



INTRO TO SHINY

COMPUTING IN OPTIMIZATION AND STATISTICS 2020

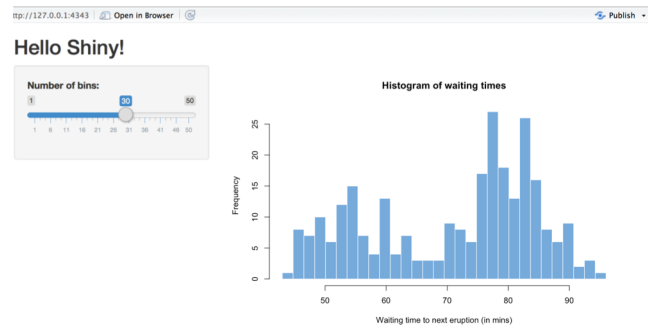
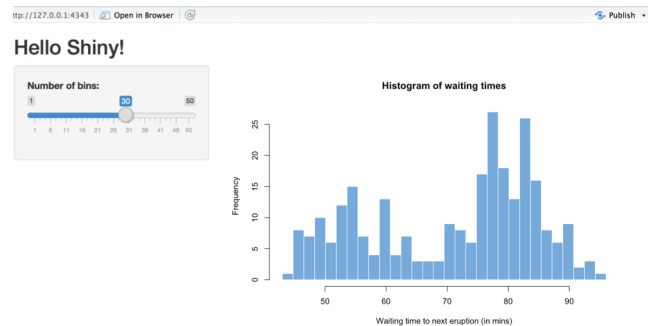
Ted Papalexopoulos & Agni Orfanoudaki
Operations Research Center
January 2020

WHY USE SHINY?

- Want to build a Graphical User Interface (GUI) but only know R?
Use Shiny!
- Interactively explore data:
 - Change model parameters
 - Filter datasets
 - Add extra “dimensions” to your visualizations
- Probably the easiest way to impress stakeholders (but don't impress them too much)
 - Users can interact with your data and analysis
- You can host standalone apps on a webpage or build dashboards within your code.



- Every Shiny app is maintained by a computer running R



SHINY BASIC STRUCTURE

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

- **UI:** defines webpage layout, buttons, plots, etc.
- **Server:** Processes inputs into outputs (i.e., everything else)
- **shinyApp:** Creates the app bringing the two elements together

LET'S START WITH THE SIMPLEST TEMPLATE

```
library(shiny)

ui <- fluidPage()

server <- function(input, output){}

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

- **UI:** defines webpage layout, buttons, plots, etc.
- **Server:** Processes inputs into outputs (i.e., everything else)
- **shinyApp:** Creates the app bringing the two elements together

THE UI ENVIRONMENT


A UI contains **Layouts**, **Inputs** and **Outputs**

```
ui <- fluidPage(  
  verticalLayout(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
    plotOutput("distPlot")  
  )  
)
```



Layouts define how objects are placed on the webpage

```
ui <- fluidPage(  
  verticalLayout(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
    plotOutput("distPlot")  
  )  
)
```



Inputs define controls for the user (e.g., sliderInput, dateInput, fileInput)

```
ui <- fluidPage(  
  verticalLayout(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
    plotOutput("distPlot")  
  )  
)
```


Outputs define things to display

```
ui <- fluidPage(  
  verticalLayout(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
    plotOutput("distPlot")  
  )  
)
```



Inputs and Outputs have **IDs** that the server uses to access their values

```
ui <- fluidPage(  
  verticalLayout(  
    sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30),  
    plotOutput("distPlot")  
  )  
)
```

The **Server** is a function. It takes a list of **inputs**, processes them using **reactives**, and assigns the results to a list of **outputs**

```
server <- function(input, output) {  
  
  x    <- faithful[, 2]  
  
  bins <- reactive({  
  
    seq(min(x), max(x), length.out = input$bins + 1)  
  
  })  
  
  output$distPlot <- renderPlot({  
  
    hist(x, breaks = bins())  
  
  })  
  
}
```

Inputs and outputs from the UI are accessible by their **ID**.

```
server <- function(input, output) {  
  
  x    <- faithful[, 2]  
  
  bins <- reactive({  
  
    seq(min(x), max(x), length.out = input$bins + 1)  
  
  })  
  
  output$distPlot <- renderPlot({  
  
    hist(x, breaks = bins())  
  
  })  
  
}
```

Reactives are functions in the server that are executed whenever their inputs change (more on this later). Objects that depend on the input **must be wrapped in a reactive**.

