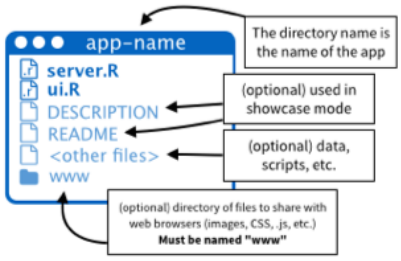
Shiny Cheat Sheet

learn more at shiny.rstudio.com

Shiny 0.10.0 Updated: 6/14

RStudio

1. Structure Each app is a directory that contains a server.R file and usually a ui.R file (plus optional extra files)



render* functions

function	expects	creates
renderDataTable	any table-like object	DataTables.js table
renderImage	list of image attributes	HTML image
renderPlot	plot	plot
renderPrint	any printed output	text
renderTable	any table-like object	plain table
renderText	character string	text
renderUI	Shiny tag object or	UI element (HTML)

input values are reactive.
They must be surrounded with one of:

render* - creates a shiny UI component
reactive - creates a reactive expression
observe - creates a reactive observer
isolate - creates a non-reactive copy of a reactive object

2. server.R A set of instructions that build the R components of your app. To write server.R:

A. Provide server.R with the minimum necessary code, **shinyServer(function(input, output) {})**

B. Define the R components for your app between the braces that follow **function(input, output)**

C. Save each R component in your UI as **output$<component name>**

D. Create each output component with a render* function.

E. Give each render* function the R code the server needs to build the component. The server will note any reactive values that appear in the code and will rebuild the component whenever these values change.

F. Refer to widget values with **input$<widget name>**

server.R

```
# load libraries, scripts, data

shinyServer(function(input, output) {

  # make user specific variables

  output$text <- renderText({
    input$title
  })

  output$plot <- renderPlot({
    x <- mtcars[ , input$x]
    y <- mtcars[ , input$y]
    plot(x, y, pch = 16)
  })

})
```

3. Execution Place code where it will be run the minimum necessary number of times

Run once - code placed *outside of* ***shinyServer*** will be run once, when you first launch your app. Use this code to set up the tools that your server will only need one copy of.

Run once per user - code placed *inside* ***shinyServer*** will be run once each time a user visits your app (or refreshes his or her browser). Use this code to set up the tools that your server will need a unique copy of for each user.

Run often - code placed within a render*, reactive, or observe function will be run many times. Place here only the code that the server needs to rebuild a UI component after a widget changes.

4. Reactivity When an input changes, the server will rebuild each output that depends on it (even if the dependence is indirect). You can control this behavior by shaping the chain of dependence.

render* - An output will automatically update whenever an input in its render* function changes.


```
output$z <- renderText({
  input$a
})
```

Reactive expression - use reactive to create objects that will be used in multiple outputs.


```
x <- reactive({
  input$a
})

output$y <- renderText({
  x()
})
output$z <- renderText({
  x()
})
```

isolate - use use isolate to use an input without depending on it. Shiny will not rebuild the output when the isolated input changes.


```
output$z <- renderText({
  paste(
    isolate(input$a),
    input$b
  )
})
```

observe - use observe to create code that runs when an input changes, but does not create an output object.


```
observe({
  input$a
  # code to run
})
```

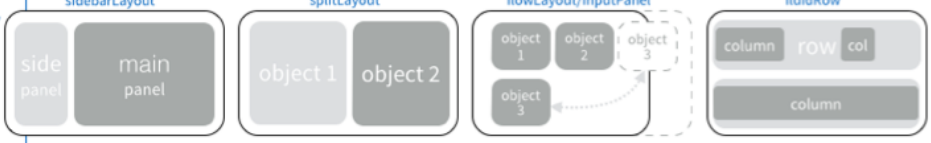

ui.R

```
shinyUI(fluidPage(

  titlePanel("mtcars data"),
  sidebarLayout(
    sidebarPanel(

      textInput("title", "Plot title:",
        value = "x v y"),

      selectInput("x", "Choose an x var:",
        choices = names(mtcars),
        selected = "disp"),

      selectInput("y", "Choose a y var:",
        choices = names(mtcars),
        selected = "mpg")
    ),

    mainPanel(
      h3(textOutput("text")),
      plotOutput("plot")
    )
  )
))
```

5. ui.R A description of your app's User Interface (UI), the web page that displays your app. To write ui.R:

A. Include the minimum necessary code for ui.R, **shinyUI(fluidPage())**
* note: use `navbarPage` instead of `fluidPage` if you'd like your app to have multiple pages connected by a navbar

