

Building Shiny Apps

Dean Attali

<http://deanattali.com>

@daattali

These slides are meant as a teaching aid for my Shiny tutorial:

<http://deanattali.com/blog/building-shiny-apps-tutorial/>

What is Shiny?

- R package from RStudio
- Web application framework for R
- R code → interactive web page
- No HTML/CSS/JavaScript knowledge required
- Great for sharing R analysis with someone scared of R

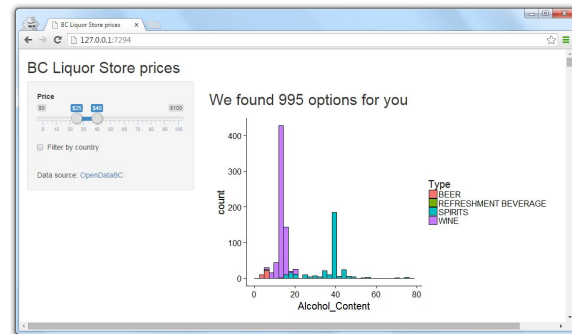
Examples

- Many examples by users <http://ShowMeShiny.com>
- Even complete websites! <http://letsrun.com/shoes>
- Explore cancer incidence data
<http://daattali.com/shiny/cancer-data/>
- What we'll build: BC Liquor Store prices explorer
<http://daattali.com/shiny/bcl/>

What is a Shiny app?

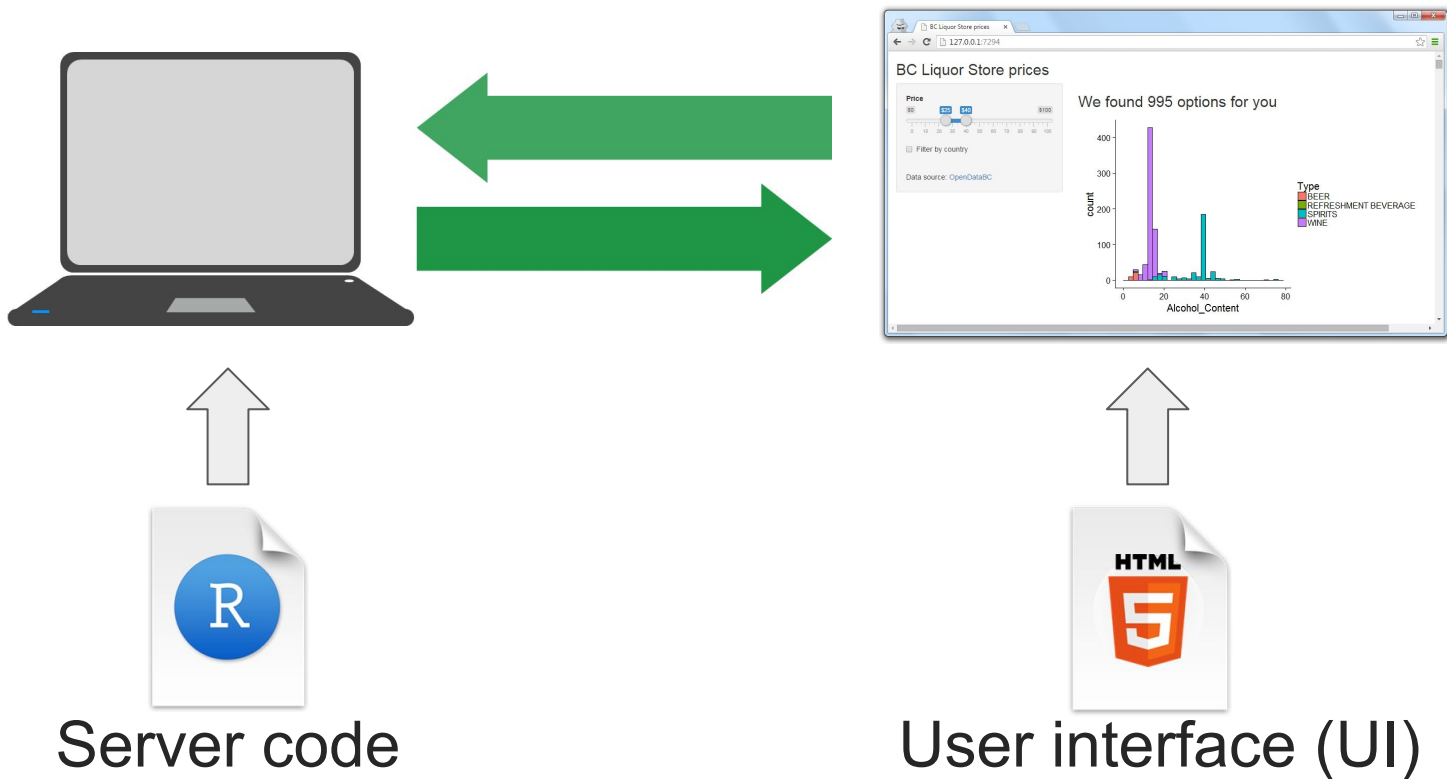


Computer
running R



Web page

What is a Shiny app?



