How to start with Shiny, Part 1
How to build a Shiny App

Garrett Grolemund
Data Scientist and Master Instructor
May 2015
Email: garrett@rstudio.com
Add elements to your app as arguments to `fluidPage()`

```r
library(shiny)
ui <- fluidPage("Hello World")

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
```
Build your app around Inputs and Outputs
Build your app around **inputs** and **outputs**
Build your app around inputs and outputs
Add elements to your app as arguments to `fluidPage()`

```r
ui <- fluidPage(
  # *Input() functions,
  # *Output() functions
)
```
Inputs
Create an input with an *Input() function.

```r
sliderInput(inputId = "num",
            label = "Choose a number",
            value = 25, min = 1, max = 100)
```

```html
<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="1" data-max="100"
           data-from="25" data-step="1" data-grid="true" data-grid-num="9.9"
           data-grid-snap="false" data-prettify-separator="," data-keyboard="true"
           data-keyboard-step="1.01010101010101"/>
</div>
```
Create an input with an input function.

```r
library(shiny)
ui <- fluidPage(

)

server <- function(input, output) {}

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

server <- function(input, output) {}

shinyApp(server = server, ui = ui)
<table>
<thead>
<tr>
<th>Buttons</th>
<th>Single checkbox</th>
<th>Checkbox group</th>
<th>Date input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ActionButton()</code></td>
<td><code>checkboxInput()</code></td>
<td><code>checkboxGroupInput()</code></td>
<td><code>dateInput()</code></td>
</tr>
<tr>
<td><code>submitButton()</code></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date range</th>
<th>File input</th>
<th>Numeric input</th>
<th>Password Input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dateRangeInput()</code></td>
<td><code>fileInput()</code></td>
<td><code>numericInput()</code></td>
<td><code>passwordInput()</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Radio buttons</th>
<th>Select box</th>
<th>Sliders</th>
<th>Text input</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>radioButtons()</code></td>
<td><code>selectInput()</code></td>
<td><code>sliderInput()</code></td>
<td><code>textInput()</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Syntax

Choose a number

sliderInput(inputId = "num", label = "Choose a number", ...)

- input name (for internal use)
- Notice: Id not ID
- label to display
- input specific arguments

?sliderInput
Outputs
Build your app around **inputs** and **outputs**
<table>
<thead>
<tr>
<th>Function</th>
<th>Inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataTableOutput()</td>
<td>an interactive table</td>
</tr>
<tr>
<td>html1Output()</td>
<td>raw HTML</td>
</tr>
<tr>
<td>imageOutput()</td>
<td>image</td>
</tr>
<tr>
<td>plotOutput()</td>
<td>plot</td>
</tr>
<tr>
<td>tableOutput()</td>
<td>table</td>
</tr>
<tr>
<td>textOutput()</td>
<td>text</td>
</tr>
<tr>
<td>uiOutput()</td>
<td>a Shiny UI element</td>
</tr>
<tr>
<td>verbatimTextOutput()</td>
<td>text</td>
</tr>
</tbody>
</table>
*Output()

To display output, add it to `fluidPage()` with an `*Output()` function.

```
plotOutput("hist")
```

- the type of output to display
- name to give to the output object
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
library(shiny)

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

server <- function(input, output) {} 

shinyApp(ui = ui, server = server)
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)

* Output() adds a space in the ui for an R object.
library(shiny)

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

server <- function(input, output) {}

shinyApp(ui = ui, server = server)
Recap

Begin each app with the template

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

Add elements as arguments to `fluidPage()`

Create reactive inputs with an `*Input()` function

Display reactive results with an `*Output()` function

Assemble outputs from inputs in the server function

Slides at: bit.ly/shiny-quickstart-1
Tell the server how to assemble inputs into outputs
Use 3 rules to write the server function

server <- function(input, output) {

}
Save objects to display to output$

```r
server <- function(input, output) {
  output$hist <- # code
}
```
Save objects to display to output$ 

```
output$hist
plotOutput("hist")
```
Build objects to display with `render*()`

```r
server <- function(input, output) {
  output$hist <- renderPlot({
    }
  }
}
Use the `render*()` function that creates the type of output you wish to make.

<table>
<thead>
<tr>
<th>function</th>
<th>creates</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>renderDataTable()</code></td>
<td>An interactive table (from a data frame, matrix, or other table-like structure)</td>
</tr>
<tr>
<td><code>renderImage()</code></td>
<td>An image (saved as a link to a source file)</td>
</tr>
<tr>
<td><code>renderPlot()</code></td>
<td>A plot</td>
</tr>
<tr>
<td><code>renderPrint()</code></td>
<td>A code block of printed output</td>
</tr>
<tr>
<td><code>renderTable()</code></td>
<td>A table (from a data frame, matrix, or other table-like structure)</td>
</tr>
<tr>
<td><code>renderText()</code></td>
<td>A character string</td>
</tr>
<tr>
<td><code>renderUI()</code></td>
<td>a Shiny UI element</td>
</tr>
</tbody>
</table>
render\(^*()\)
Builds reactive output to display in UI

renderPlot({ hist(rnorm(100)) })

type of object to build

code block that builds the object
Build objects to display with `render*()`

```r
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(100))
  })
}
```
Build objects to display with `render()`

```r
server <- function(input, output) {
  output$hist <- renderPlot({
    title <- "100 random normal values"
    hist(rnorm(100), main = title)
  })
}
```
Access **input** values with `input$`.

```r
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```
Access **input** values with `input$`.

