Building Shiny Apps

UBC STAT 545/547

Dean Attali
http://deanattali.com
Dec 2015

Some material borrowed from Garrett Grolemund/RStudio

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

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

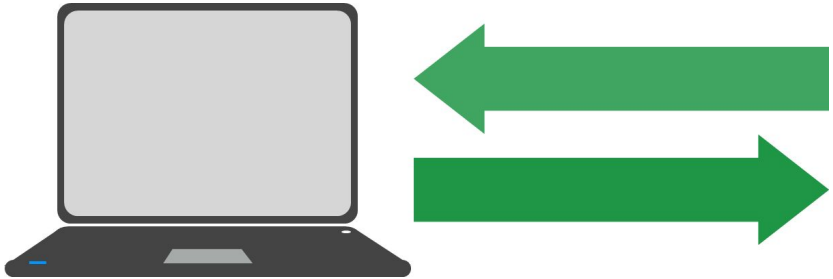
What is Shiny?

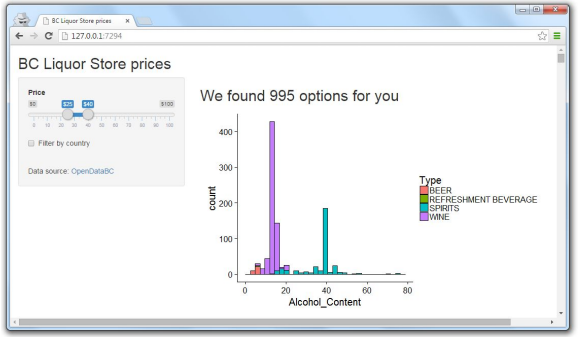
- R package from RStudio
- Web application framework for R
- R code → interactive web page
- No HTML/CSS/JavaScript knowledge required
- Great for sharing R analysis with someone scared of R

Examples

- `install.packages("shiny")`
  `shiny::runExample("01_hello")`
- 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?

