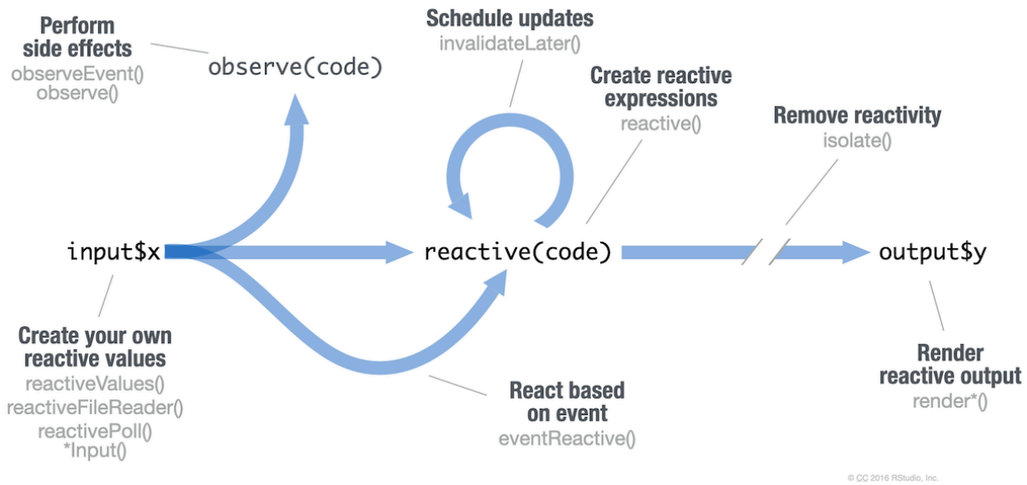
Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context.**



CREATE YOUR OWN REACTIVE VALUES

```
# *Input() example
ui <- fluidPage(
 textInput("a","","A")
)
```

```
#reactiveValues example
server <- function(input,output){
 rv <- reactiveValues()
 rv$number <- 5
}
```

***Input() functions**
(see front page)

Each input function creates a reactive value stored as **input$<inputId>**.

reactiveValues(...)

Creates a list of reactive values whose values you can set.

CREATE REACTIVE EXPRESSIONS

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A"),
 textInput("z","","Z"),
 textOutput("b"))

server <- function(input,output){
 re <- reactive({
  paste(input$a,input$z)})
 output$b <- renderText({
  re()
 })
}
shinyApp(ui, server)
```

reactive(x, env, quoted, label, domain**)**

Reactive expressions:

- **cache** their value to reduce computation
- can be called elsewhere
- notify dependencies when invalidated

Call the expression with function syntax, e.g. re().

REACT BASED ON EVENT

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go"),
 textOutput("b")
)

server <- function(input,output){
 re <- eventReactive(
  input$go,{input$a})
 output$b <- renderText({
  re()
 })
}
```

eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, ..., label, domain, ignoreNULL, ignoreInit**)**

Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

RENDER REACTIVE OUTPUT

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <- function(input,output){
 output$b <-
  renderText({
   input$a
  })
}

shinyApp(ui, server)
```

render*() functions
(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.

Save the results to **output$<outputId>**.

PERFORM SIDE EFFECTS

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go")
)

server <- function(input,output){
 observeEvent(input$go,{
  print(input$a)
 })
}

shinyApp(ui, server)
```

observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, ..., label, suspended, priority, domain, autoDestroy, ignoreNULL, ignoreInit, once**)**

Runs code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

REMOVE REACTIVITY

```
library(shiny)

ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b")
)

server <- function(input,output){
 output$b <-
  renderText({
   isolate({input$a})
  })
}

shinyApp(ui, server)
```

isolate(expr**)**

Runs a code block. Returns a **non-reactive** copy of the results.

UI - An app's UI is an HTML document.

Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
  textInput("a","")
)
## <div class="container-fluid">
##  <div class="form-group shiny-input-container">
##    <label for="a"></label>
##    <input id="a" type="text"
##      class="form-control" value=""/>
##  </div>
## </div>
```

Returns HTML

Add static HTML elements with **tags**, a list of functions that parallel common HTML tags, e.g. **tags$a()**. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

Run **names(tags)** for a complete list.
tags$h1("Header") -> <h1>Header</h1>

The most common tags have wrapper functions. You do not need to prefix their names with **tags$**

```
ui <- fluidPage(
 h1("Header 1"),
 hr(),
 br(),
 p(strong("bold")),
 p(em("italic")),
 p(code("code")),
 a(href="", "link"),
 HTML("<p>Raw html</p>")
)
```

To include a CSS file, use **includeCSS()**, or
1. Place the file in the **www** subdirectory
2. Link to it with:

```
tags$head(tags$link(rel = "stylesheet",
  type = "text/css", href = "<file name>"))
```

To include JavaScript, use **includeScript()** or
1. Place the file in the **www** subdirectory
2. Link to it with:

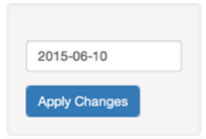
```
tags$head(tags$script(src = "<file name>"))
```

To include an image:
1. Place the file in the **www** subdirectory
2. Link to it with img(src="<file name>")

Layouts

Combine multiple elements into a "single element" that has its own properties with a panel function, e.g.

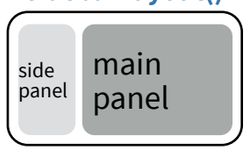
```
wellPanel(
  dateInput("a", ""),
  submitButton()
)
```



absolutePanel()
conditionalPanel()
fixedPanel()
headerPanel()
inputPanel()
mainPanel()
navlistPanel()
sidebarPanel()
tabPanel()
tabsetPanel()
titlePanel()
wellPanel()

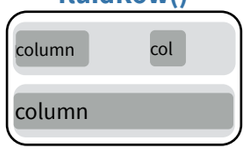
Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

sidebarLayout()




```
ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(),
    mainPanel()
  )
)
```

fluidRow()




```
ui <- fluidPage(
  fluidRow(column(width = 4),
    column(width = 2,  offset = 3)),
  fluidRow(column(width = 12))
)
```

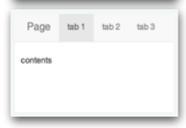
Also **flowLayout()**, **splitLayout()**, **verticalLayout()**, **fixedPage()**, **and fixedRow()**.

Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage( tabsetPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```


```
ui <- fluidPage( navlistPanel(
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents")))
```

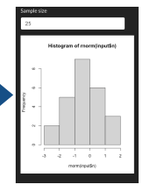

```
ui <- navbarPage(title = "Page",
    tabPanel("tab 1", "contents"),
    tabPanel("tab 2", "contents"),
    tabPanel("tab 3", "contents"))
```



Themes

Use the **bslib** package to add existing themes to your Shiny app ui, or make your own.

```
library(bslib)
ui <- fluidPage(
  theme = bs_theme(
    bootswatch = "darkly",
    ...
  )
)
```

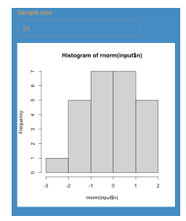


bootswatch_themes() Get a list of themes.

Build your own theme by customizing individual arguments.

bs_theme(bg = "#558AC5", fg = "#F9B02D", ...**)**

?bs_theme for a full list of arguments.



bs_themer() Place within the server function to use the interactive theming widget.