Shiny app template

```
library(shiny)

ui <- fluidPage()

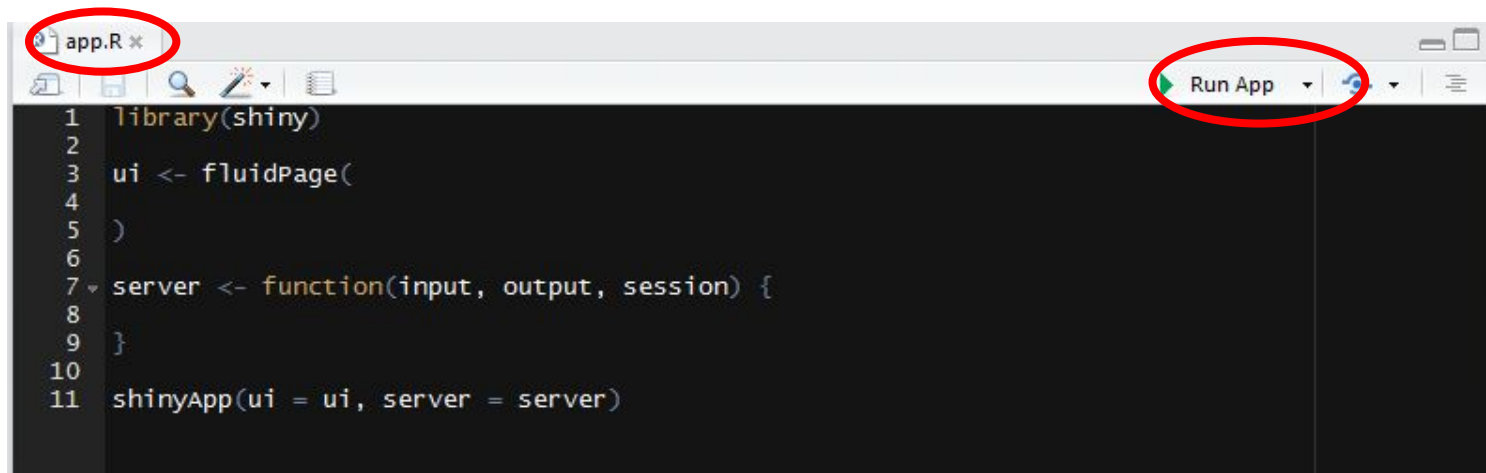
server <- function(input, output) {}

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

Important: Do not place any code after shinyApp(...)

Run Shiny app in RStudio, method 1

Save file as “**app.R**” → “Run” button turns to “Run App”



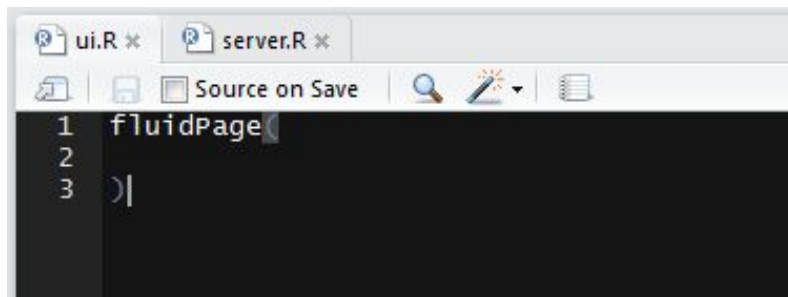
The screenshot shows the RStudio interface. The top-left pane displays the file name 'app.R' with a red circle around it. The top-right pane shows the 'Run App' button, also circled in red. The main editor pane contains the following R code:

```
1 library(shiny)
2
3 ui <- fluidPage(
4   )
5
6
7 server <- function(input, output, session) {
8
9 }
10
11 shinyApp(ui = ui, server = server)
```

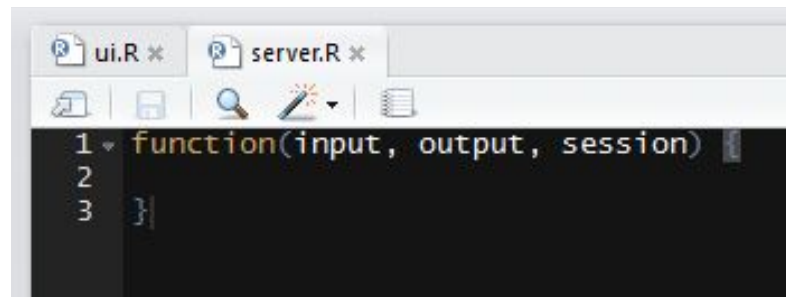
Good for creating Shiny apps quickly, all code in one file

Run Shiny app in RStudio, method 2

Save UI as “**ui.R**” and server as “**server.R**” in **same directory**

A screenshot of the RStudio editor window. The top tab bar shows two files: 'ui.R' and 'server.R'. The 'ui.R' file is active, and its content is visible in the editor. The code consists of three lines: '1 fluidPage', '2', and '3)|'. The line numbers are on the left margin. The editor has a dark background with light-colored text.