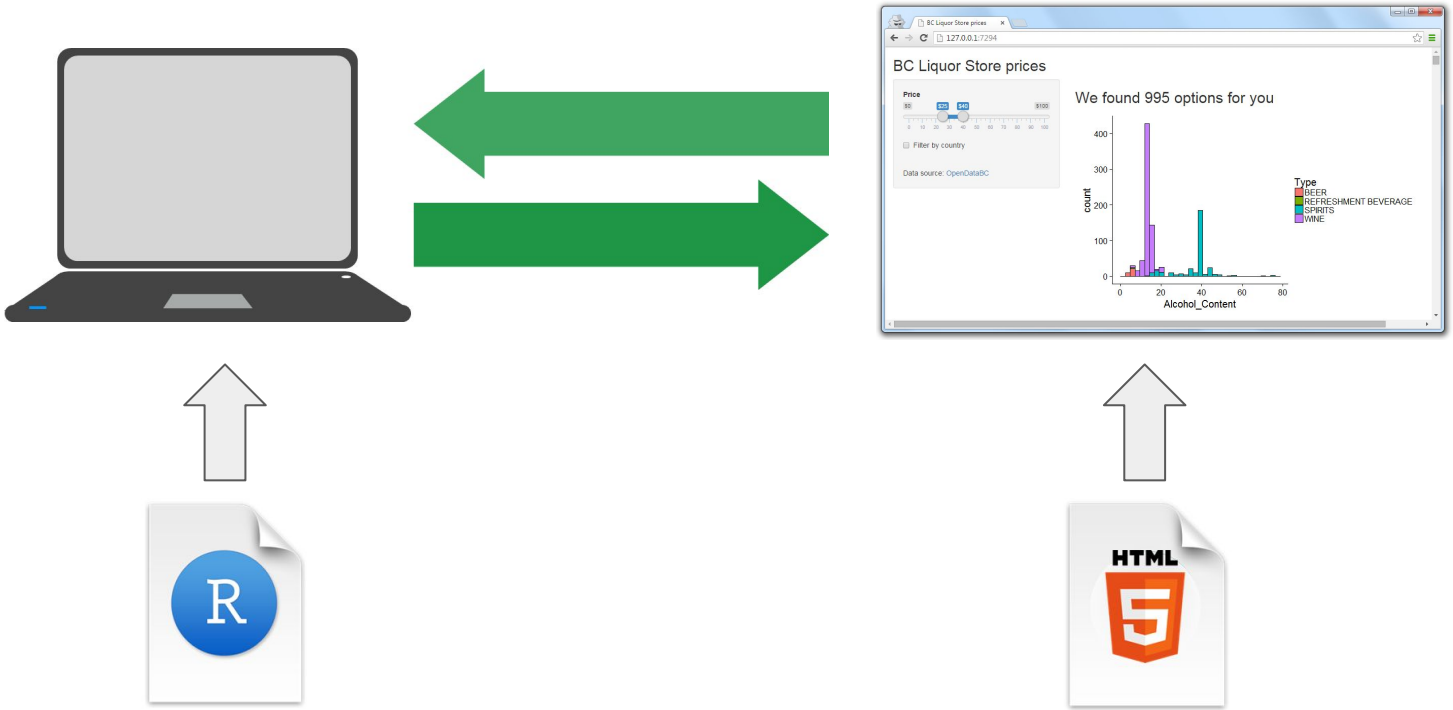




Computer running R

Web page

What is a Shiny app?

