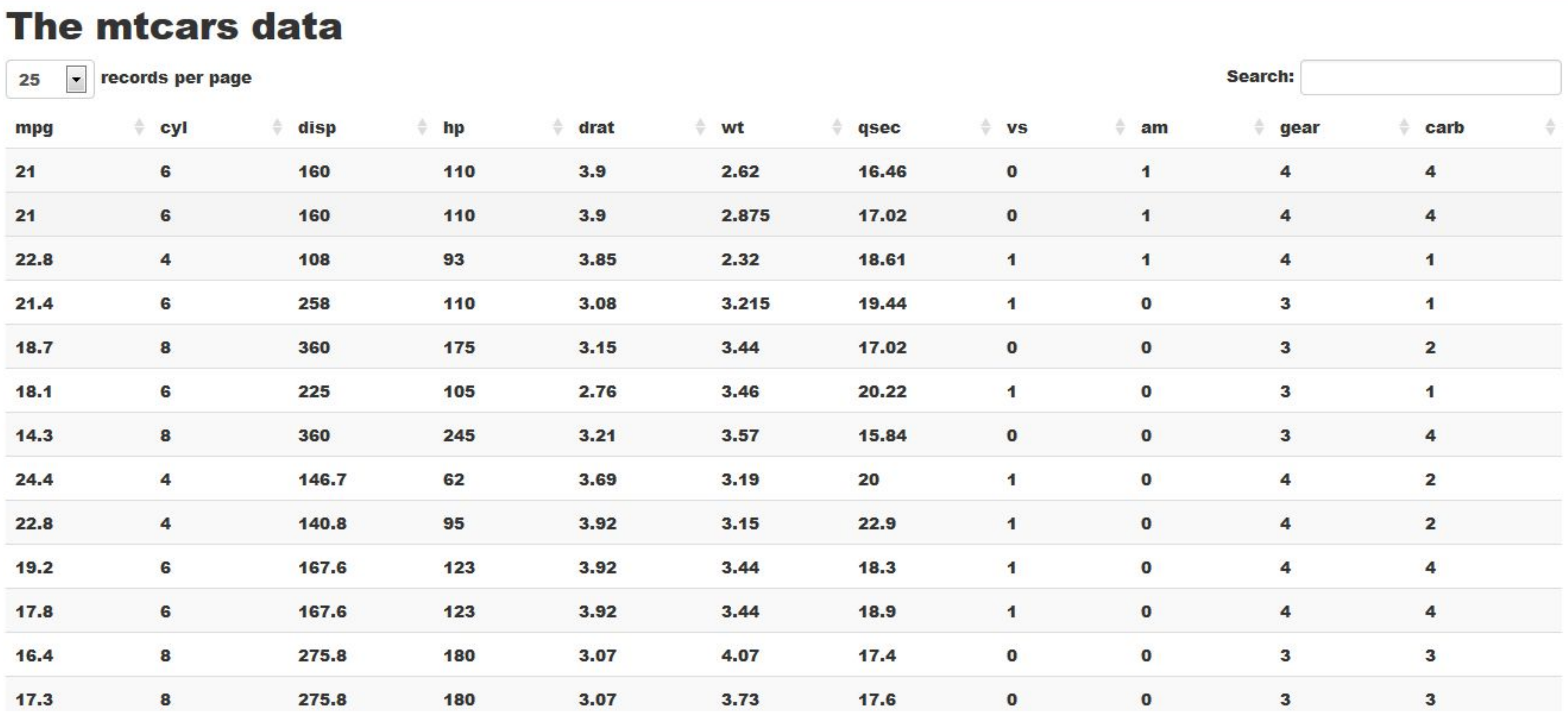
- renderDataTable()
  - The DataTables application demonstrates HTML tables using the jQuery library DataTables.
  - The basic usage:
  - To create an output element in the UI using

```
output$foo <- renderDataTable({ data })
```