```
1 fluidPage
2
3 )|
```

A screenshot of the RStudio editor window. The top tab bar shows two files: 'ui.R' and 'server.R'. The 'server.R' file is active, and its content is visible in the editor. The code consists of three lines: '1 function(input, output, session)', '2', and '3 }|'. The line numbers are on the left margin. The editor has a dark background with light-colored text.

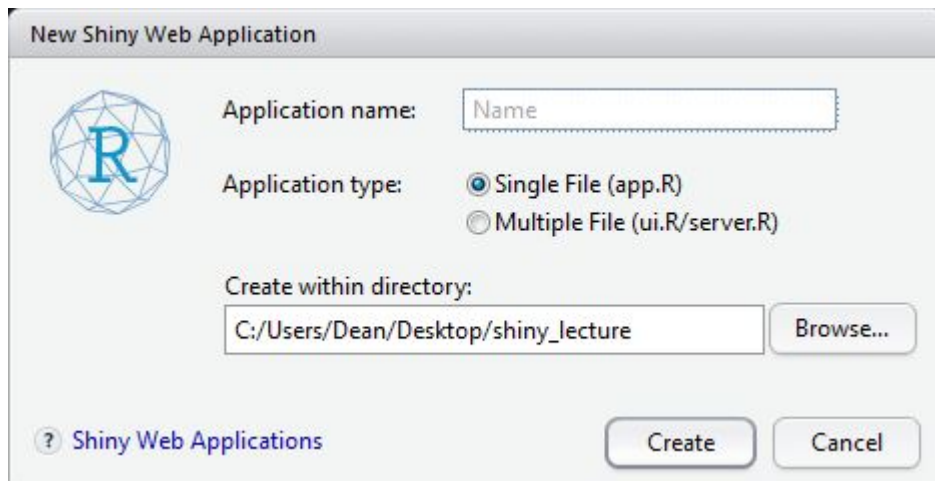
```
1 function(input, output, session)
2
3 }
```

Good for complex Shiny apps, separates view vs logic

If using this method, **do not** include a call to shinyApp(...)

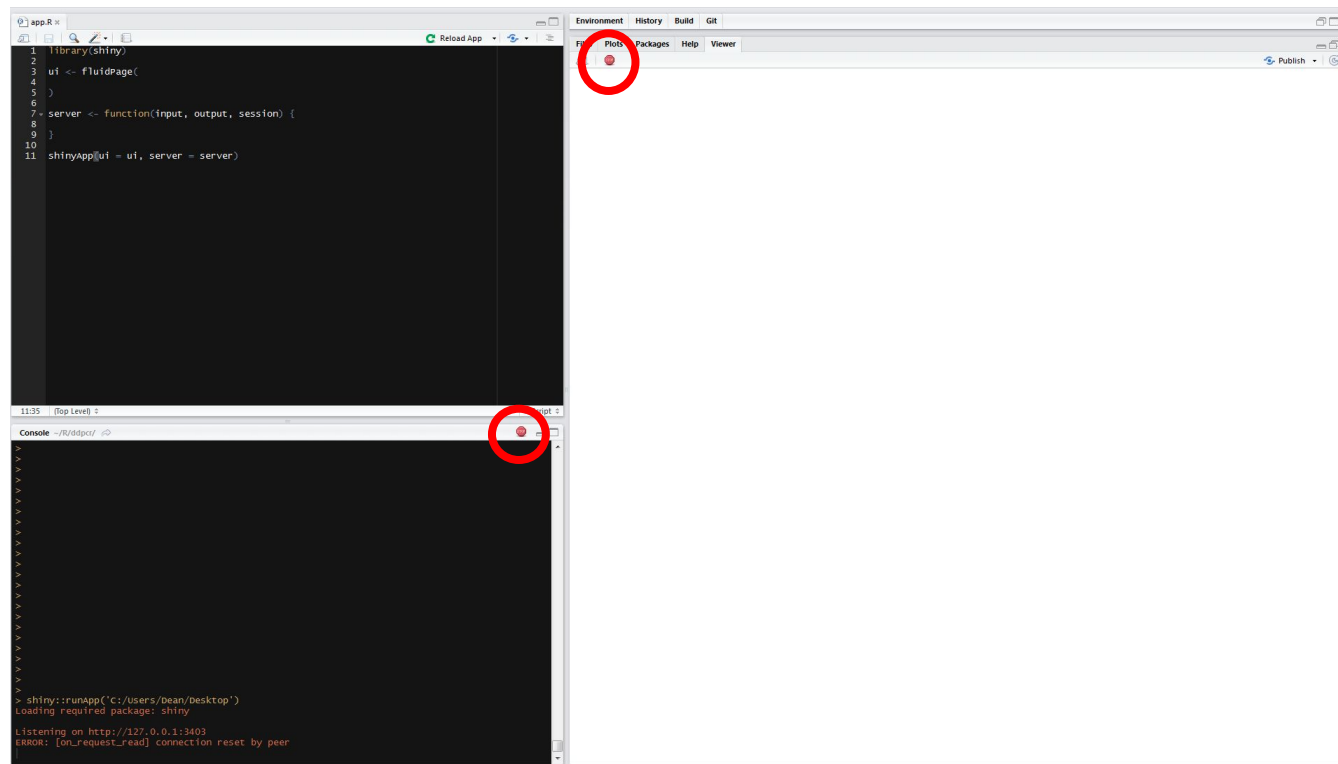
Run Shiny app in RStudio, method 3

File > New File > Shiny Web App...



Generates the template for you

Stop Shiny app in RStudio



Press
“Esc” or
click the
Stop icon

Work through the tutorial
until the end of Section 4

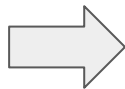
Add elements to app inside fluidPage()