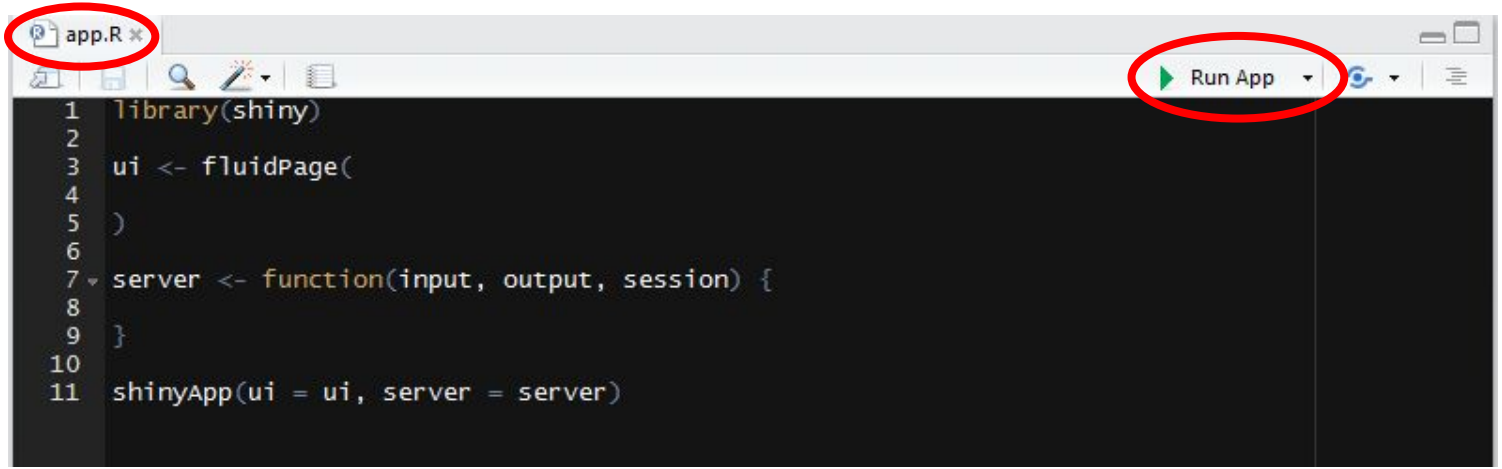


Shiny app template

```
library(shiny)

ui <- fluidPage()

server <- function(input, output) {}

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

Run Shiny app in RStudio, method 1

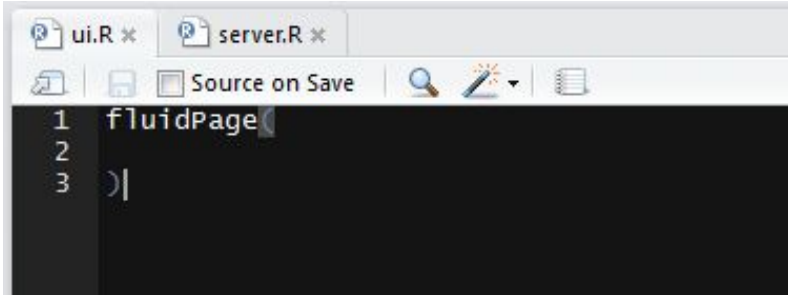
Save file as “**app.R**” → “Run” button turns to “Run App”

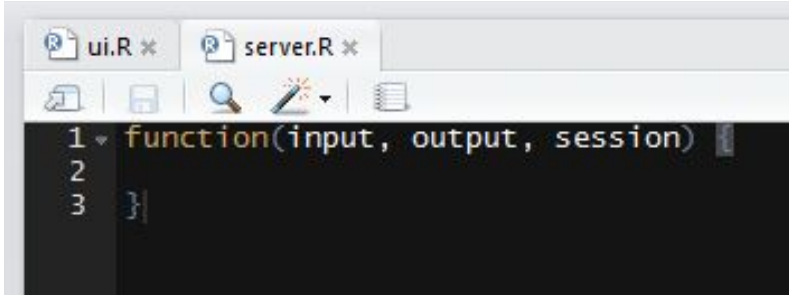


Good for creating Shiny apps quickly, all code in one file

Run Shiny app in RStudio, method 2