  - Render a table on the server side using

```
dataTableOutput(id = 'foo')
```

## A Simple Example for renderDataTable



### The mtcars data

25 records per page

Search:

| mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 6 | 160 | 110 | 3.9 | 2.62 | 16.46 | 0 | 1 | 4 | 4 |
| 21 | 6 | 160 | 110 | 3.9 | 2.875 | 17.02 | 0 | 1 | 4 | 4 |
| 22.8 | 4 | 108 | 93 | 3.85 | 2.32 | 18.61 | 1 | 1 | 4 | 1 |
| 21.4 | 6 | 258 | 110 | 3.08 | 3.215 | 19.44 | 1 | 0 | 3 | 1 |
| 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17.02 | 0 | 0 | 3 | 2 |
| 18.1 | 6 | 225 | 105 | 2.76 | 3.46 | 20.22 | 1 | 0 | 3 | 1 |
| 14.3 | 8 | 360 | 245 | 3.21 | 3.57 | 15.84 | 0 | 0 | 3 | 4 |
| 24.4 | 4 | 146.7 | 62 | 3.69 | 3.19 | 20 | 1 | 0 | 4 | 2 |
| 22.8 | 4 | 140.8 | 95 | 3.92 | 3.15 | 22.9 | 1 | 0 | 4 | 2 |
| 19.2 | 6 | 167.6 | 123 | 3.92 | 3.44 | 18.3 | 1 | 0 | 4 | 4 |
| 17.8 | 6 | 167.6 | 123 | 3.92 | 3.44 | 18.9 | 1 | 0 | 4 | 4 |
| 16.4 | 8 | 275.8 | 180 | 3.07 | 4.07 | 17.4 | 0 | 0 | 3 | 3 |
| 17.3 | 8 | 275.8 | 180 | 3.07 | 3.73 | 17.6 | 0 | 0 | 3 | 3 |

## A Simple Example for renderDataTable

❖ The script is as follows:

(It's so simple that you don't need to create ui.R,server.R)

```
runApp(list(
  ui = basicPage(
    h2('The mtcars data'),
    dataTableOutput('mytable')
  ),
  server = function(input, output) {
    output$mytable = renderDataTable({
      mtcars
    })
  }
))
```

NYC DATA SCIENCE ACADEMY

## A Complete Example for renderDataTable

```
# ui.R
library(shiny)
library(ggplot2)  # for the diamonds dataset
shinyUI(pageWithSidebar(
  headerPanel('Examples of DataTables'),
  sidebarPanel(
    checkboxGroupInput('show_vars', 'Columns in diamonds to show:',
names(diamonds),
                       selected = names(diamonds)),
   helpText('For diamonds data, we can select variables to show in the table;
             for the mtcars example, we use bSortClasses = TRUE so that sorted
             columns are colored since they have special CSS classes attached;
             for the iris data, we customize the length menu so we can display 5
             rows per page.')
    ),
  mainPanel(
    tabsetPanel(
      tabPanel('diamonds', dataTableOutput("mytable1")),
      tabPanel('mtcars', dataTableOutput("mytable2")),
      tabPanel('iris', dataTableOutput("mytable3"))
    )
  )
))
```