```
library(shiny)

ui <- fluidPage("Hello STAT545")

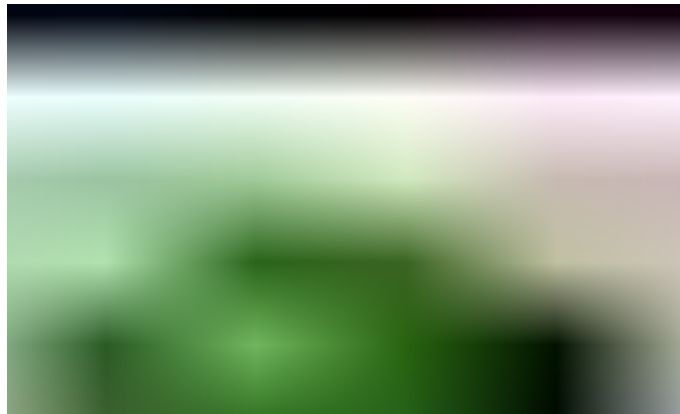
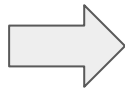
server <- function(input, output) {}

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



```
fluidPage(  
  h1("My Shiny app"),  
  "Hello STAT545"  
)
```

ui



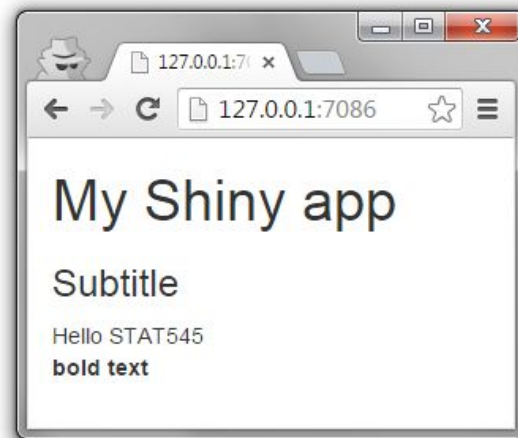
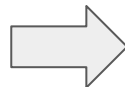
Add HTML to fluidPage()

- Remember the UI simply creates HTML
- Can use any HTML tags
 - <http://shiny.rstudio.com/articles/tag-glossary.html>
- **h1()** = header1, **br()** = line break, **strong()** = bold text
- Any HTML tag can be accessed using `tags` object
 - `h1 = tags$h1()`, `br = tags$br()`
- Common tags can be accessed without `tags`

Add HTML to fluidPage()

```
fluidPage(  
  h1("My Shiny app"),  
  h3("Subtitle"),  
  "Hello",  
  "STAT545",  
  br(),  
  strong("bold text")  
)
```