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

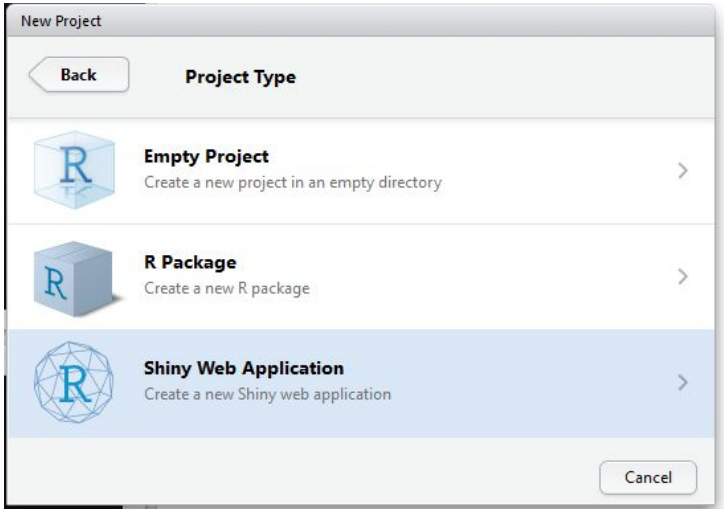




Good for complex Shiny apps, separates view vs logic

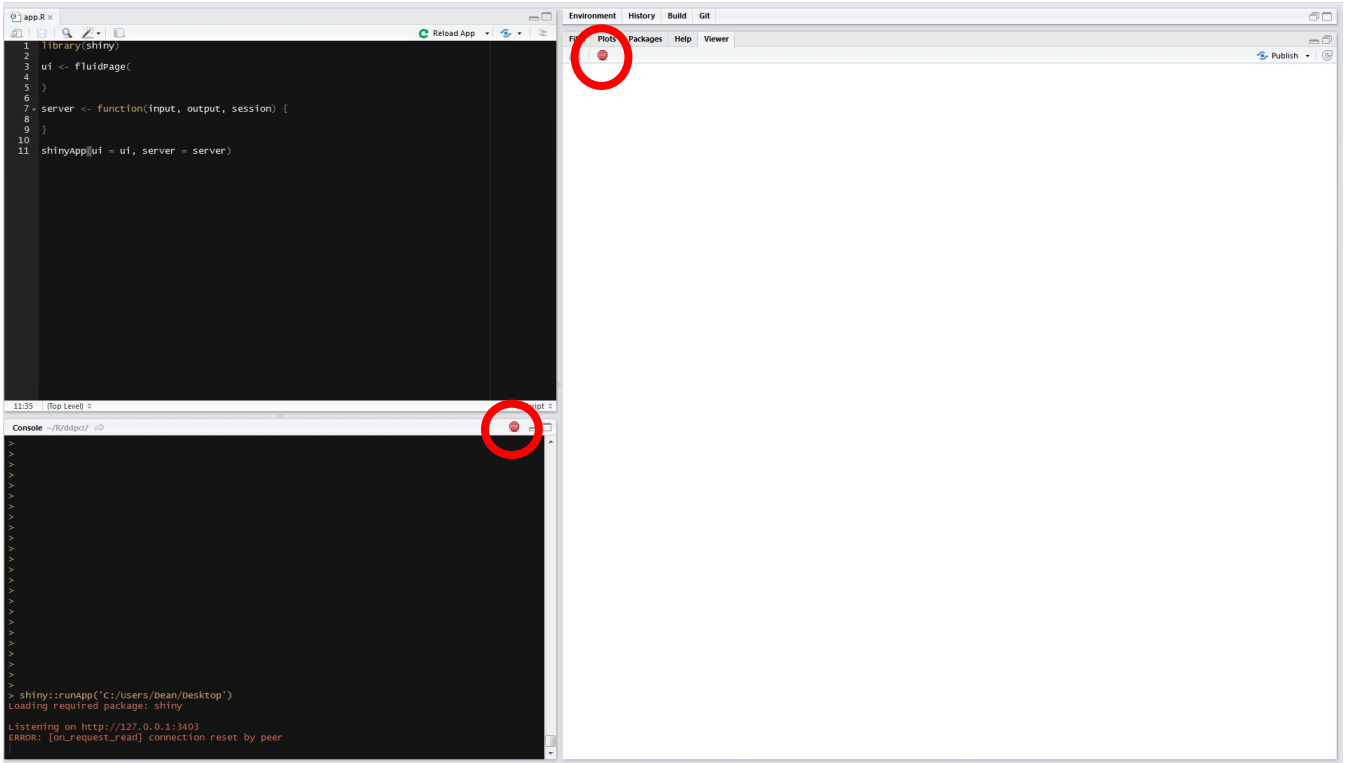
Run Shiny app in RStudio, method 3

File > New Project > New Directory > Shiny Web Application



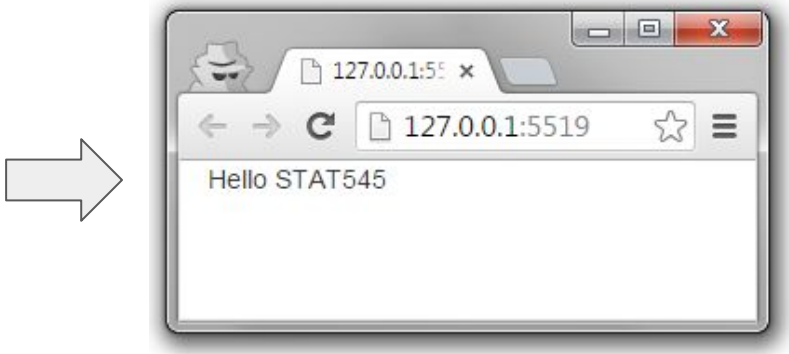
Same as method 2 (ui.R and server.R), but created for you

Stop Shiny app in RStudio



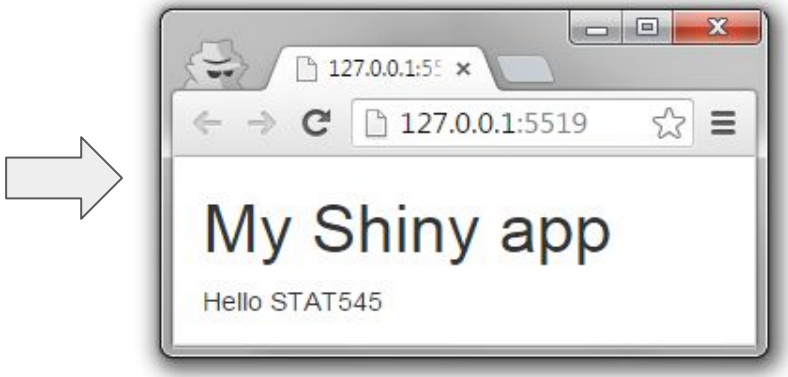
Add elements to app inside fluidPage()