```r
sliderInput(inputId = "num", ...)
```

`input$num`
Input values

The input value changes whenever a user changes the input.

input$num = 25

input$num = 50

input$num = 75
Input values

The input value changes whenever a user changes the input.

Input values
Output will automatically update if you follow the 3 rules
Reactivity 101

Reactivity automatically occurs whenever you use an input value to render an output object

```r
function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$num))
  })
}
```
renderPlot(
  {
    hist(rnorm(input$num))
  }
)
renderPlot({
  hist(rnorm(input$num))
})
Recap: Server

Use the server function to assemble inputs into outputs. Follow 3 rules:

1. Save the output that you build to `output$hist` <-

2. Build the output with a `render*()` function

3. Access input values with `input$num`

Create reactivity by using Inputs to build rendered Outputs
Share your app
Every Shiny app is maintained by a computer running R
Every Shiny app is maintained by a computer running R
How to save your app

One directory with every file the app needs:

- **app.R** (your script which ends with a call to `shinyApp()`)
- datasets, images, css, helper scripts, etc.

You must use this exact name (**app.R**).
Two file apps

```R
# ui.R
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)
```

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

© 2015 RStudio, Inc.
Two file apps

One directory with two files:

- server.R
- ui.R

You must use these exact names
Launch an app
Display options
Close an app
Use shinyapps.io by RStudio®
Shinyapps.io

A server maintained by RStudio

- free
- easy to use
- secure
- scalable
Getting started guide

shiny.rstudio.com/articles/shinyapps.html

Getting started with shinyapps.io

Added: 18 Mar 2014
By: Andy Kripp

Shinyapps.io is a platform as a service (PaaS) for hosting Shiny web apps (applications). This guide will show you how to create a shinyapps.io account and deploy your first application to the cloud.

Before you get started with shinyapps.io, you will need:

- An R development environment, such as the RStudio IDE
- (for Windows users only) Rtools for building packages
- (for Mac users only) Xcode Command Line Tools for building packages
- (for Linux users only) GCC
- The devtools R package (version 1.4 or later)
- The latest version of the shinyapps R package

How to install devtools

Shinyapps.io uses the latest improvements to the devtools package. To use shinyapps.io, you must update devtools to version 1.4 or later. To install devtools from CRAN, run the code below. Then restart your R session.

install.packages('devtools')
RStudio

FREE

$0/month

New to Shiny? Deploy your applications to the cloud for FREE. Perfect for teachers and students or those who want a place to learn and play. No credit card required.

5 Applications
25 Active Hours
Community Support
RStudio Branding

BASIC

$39/month
(or $440/year)

Take your users' experience to the next level. shinyapps.io Basic lets you scale your application performance by adding R processes dynamically as usage increases.

Unlimited Applications
250 Active Hours
Multiple Instances
Email Support

STANDARD

$99/month
(or $1,100/year)

Need password protection? shinyapps.io Standard lets you authenticate your application users.

Unlimited Applications
1000 Active Hours
Authentication
Multiple Instances
Email Support

PROFESSIONAL

$299/month
(or $3,300/year)

shinyapps.io Professional has it all. Share an account with others in your business or change your shinyapps.io domain into a URL of your own.

Unlimited Applications
5000 Active Hours
Authentication
Multiple Users
Multiple Instances
Custom Domains*
Email Support

© CC 2015 RStudio, Inc.

Slides at: bit.ly/shiny-quickstart-1
Build your own server!
A back end program that builds a linux web server specifically designed to host Shiny apps.
Shiny Server Pro
www.rstudio.com/products/shiny/shiny-server/

- **Secure access** - LDAP, GoogleAuth, SSL, and more
- **Performance** - fine tune at app and server level
- **Management** - monitor and control resource use
- **Support** - direct priority support

45 day evaluation free trial
Recap: Sharing

Save your app in its own directory as `app.R`, or `ui.R` and `server.R`

Host apps at `shinyapps.io` by:

1. Sign up for a free `shinyapps.io` account

2. Install the `shinyapps` package

Build your own server with Shiny Server or Shiny Server Pro
Learn more
You now how to

Build an app

Create interactions

Share your apps

```r
library(shiny)

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

shinyApp(ui = ui, server = server)

input$num

renderPlot({
  hist(rnorm(input$num))
})
```

*Input()

*Output()
How to start with Shiny

1. How to build a Shiny app (Today)
2. How to customize reactions (May 27)
3. How to customize appearance (June 3)
The Shiny Development Center

shiny.rstudio.com