```
server <- function(input, output) {  
  
  x    <- faithful[, 2]  
  
  bins <- reactive({  
  
    seq(min(x), max(x), length.out = input$bins + 1)  
  
  })  
  
  output$distPlot <- renderPlot({  
  
    hist(x, breaks = bins())  
  
  })  
  
}
```

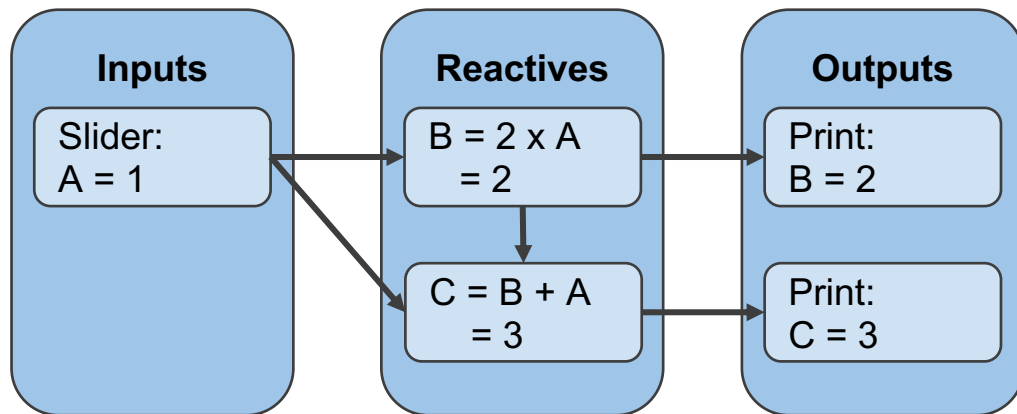
Objects that don't depend on the input don't have to be inside reactives.

```
server <- function(input, output) {  
  x <- faithful[, 2]  
  
  bins <- reactive({  
    seq(min(x), max(x), length.out = input$bins + 1)  
  })  
  
  output$distPlot <- renderPlot({  
    hist(x, breaks = bins())  
  })  
  
}
```

Reactives are typically defined using `reactive({ ... })`. Reactives that generate outputs are special and correspond to the type of output: `renderPlot` corresponds to `plotOutput` in the UI, etc.

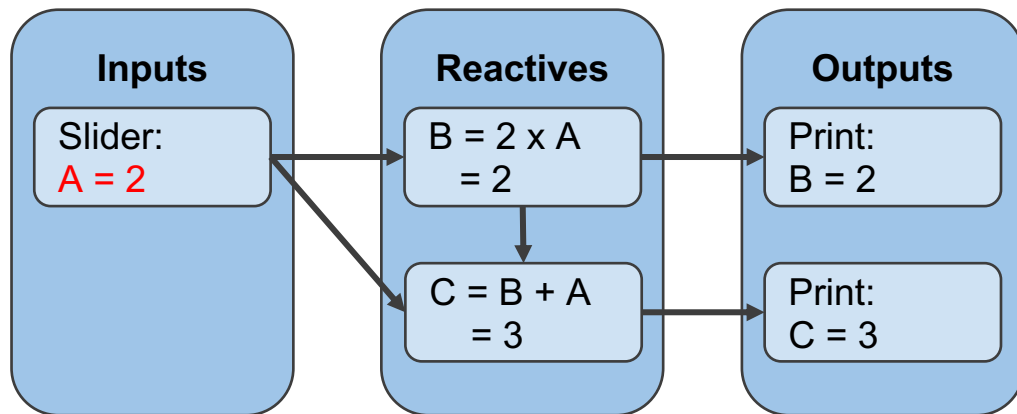
```
server <- function(input, output) {  
  
  x    <- faithful[, 2]  
  
  bins <- reactive({  
  
    seq(min(x), max(x), length.out = input$bins + 1)  
  
  })  
  
  output$distPlot <- renderPlot({  
  
    hist(x, breaks = bins())  
  
  })  
  