```
library(shiny)

ui <- fluidPage("Hello STAT545")

server <- function(input, output) {}

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



ui

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



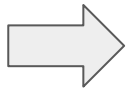
Add HTML to fluidPage()

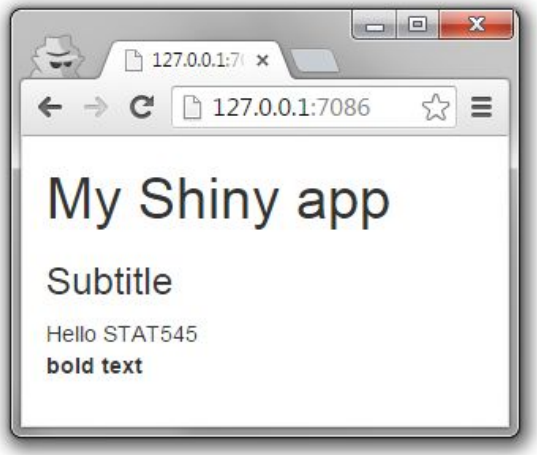
- Remember the UI simply creates HTML
- Can use any HTML tags http://shiny.rstudio.com/articles/tag-glossary.html
- **h1()** = header1, **br()** = line break, **strong()** = bold text
- Any HTML tag can be accessed using `tags` object
  - h1 = tags$h1, br = tags$br
- Common tags can be accessed without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

Add HTML to fluidPage()

ui

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





Use a layout

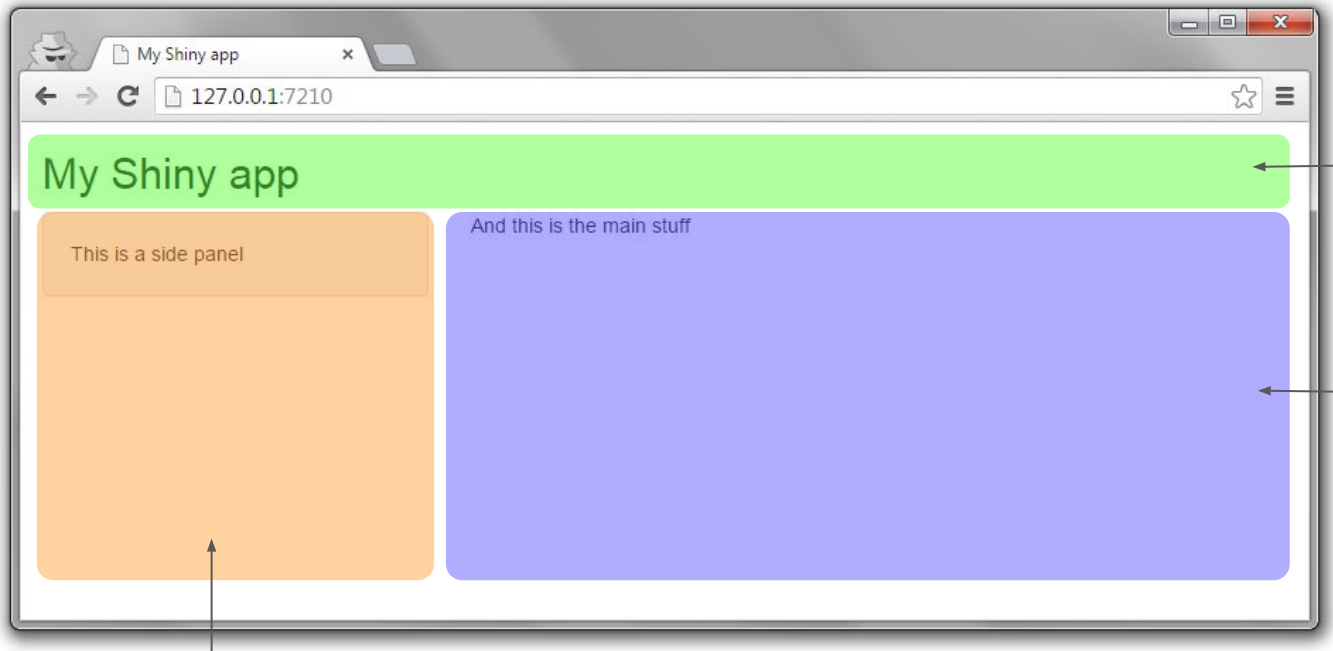
- By default, all elements stack up one after the other
- Can use different layouts http://shiny.rstudio.com/articles/layout-guide.html
- We'll use **sidebarLayout()**