ui



Use a layout

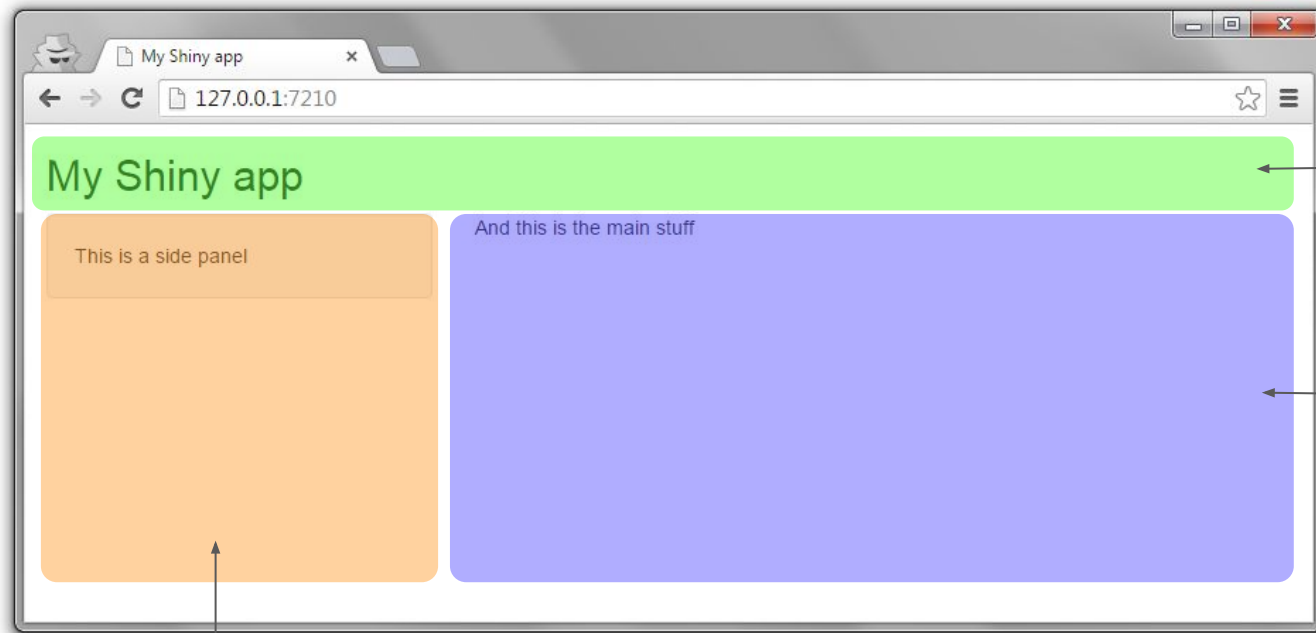
- By default, all elements stack up one after the other
- Can use different layouts
<http://shiny.rstudio.com/articles/layout-guide.html>
- We'll use **sidebarLayout()**

Use a layout - sidebarLayout()

```
fluidPage(  
  titlePanel("My Shiny app"),  
  sidebarLayout(  
    sidebarPanel(  
      "This is a side panel"  
    ),  
    mainPanel(  
      "And this is the main stuff"  
    )  
  )  
)
```

ui

Use a layout - sidebarLayout()



titlePanel()

mainPanel()

sidebarPanel()

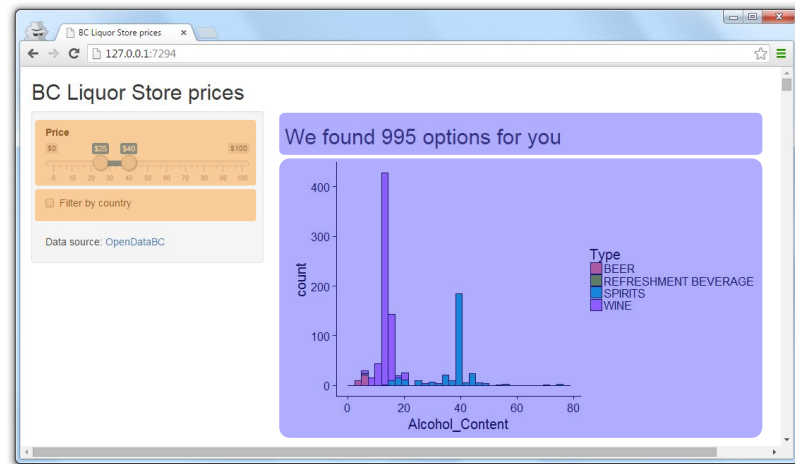
Work through the tutorial
until the end of Section 5

Inputs and outputs

- For interactivity, app needs inputs and outputs
- **Inputs** - things user can toggle
- **Output** - R objects user can see, often depend on inputs

```
fluidPage(  
  # *Input() functions,  
  # *Output() functions  
)
```

ui



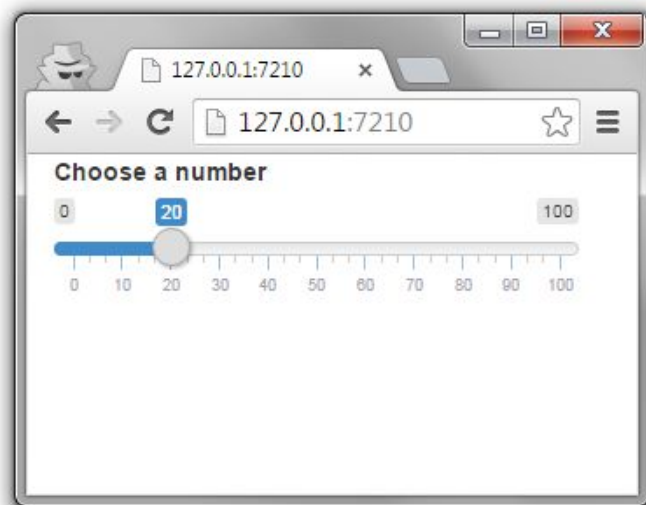
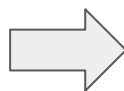
Inputs

```
library(shiny)

ui <- fluidPage(
  sliderInput(
    "num", "Choose a number",
    min = 0, max = 100,
    value = 20)
)

server <- function(input, output) {}

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



Inputs