## A Complete Example for renderDataTable

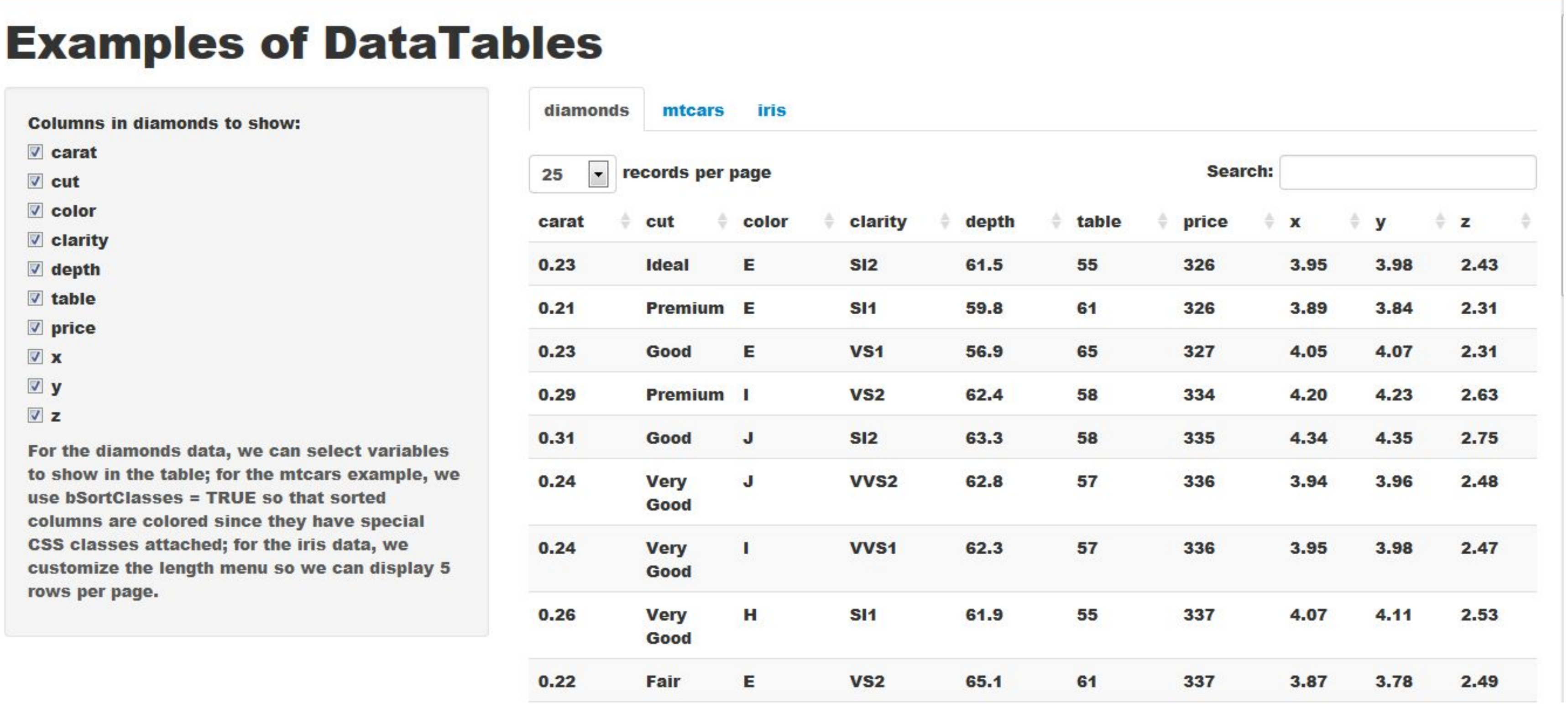
❖ Script of server.R

- ➢ Argument in renderDataTable() can take a list (literally an R list) of options, and pass them to DataTables when the table is initialized.
- ➢ For the mtcars data, we pass bSortClasses = TRUE to DataTables so that the sorted columns will have CSS classes attached on them (this is disabled by default).
- ➢ For the iris data, we pass the options aLengthMenu and iDisplayLength to customize the drop down menu, which has items [10, 25, 50, 100] by default.
- ➢ In iris now the menu has three items [5, 30, 50], and 5 is selected as the default value.

## A Complete Example for renderDataTable

```
shinyServer(function(input, output) {
  # a large table, reactive to input$show_vars
  output$mytable1 = renderDataTable({
    library(ggplot2)
    diamonds[, input$show_vars, drop = FALSE]
  })
  # sorted columns are colored now because CSS are attached to 
them
  output$mytable2 = renderDataTable({
    mtcars
  }, options = list(bSortClasses = TRUE))
  # customize the length drop-down menu; display 5 rows per page 
by default
  output$mytable3 = renderDataTable({
    iris
  }, options = list(aLengthMenu = c(5, 30, 50), iDisplayLength = 
5))
```