Use a layout - sidebarLayout()

ui

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

Use a layout - sidebarLayout()

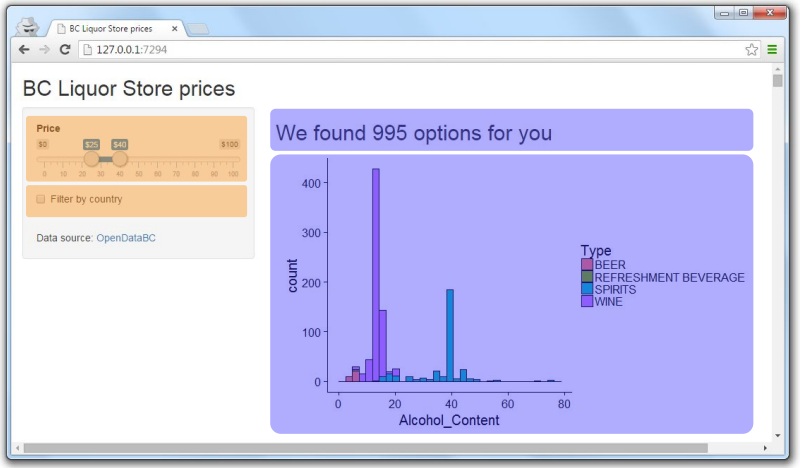


Inputs and outputs

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

ui

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



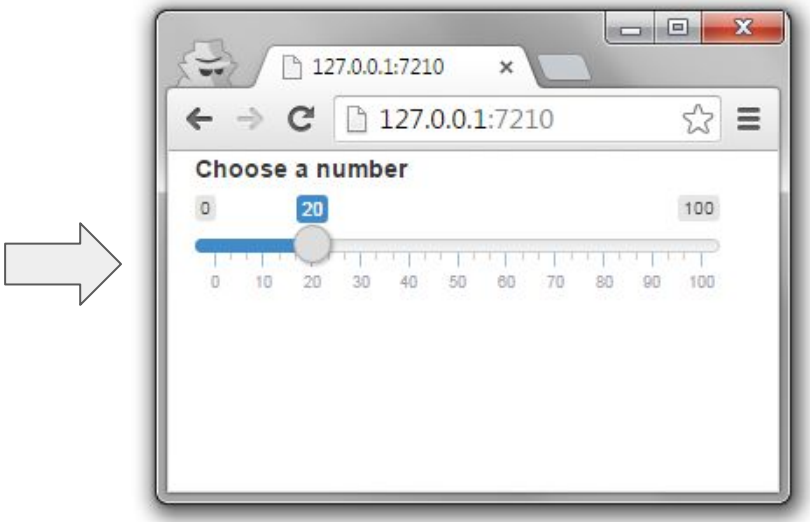
Inputs

```
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

```
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

Buttons

Action

Submit

actionButton()
submitButton()

Single checkbox

Choice A

checkboxInput()

Checkbox group

Choice 1
Choice 2
Choice 3

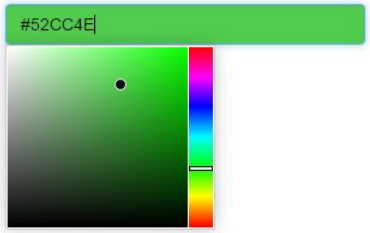
checkboxGroupInput()

Date input

2014-01-01

dateInput()

Colour input



shinyjs::colourInput()

Date range

2014-01-24 to 2014-01-24

dateRangeInput()

File input

Choose File No file chosen

fileInput()

Numeric input

1

numericInput()

Password Input

••••••••••

passwordInput()

Radio buttons

Choice 1
Choice 2
Choice 3

radioButtons()

Select box

Choice 1

selectInput()

