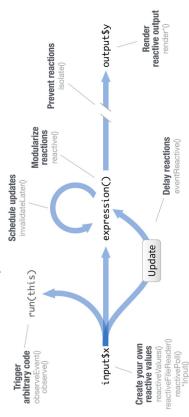
Reactivitv

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() functions (see front page) reactiveValues()	Each input function creates a reactive value stored as input\$ <inputid> reactiveValues() creates a list of reactive values whose values you can set.</inputid>
<pre># example snippets ui <- fluidPage(textInput("a","","A"))</pre>	<pre>server <- function(input,output){ rv <- reactive/alues() } rv\$number <- 5</pre>

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

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

Builds an object to

render*() functions

RENDER REACTIVE OUTPUT

(see front page)

, "A")

ui <- fluidPage(
 textInput("a",""
 textOutput("b")</pre> library(shiny)

PREVENT REACTIONS

library(shiny)	180
ui <- fluidPage(textInput("a",","A"), textOutput("b"))	Rur Ret Cop
<pre>server <- function(input,output) { outputsb < renderText({ isolate({input\$a}) }) }</pre>	•
,	

MODULARIZE REACTIONS

shinyApp(ui, server)

read	lak	Create	· cach	com	• can	• notii	whe	
ul <= rtuldrage(textInput("a","","A"),	textInput("z","","Z"),	Server <-	runction(input,output){ re <- reactive({	paste(input\$a,input\$z)})	re()	({ }	shinyApp(ui, server)	

ctive(x, env, quoted es a **reactive expressio** pel, domain)

• caches its value to reduce computation	• can be called by other code	 notifies its dependencies when it ha been invalidated 	Call the expression with function syntax, e.g. re()
--	-------------------------------	---	---

turns a **non-reactive** py of the results. ns a code block.

olate(expr)

observeEvent(eventExpr

TRIGGER ARBITRARY CODE

output\$<outputId>

shinyApp(ui, server)

Save the results to

in the code changes.

Library(shiny)

		מא
	ui <- fluidPage(+ox+Tanu+("a" "" na")	٧a
nc	actionButton("go", "Go"),	eV
	textOutput("b"))	Va
	-> 10/10/	p
	function(input,output){	2
	re <- eventReactive(ב ל
	output\$b <- renderText({	אָל ע
_	re()	Znc
_	({ 1	in
		val
	shinyApp(ui, server)	, h

argument when reactive values in 1st argument change. See **observe()** for suspended, priority, domain, , handlerExpr, event.env, event.quoted, handler.env, handler.quoted, labe, autoDestroy, ignoreNULL) Runs code in 2nd alternative. server <function(input,output){
 observeEvent(input\$go,{
 print(input\$a)
 }
}</pre> ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go")</pre> shinyApp(ui, server)

DELAY REACTIONS

library(shiny)	eventR
ui <- fluidPage(value
actionButton("go","Go"),	event
textOutput("b")	value.
1	doma
function(input,output){	Creates
input\$go,{input\$a})	PXDres
output\$b <- renderText({	2nd ars
3)	invalid
~,	values
shinvApp(ui, server)	Choncho

eactive (eventExpr, Expr, event.env,

.quoted, value.env,

ates when reactive in, ignoreNUĹL) sion with code in gument that only in 1st argument s reactive

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

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





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

tags\$ a	tags\$ data	tags\$ h6	tags\$ nav	tags\$ span
tags\$ abbr	tags\$ datalist	tags\$ head	tags\$noscript	
tags\$ address	tags\$ dd	tags\$header	tags\$ object	tags\$ style
tags\$ area	tags\$ del	tags\$hgroup	tags\$ ol	tags\$ sub
tags\$article	tags\$ details		tags\$optgroup	
tags\$ aside	tags\$ dfn	M	tags\$ option	tags\$ sup
tags\$audio	tags\$ div		tags\$output	tags\$ table
tags\$ b	tags\$ dl	tags\$ iframe	tags\$ p	tags\$ tbody
tags\$ base	tags\$ dt	tags\$ img	tags\$ param	tags\$ td
tags\$ bdi	tags\$ em	tags\$input	tags\$ pre	tags\$ textarea
tags\$ bdo	tags\$ embed	tags\$ins	tags\$progress	tags\$ tfoot
tags\$blockquote			tags\$ q	tags\$ th
tags\$body		tags\$keygen	tags\$ ruby	tags\$ thead
tags\$ br	tags\$figcaption	tags\$ label	tags\$ rp	tags\$ time
tags\$ button		tags\$ legend	tags\$ rt	tags\$ title
tags\$canvas	Ļ.	tags\$li	tags\$ s	tags\$ tr
tags\$caption	tags\$ form	tags\$link	tags\$ samp	tags\$track
tags\$ cite	tags\$ h1	tags\$ mark	tags\$ script	tags\$ u
tags\$code	tags\$ h2	tags\$ map	tags\$ section	tags\$ ul
tags\$ col	tags\$ h3	tags\$ menu	tags\$ select	tags\$ var
tags\$colgroup	tags\$ h4	tags\$ meta	tags\$ small	tags\$video
tags\$command	tags\$ h5	tags\$ meter	tags\$ source	tags\$ wbr
F				

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



To include a CSS file, use includeCSS(), or 1. Place the file in the www subdirectory

tags\$head(tags\$link(rel = "stylesheet",
 type = "text/css", href = "<file name>")) 2. Link to it with

To include JavaScript, use includeScript() or 1. Place the file in the www subdirectory 2. Link to it with 4

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

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

M

Lavouts

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



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

fluidRow(column(width = 4),
 column(width = 2, offset = 3)),
fluidRow(column(width = 12)) Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions. ui <- fluidPage fluidRow()

flowLayout() object object 1

column

flowLayout(#object 1, #object 2, #object 3 ui <- fluidPage(

sidebarLayout()

object 3

sidebarLayout(sidebarPanel(),

ui <- fluidPage(

mainPanel()

panel main side panel

splitLayout()

ui <- fluidPage(

object object

splitLayout(# object 1,
object 2

verticalLayout() object 1

object 2

ui <- fluidPage(

Layer tabPanels on top of each other, and navigate between them, with: object 3

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

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

tabPanel("tab 2", "contents"), tabPanel("tab 3", "contents"))

tab 2 Page tab 1

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio• info@rstudio.com • 844 448-1212 • rstudio.com • Learn more at shiny.rstudio.com • shiny 0.12.0 • Updated: 2016-01