```
sliderInput("num", "Choose a number",  
            min = 0, max = 100, value = 20)
```

```
<div class="form-group shiny-input-container">  
  <label class="control-label" for="num">Choose a number</label>  
  <input class="js-range-slider" id="num" data-min="0"  
data-max="100" data-from="20" data-step="1" data-grid="true"  
data-grid-num="10" data-grid-snap="false"  
data-prettify-separator="," data-keyboard="true"  
data-keyboard-step="1" data-drag-interval="true"  
data-data-type="number"/>  
</div>
```

Inputs

Buttons



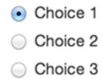
`actionButton()`
`submitButton()`

Date range



`dateRangeInput()`

Radio buttons



`radioButtons()`

Single checkbox



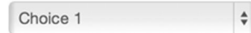
`checkboxInput()`

File input



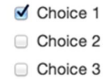
`fileInput()`

Select box



`selectInput()`

Checkbox group



`checkboxGroupInput()`

Numeric input



`numericInput()`

Sliders



`sliderInput()`

Date input



`dateInput()`

Password Input



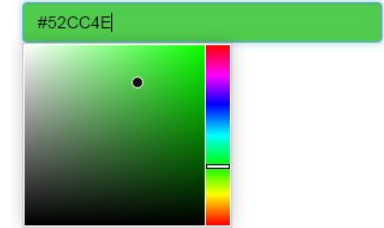
`passwordInput()`

Text input



`textInput()`

Colour input



`colourpicker::colourInput()`

Inputs

Input name Label to display Input-specific arguments

```
sliderInput("num", "Choose a number", min = 0, max = 100, value = 20)
```

```
*Input(inputId, label, ...)
```

What arguments can I pass to an input function?

```
?sliderInput
```


Work through the tutorial
until the end of Section 6

Outputs

- Plots, tables, text - anything that R creates and users see
- Initialize as empty placeholder space until object is created

Function	Outputs
<code>plotOutput()</code>	plot
<code>tableOutput()</code>	table
<code>uiOutput()</code>	Shiny UI element
<code>textOutput()</code>	text

Outputs

Output name Output-specific arguments

```
plotOutput("myplot", width = "300px")
```

*Output(outputId, ...)

What arguments can I pass to an output function?

```
?plotOutput
```

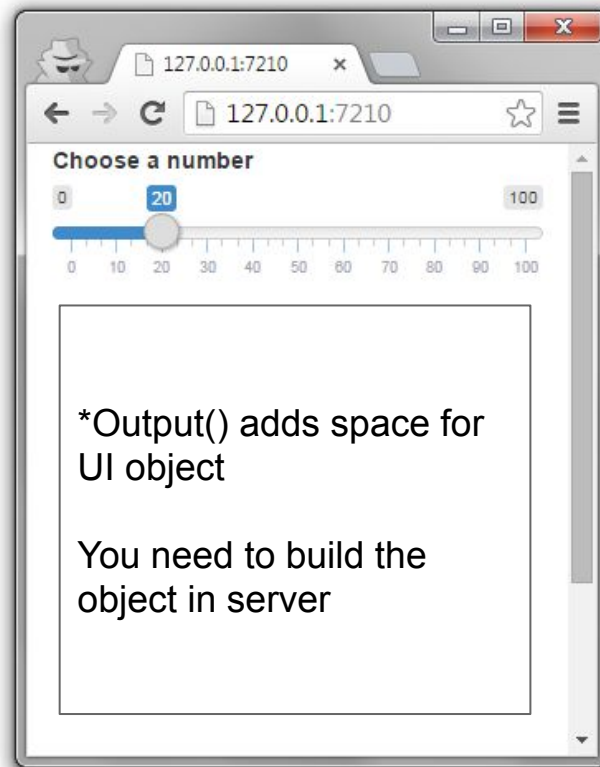
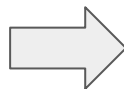
Outputs

```
library(shiny)

ui <- fluidPage(
  sliderInput("num", "Choose a number",
             0, 100, 20),
  plotOutput("myplot")
)