## A Complete Example for renderDataTable



# Examples of DataTables

**Columns in diamonds to show:**

- ☑ carat
- ☑ cut
- ☑ color
- ☑ clarity
- ☑ depth
- ☑ table
- ☑ price
- ☑ x
- ☑ y
- ☑ z

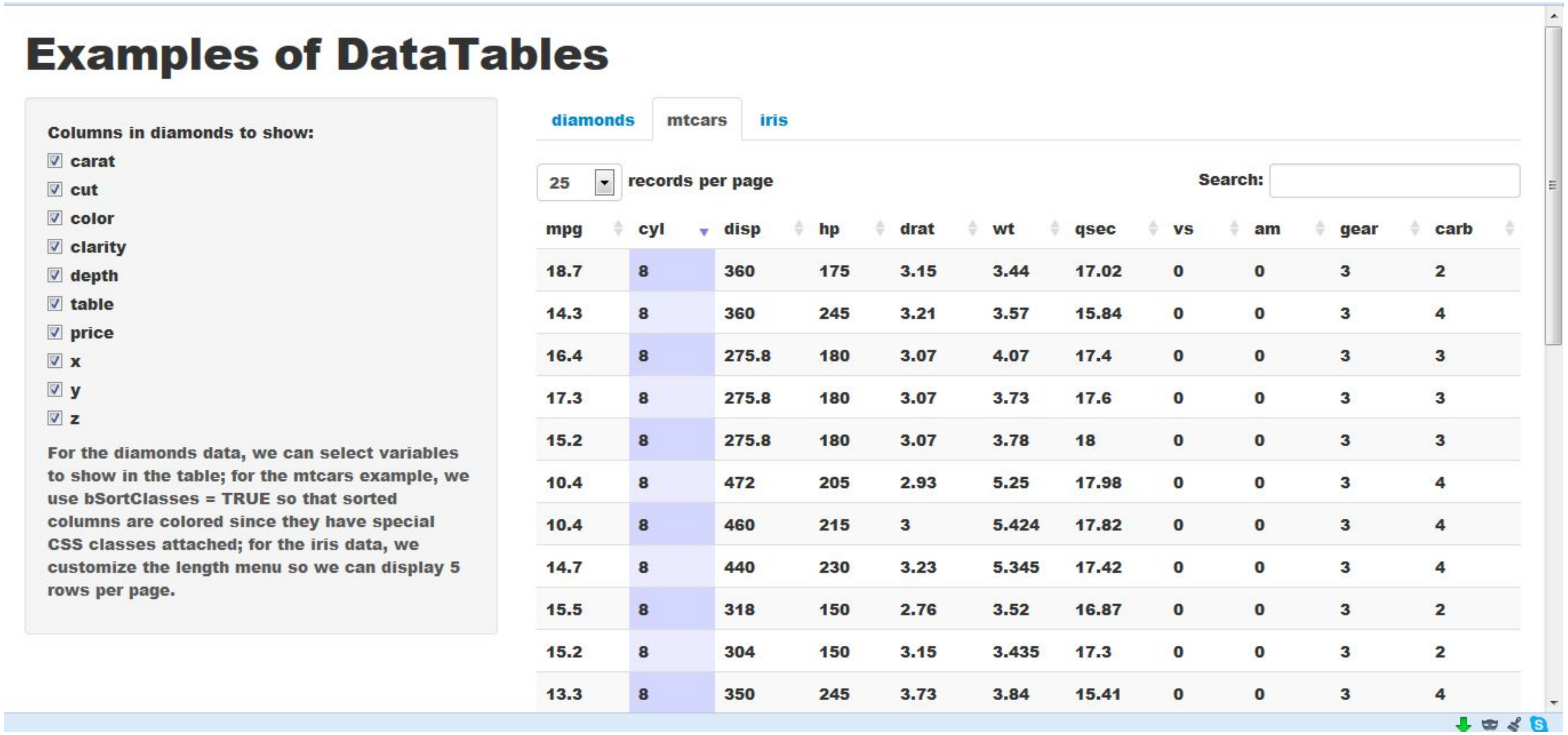
For the diamonds data, we can select variables to show in the table; for the mtcars example, we use bSortClasses = TRUE so that sorted columns are colored since they have special CSS classes attached; for the iris data, we customize the length menu so we can display 5 rows per page.