B. Build a layout for your UI. **sidebarLayout** provides a default layout when used with **sidebarPanel** and **mainPanel**. **splitLayout**, **flowLayout**, and **inputLayout** divide the page into equally spaced regions. **fluidRow** and **column** work together to create a grid-based layout, which you can use to layout a page or a panel.




```
shinyUI(fluidPage(
  sidebarLayout(
    sidebarPanel(…),
    mainPanel(…)
  )
))
```

```
shinyUI(fluidPage(
  splitLayout(
    numericInput(…),
    selectInput(…)
  )
))
```

```
shinyUI(fluidPage(
  flowLayout(
    numericInput(…),
    selectInput(…),
    sliderInput(…)
  )
))
```

```
shinyUI(fluidPage(
  fluidRow(
    column(width = 4, …),
    column(width = 2,
      offset = 3, …),
  ),
  fluidRow(
    column(width = 12, …)
  )
))
```

C. In each panel or column, place…

R components - These are the output objects that you defined in **server.R**. To place a component:

1. Select the ***Output** function that builds the type of object you want to place in the UI.
2. Pass the ***Output** function a character string that corresponds to the name you assigned the object in **server.R**, e.g.

output$plot <- renderPlot({ … }) ⟷ plotOutput("plot")

*Output functions

dataTableOutput	tableOutput
htmlOutput	textOutput
imageOutput	uiOutput
plotOutput	verbatimTextOutput

Widgets - The first argument of each widget function is the **<name>** for the widget. You can access a widget's current value in **server.R** with **input$<name>**

widget	function	common arguments
Action button	actionButton	inputId, label
checkbox	checkboxInput	inputId, label, value
checkbox group	checkboxGroupInput	inputId, label, choices, selected
date selector	dateInput	inputId, label, value, min, max, format
date range selector	dateRangeInput	inputId, label, start, end, min, max, format
file uploader	fileInput	inputId, label, multiple
Number field	numericInput	inputId, label, value, min, max, step
Radio buttons	radioButtons	inputId, label, choices, selected
select box	selectInput	inputId, label, choices, selected, multiple
slider	sliderInput	inputId, label, min, max, value, step
submit button	submitButton	text
text field	textInput	inputId, label, value

HTML elements - Add html elements with shiny functions that parallel common HTML tags.

a	tagsCol	tagsForm	tagsInput	tagsOutput	tagsSub
tagsAbbr	tagsColgroup	h1	tagsIns	p	tagsSummary
tagsAddress	tagsCommand	h2	tagsKbd	tagsParam	tagsSup
tagsArea	tagsData	h3	tagsKeygen	pre	tagsTable
tagsArticle	tagsDatalist	h4	tagsLabel	tagsProgress	tagsTbody
tagsAside	tagsDd	h5	tagsLegend	tagsQ	tagsTd
tagsAudio	tagsDel	h6	tagsLi	tagsRuby	tagsTextarea
tagsB	tagsDetails	tagsHead	tagsLink	tagsRp	tagsTfoot
tagsBase	tagsDfn	tagsHeader	tagsMark	tagsRt	tagsTh
tagsBdi	div	tagsHgroup	tagsMap	tagsRs	tagsThead
tagsBdo	tagsDl	hr	tagsMenu	tagsSamp	tagsTime
tagsBlockquote	tagsDt	HTML	tagsMeta	tagsScript	tagsTitle
tagsBody	em	tagsI	tagsMeter	tagsSection	tagsTr
br	tagsEmbed	tagsIframe	tagsNav	tagsSelect	tagsTrack
tagsButton	tagsEventsource	img	tagsNoscript	tagsSmall	tagsU
tagsCanvas	tagsFieldset	includeCSS	tagsObject	tagsSource	tagsUl
tagsCaption	tagsFigcaption	includeMarkdo	tagsOl	span	tagsVar
tagsCite	tagsFigure	wn	tagsOptgroup	strong	tagsVideo
code	tagsFooter	includeScript	tagsOption	tagsStyle	tagsWbr

6. Run your app

runApp - run from local files
runGitHub - run from files hosted on www.GitHub.com
runGist - run from files saved as a gist (gist.github.com)
runURL - run from files saved at any URL

RStudio

7. Share your app Launch your app as a live web page that users can visit online.

ShinyApps.io

Host your apps on RStudio's server. Free and paid options
www.shinyapps.io

Shiny Server

Build your own linux server to host apps. Free and open source.
shiny.rstudio.com/deploy

Shiny Server Pro

Build a commercial server with authentication, resource management, and more.
shiny.rstudio.com/deploy