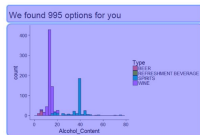
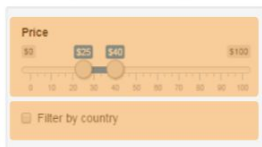
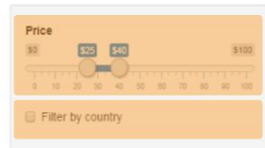
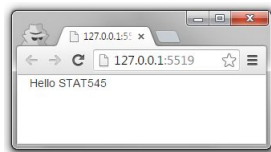
server <- function(input, output) {}

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



Summary

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```



Begin app with template

Add elements as arguments to **fluidPage()**

Create inputs with ***Input()** functions

Create outputs with ***Output()** functions

Use **server** to assemble inputs into outputs

Two most common “why isn’t my app running?!” problems

Remember to:

- comma-separate **all** the elements!
- **not** add comma to the **last** element!

Work through the tutorial
until the end of Section 8

Server: assemble input into outputs with 3 rules

```
server <- function(input, output) {  
  
  output$myplot <- renderPlot({  
    plot(rnorm(input$num))  
  })  
  
}
```


Building outputs

1 - Save objects into output\$

```
server <- function(input, output) {  
  
  output$myplot <- renderPlot({  
    plot(rnorm(input$num))  
  })  
  # in UI: plotOutput("myplot")  
}
```

Building outputs

2 - Build objects with render*

```
server <- function(input, output) {  
  
  output$myplot <- renderPlot({  
    plot(rnorm(input$num))  
  })  
  
}
```

***Output() → render*()**

Output function	Render function
plotOutput()	renderPlot({})
tableOutput()	renderTable({})
uiOutput()	renderUI({})
textOutput()	renderText({})

render*() functions build reactive output to display in UI

```
renderPlot({ plot(rnorm(100)) })
```

Type of object
to build

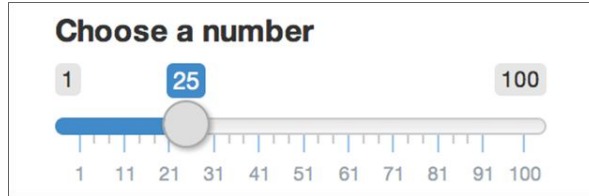
Code to build
the object

Building outputs

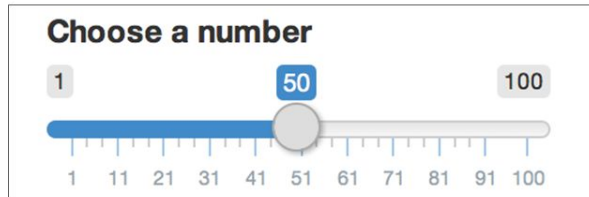
3 - Access input values with input\$

```
server <- function(input, output) {  
  output$myplot <- renderPlot({  
    plot(rnorm(input$num))  
  
    # in UI:sliderInput("num", ...)  
  })  
}
```

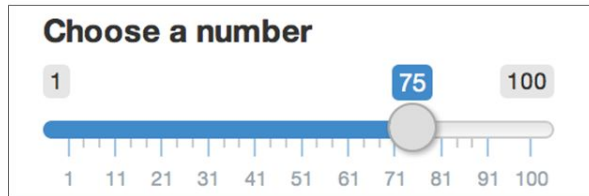
Using input\$



`input$num` returns 25



`input$num` returns 50



`input$num` returns 75

Work through the tutorial
until the end of Section 9

Reactivity

- Shiny uses **reactive programming**
- Supports **reactive variables**
 - When value of variable `x` changes, anything that relies on `x` is re-evaluated
 - Contrast with regular R:

```
x <- 5  
y <- x + 1  
x <- 10  
# y is still 6
```


Reactivity

- `input$num` is a **reactive** value

```
output$myplot <- renderPlot({  
  plot(rnorm(input$num))  
})
```

- `output$myplot` **depends on** `input$num`
 - `input$num` **changes** → `output$myplot` **reacts**
- All inputs are automatically reactive, so if you use any input inside a `render*` function, the output will re-render any time input changes

Reactive contexts

- You can define your own reactive variables
- Reactive values can only be used inside **reactive contexts**
- Any render* function is a reactive context
- Use `reactive({ ... })` to assign a reactive variable
- Use `observe({ ... })` to access a reactive variable
- Remember: reactive variable means anything that depends on it gets re-executed automatically

Reactive contexts

Assign variable

```
server <- function(input, output) {  
  x <- input$num + 1  
}  
# error
```

```
server <- function(input, output) {  
  x <- reactive({  
    input$num + 1  
  })  
}  
# OK
```

Access variable

```
server <- function(input, output) {  
  print(input$num)  
}  
# error
```

```
server <- function(input, output) {  
  observe({  
    print(input$num)  
  })  
}  
# OK
```