25 records per page

Search:

| carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|
| 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 0.23 | Good | E | VS1 | 56.9 | 65 | 327 | 4.05 | 4.07 | 2.31 |
| 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.20 | 4.23 | 2.63 |
| 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| 0.24 | Very Good | J | VVS2 | 62.8 | 57 | 336 | 3.94 | 3.96 | 2.48 |
| 0.24 | Very Good | I | VVS1 | 62.3 | 57 | 336 | 3.95 | 3.98 | 2.47 |
| 0.26 | Very Good | H | SI1 | 61.9 | 55 | 337 | 4.07 | 4.11 | 2.53 |
| 0.22 | Fair | E | VS2 | 65.1 | 61 | 337 | 3.87 | 3.78 | 2.49 |

## A Complete Example for renderDataTable



# Examples of DataTables

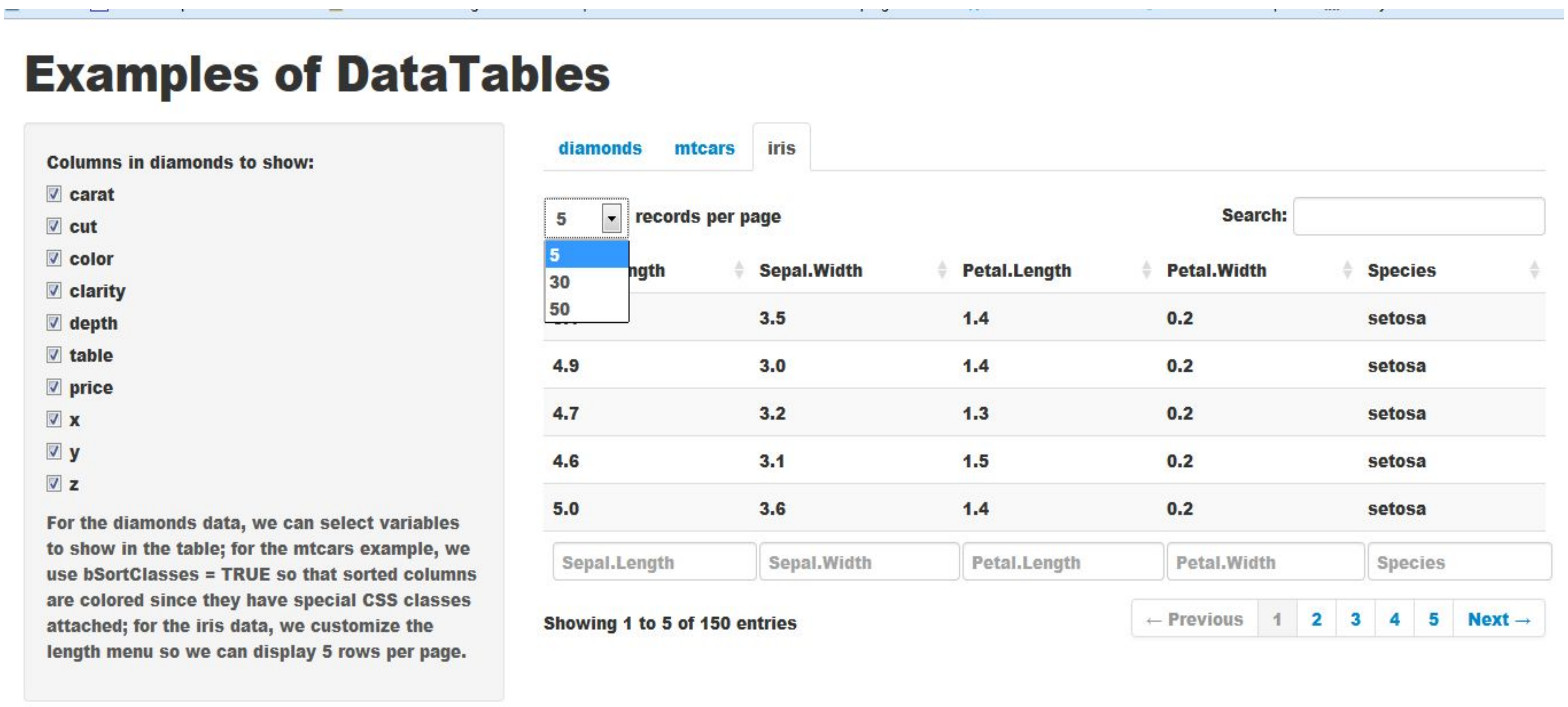
**Columns in diamonds to show:**

- ☑ carat
- ☑ cut
- ☑ color
- ☑ clarity
- ☑ depth
- ☑ table
- ☑ price
- ☑ x
- ☑ y
- ☑ z

For the diamonds data, we can select variables to show in the table; for the mtcars example, we use bSortClasses = TRUE so that sorted columns are colored since they have special CSS classes attached; for the iris data, we customize the length menu so we can display 5 rows per page.

| mpg | cyl | disp | hp | drat | wt | qsec | vs | am | gear | carb |
|---|---|---|---|---|---|---|---|---|---|---|
| 18.7 | 8 | 360 | 175 | 3.15 | 3.44 | 17.02 | 0 | 0 | 3 | 2 |
| 14.3 | 8 | 360 | 245 | 3.21 | 3.57 | 15.84 | 0 | 0 | 3 | 4 |
| 16.4 | 8 | 275.8 | 180 | 3.07 | 4.07 | 17.4 | 0 | 0 | 3 | 3 |
| 17.3 | 8 | 275.8 | 180 | 3.07 | 3.73 | 17.6 | 0 | 0 | 3 | 3 |
| 15.2 | 8 | 275.8 | 180 | 3.07 | 3.78 | 18 | 0 | 0 | 3 | 3 |