Sliders

0 50 100

0 25 75 100

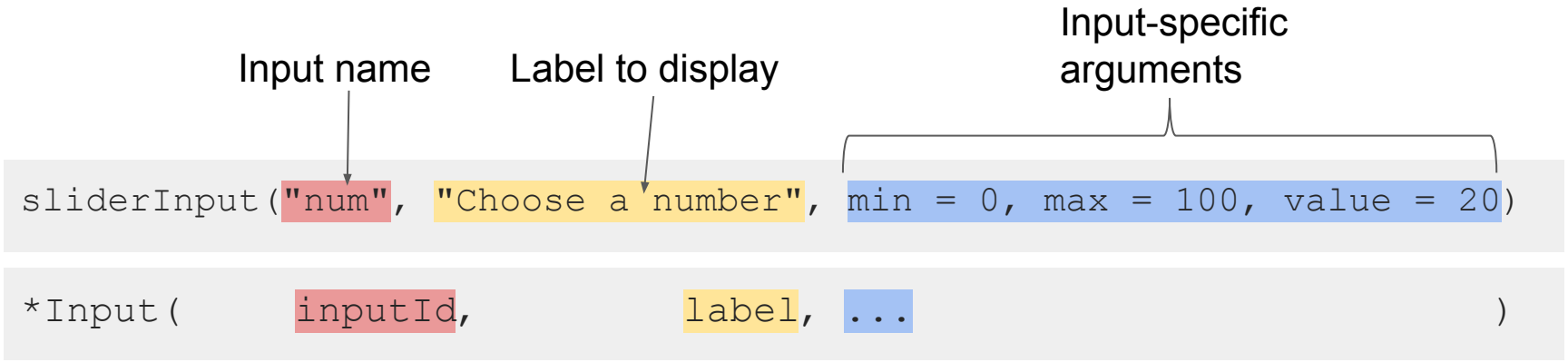
sliderInput()

Text input

Enter text...

textInput()

Inputs



What arguments can I pass to an input function?

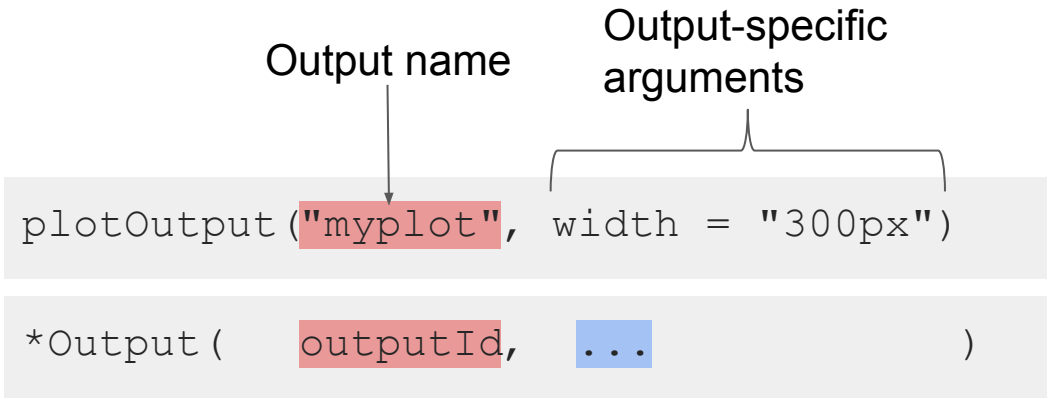
```
?sliderInput
```

Outputs

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

Function	Outputs
plotOutput()	plot
tableOutput()	table
uiOutput()	Shiny UI element
textOutput()	text

Outputs

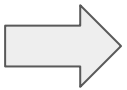


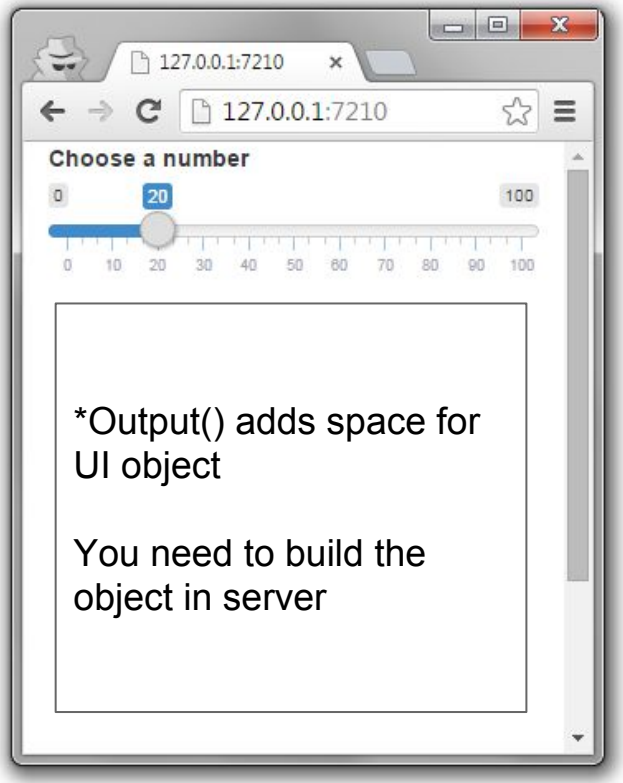
What arguments can I pass to an output function?

```
?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)
```