Simple Shiny app using basic reactivity

Single file

```
library(shiny)

ui <- fluidPage(
  sliderInput("num", "Choose a number",
             0, 100, 20),
  plotOutput("myplot")
)

server <- function(input, output) {
  output$myplot <- renderPlot({
    plot(seq(input$num))
  })
  x <- reactive({
    input$num + 1
  })
  observe({
    print(x())
  })
}

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

app.R



Two files

```
library(shiny)

fluidPage(
  sliderInput("num", "Choose a number",
             0, 100, 20),
  plotOutput("myplot")
)

library(shiny)

function(input, output) {
  output$myplot <- renderPlot({
    plot(seq(input$num))
  })
  x <- reactive({
    input$num + 1
  })
  observe({
    print(x())
  })
}
```

ui.R

server.R

Work through the tutorial
until the end of Section 10

Using buttons in the UI

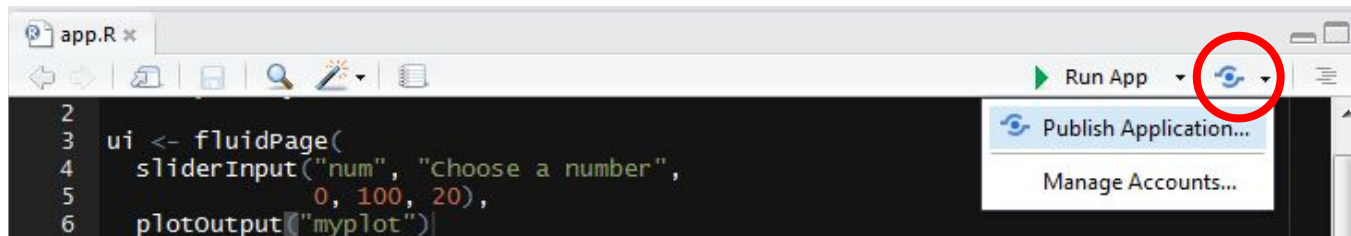
- Different from other inputs: you usually don't care about the "value" of the button, you care when it's clicked

```
ui <- fluidPage(  
  actionButton("btn", "Click me")  
)  
server <- function(input, output, session) {  
  observe({  
    cat(input$btn)  
  })  
}  
shinyApp(ui = ui, server = server)
```

Work through the tutorial
until the end of Section 12

Share your app: shinyapps.io

- Go to <http://www.shinyapps.io/> and make an account
- Make sure all your app files are in an isolated folder
- Click “Publish Application” in RStudio
 - You might be asked to install a couple packages



- Follow instructions from RStudio

Work through the tutorial
until the end of Section 13

PS. Shiny in Rmarkdown

- Set output: `html_document`
- Set runtime: `shiny`
- You can now use interactive inputs/outputs in Rmarkdown!

```
---  
output: html_document  
runtime: shiny  
---  
  
```${r echo=FALSE}  
sliderInput("num", "Choose a number",
 0, 100, 20)

renderPlot({
 plot(seq(input$num))
})
```,
```

PPS. More things to check out

- `?conditionalPanel` - conditionally show UI elements
- `global.R` - objects here are available to both `ui.R` and `server.R`
- Use `navbarPage()` or `tabsetPanel()` for multiple tabs in UI
- Use `DT::dataTableOutput()` instead of `tableOutput()` for an interactive table instead of ugly static table
- Add images by placing image under “`www/image.png`” and using UI function `img(src = "image.png")`
- Use `update*Input()` functions to update input values from R
- Know JavaScript/CSS? Use `includeScript()` or `includeCSS()`

Recommended add-on packages to Shiny

- leaflet (<http://rstudio.github.io/leaflet/>)
 - Add interactive maps to your apps
- shinyjs (<https://github.com/daattali/shinyjs>)
 - Enhance user experience in Shiny apps
- shinythemes (<http://rstudio.github.io/shinythemes/>)
 - Easily alter the appearance of your app
- ggvis (<http://ggvis.rstudio.com/>)
 - Similar to ggplot2 but plots are web-based and more interactive
- shinydashboard (<https://rstudio.github.io/shinydashboard/>)
 - Gives you tools to create “dashboards”

Awesome non-intimidating resources

- Shiny official tutorial <http://shiny.rstudio.com/tutorial>
- Shiny cheatsheet <http://shiny.rstudio.com/images/shiny-cheatsheet.pdf>
- Lots of short useful topics <http://shiny.rstudio.com/articles>
- Shiny in Rmarkdown http://rmarkdown.rstudio.com/authoring_shiny.html
- Get help from <https://groups.google.com/forum/#!forum/shiny-discuss> or <http://stackoverflow.com/questions/tagged/shiny>
- Publish your app free with RStudio <http://www.shinyapps.io>
- Host your app on your own Shiny server
<http://deanattali.com/2015/05/09/setup-rstudio-shiny-server-digital-ocean/>