| 10.4 | 8 | 472 | 205 | 2.93 | 5.25 | 17.98 | 0 | 0 | 3 | 4 |
| 10.4 | 8 | 460 | 215 | 3 | 5.424 | 17.82 | 0 | 0 | 3 | 4 |
| 14.7 | 8 | 440 | 230 | 3.23 | 5.345 | 17.42 | 0 | 0 | 3 | 4 |
| 15.5 | 8 | 318 | 150 | 2.76 | 3.52 | 16.87 | 0 | 0 | 3 | 2 |
| 15.2 | 8 | 304 | 150 | 3.15 | 3.435 | 17.3 | 0 | 0 | 3 | 2 |
| 13.3 | 8 | 350 | 245 | 3.73 | 3.84 | 15.41 | 0 | 0 | 3 | 4 |

## A Complete Example for renderDataTable



# Examples of DataTables

Columns in diamonds to show:

- ☑ carat
- ☑ cut
- ☑ color
- ☑ clarity
- ☑ depth
- ☑ table
- ☑ price
- ☑ x
- ☑ y
- ☑ z

For the diamonds data, we can select variables to show in the table; for the mtcars example, we use bSortClasses = TRUE so that sorted columns are colored since they have special CSS classes attached; for the iris data, we customize the length menu so we can display 5 rows per page.

| Sepal.Length | Sepal.Width | Petal.Length | Petal.Width | Species |
|---|---|---|---|---|
| [illegible] | 3.5 | 1.4 | 0.2 | setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | setosa |

Showing 1 to 5 of 150 entries

## Summary

❖ In this section, you created your first reactive Shiny app. Including:

- ➢ use an *Output function in the ui.R script to place reactive objects in your Shiny app.
- ➢ use a render* function in the server.R script to tell Shiny how to build your objects.
- ➢ surround R expressions by braces, {}, in each render* function.
- ➢ save your render* expressions in the output list, with one entry for each reactive object in your app.
- ➢ create reactivity by including an input value in a render* expression.