

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 - in three ways:

1. **Host it on shinyapps.io**, a cloud based service from RStudio. To do so:
 - Create a free or professional account at <http://shinyapps.io>
 - Click the Publish icon in RStudio IDE, or run: `rsconnect::deployApp("<path to directory>")`
2. **Purchase RStudio Connect**, a publishing platform for R and Python. www.rstudio.com/products/connect/
3. **Build your own Shiny Server** <https://rstudio.com/products/shiny/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:

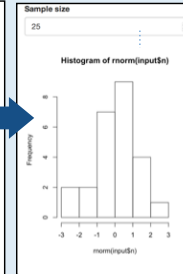
1. Refer to outputs with `output$<id>`
2. Refer to inputs with `input$<id>`
3. Wrap code in a `render*()` function before saving to output

```
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))
  })
}

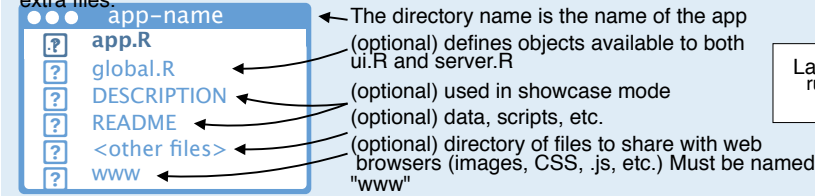
shinyApp(ui = ui, server = server)
```

`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.



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

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



`DT::renderDataTable(expr, options, callback, escape, env, quoted)`

`renderImage(expr, env, quoted, deleteFile)`



`renderPlot(expr, width, height, res, ..., env, quoted, func)`



`renderPrint(expr, env, quoted, func, width)`

`renderTable(expr, ..., env, quoted, func)`

`renderText(expr, env, quoted, func)`

`renderUI(expr, env, quoted, func)`



`dataTableOutput(outputId, icon, ...)`

`imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)`

`plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)`

`verbatimTextOutput(outputId)`

`tableOutput(outputId)`

`textOutput(outputId, container, inline)`

`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.

`Action` `actionButton(inputId, label, icon, ...)`

`Link` `actionLink(inputId, label, icon, ...)`

☒ Choice 1 `checkboxGroupInput(inputId, label, choices, selected, inline)`

☒ Choice 2 `checkboxGroupInput(inputId, label, choices, selected, inline)`

☐ Choice 3 `checkboxGroupInput(inputId, label, choices, selected, inline)`

☒ Check me `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)`

`Choose File` `fileInput(inputId, label, multiple, accept)`

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

`passwordInput(inputId, label, value)`

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

☒ Choice A `radioButtons(inputId, label, choices, selected, inline)`

☐ Choice B `radioButtons(inputId, label, choices, selected, inline)`

☐ Choice C `radioButtons(inputId, label, choices, selected, inline)`

`Choice 1` `selectInput(inputId, label, choices, selected, multiple, selectize, width, size) (also selectizeInput())`

`Choice 2` `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)`

`Enter text` `textInput(inputId, label, value)`

