Shiny : : CHEAT SHEET

Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)

Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

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

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines `ui` and `server` into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

SHARE YOUR APP

The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at **http://shinyapps.io**
2. Click the **Publish** icon in the RStudio IDE or run: **rsconnect::deployApp**("<path to directory>")

Build or purchase your own Shiny Server
at www.rstudio.com/products/shiny-server/

Building an App

Complete the template by adding arguments to fluidPage() and a body to the server function.

Add inputs to the UI with *Input() functions

Add outputs with *Output() functions

Tell server how to render outputs with R in the server function. To do this:

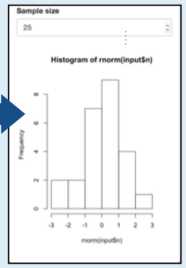
1. Refer to outputs with **output$<id>**
2. Refer to inputs with **input$<id>**
3. Wrap code in a **render*()** function before saving to output

```
library(shiny)

ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)

server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}

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



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
server <- function(input, output) {
  output$hist <- renderPlot({
    hist(rnorm(input$n))
  })
}
shinyApp(ui = ui, server = server)
```

```
# ui.R
fluidPage(
  numericInput(inputId = "n",
    "Sample size", value = 25),
  plotOutput(outputId = "hist")
)
```

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

ui.R contains everything you would save to ui.

server.R ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

The directory name is the name of the app

(optional) defines objects available to both ui.R and server.R

(optional) used in showcase mode

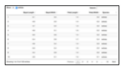
(optional) data, scripts, etc.

(optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "**www**"

Launch apps with `runApp(<path to directory>)`

Outputs - render*() and *Output() functions work together to add R output to the UI

works with



DT::**renderDataTable**(expr, options, callback, escape, env, quoted) — **dataTableOutput**(outputId, icon, …)

renderImage(expr, env, quoted, deleteFile) — **imageOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

renderPlot(expr, width, height, res, …, env, quoted, func) — **plotOutput**(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)

renderPrint(expr, env, quoted, func, width) — **verbatimTextOutput**(outputId)

renderTable(expr,…, env, quoted, func) — **tableOutput**(outputId)

foo

renderText(expr, env, quoted, func) — **textOutput**(outputId, container, inline)

renderUI(expr, env, quoted, func) — & **uiOutput**(outputId, inline, container, …) **htmlOutput**(outputId, inline, container, …)

Inputs

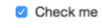
collect values from the user

Access the current value of an input object with **input$<inputId>**. Input values are **reactive**.

actionButton(inputId, label, icon, …)

actionLink(inputId, label, icon, …)

checkboxGroupInput(inputId, label, choices, selected, inline)



checkboxInput(inputId, label, value)

dateInput(inputId, label, value, min, max, format, startview, weekstart, language)

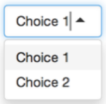
dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)

fileInput(inputId, label, multiple, accept)

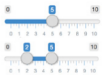
numericInput(inputId, label, value, min, max, step)

passwordInput(inputId, label, value)

radioButtons(inputId, label, choices, selected, inline)



selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())



sliderInput(inputId, label, min, max, value, step, round, format, locale, ticks, animate, width, sep, pre, post)

submitButton(text, icon)
(Prevents reactions across entire app)

textInput(inputId, label, value)