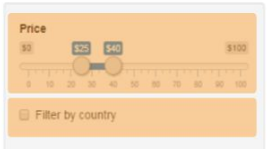


Summary

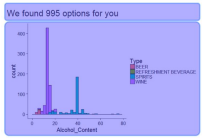
```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

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!

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)) })
```

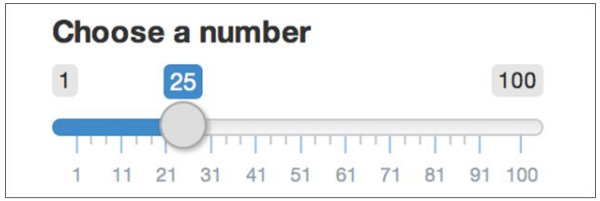
Type of object to build

Code to build the object

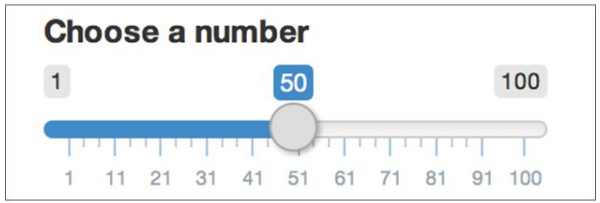
Building outputs
3 - Access input values with input$

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

      # in UI:sliderInput("num", ...)
   })
}
```

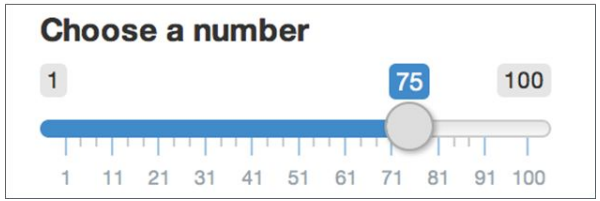
Using input$



`input$num` returns 25



`input$num` returns 50



`input$num` returns 75

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

app.R

```
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)
```

Two files

ui.R

```
library(shiny)

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

server.R

```
library(shiny)

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

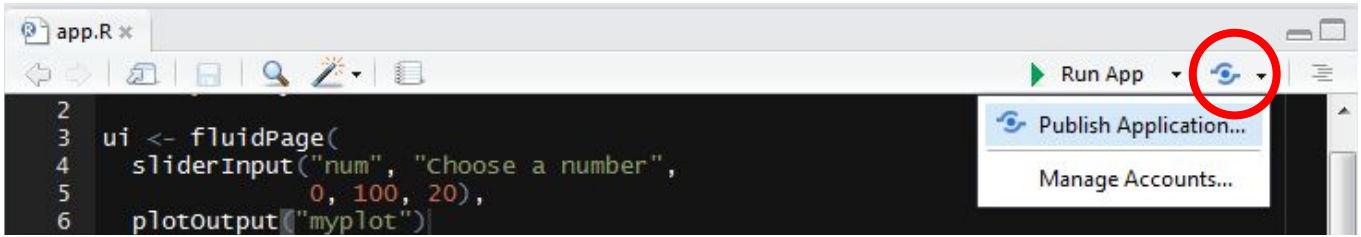
Using buttons in the UI

- 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)
```

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

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

- shinythemes (http://rstudio.github.io/shinythemes/)
  - Easily alter the appearance of your app
- shinyjs (https://github.com/daattali/shinyjs)
  - Enhance user experience in Shiny apps
- leaflet (http://rstudio.github.io/leaflet/)
  - Add interactive maps to your apps
- 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/