}
```

- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



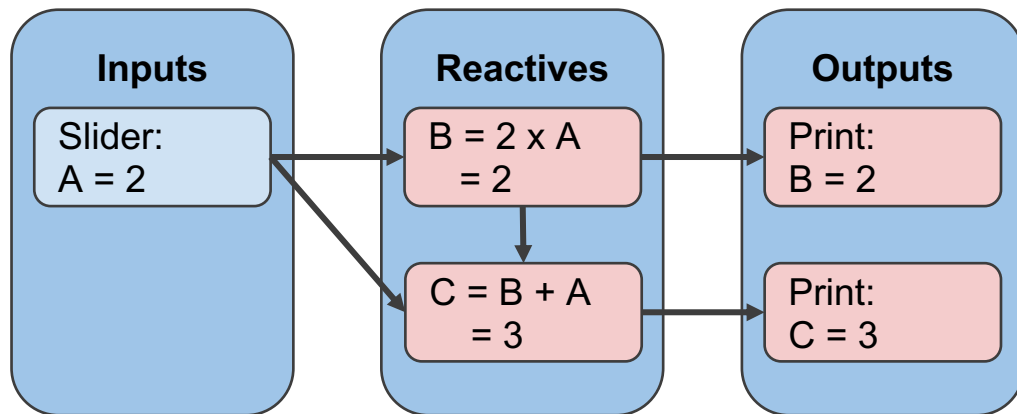
- **Inputs, Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

1. Change
input: A -> 2



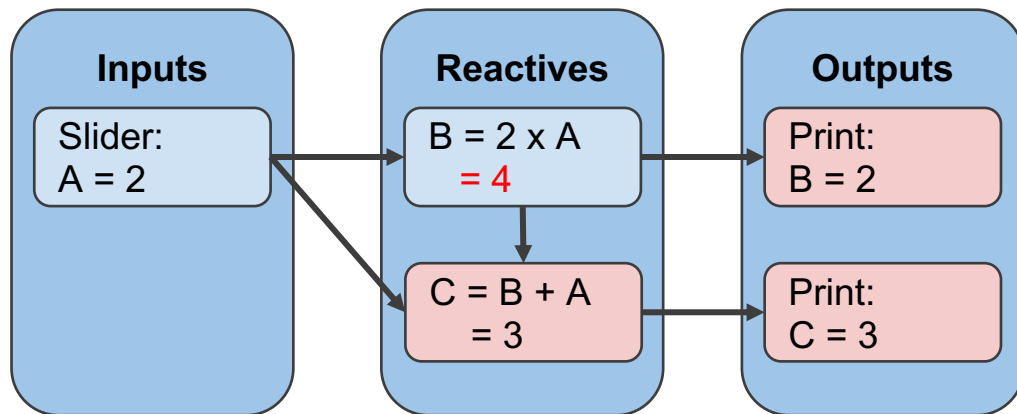
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

2. Mark all downstream nodes as “dirty”



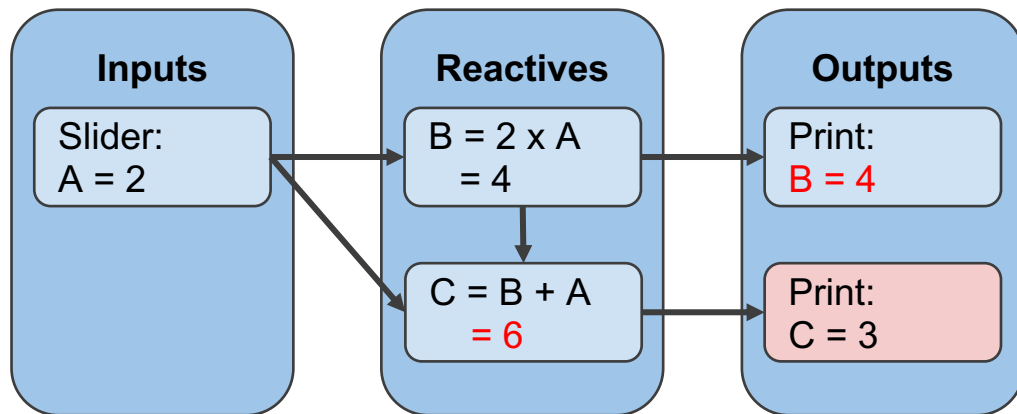
- **Inputs, Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

3. Update nodes whose
parents are clean



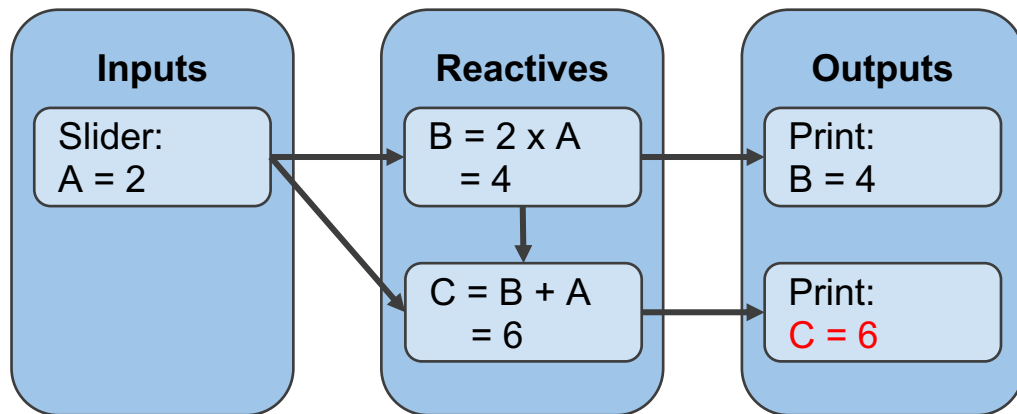
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

3. Update nodes whose
parents are clean

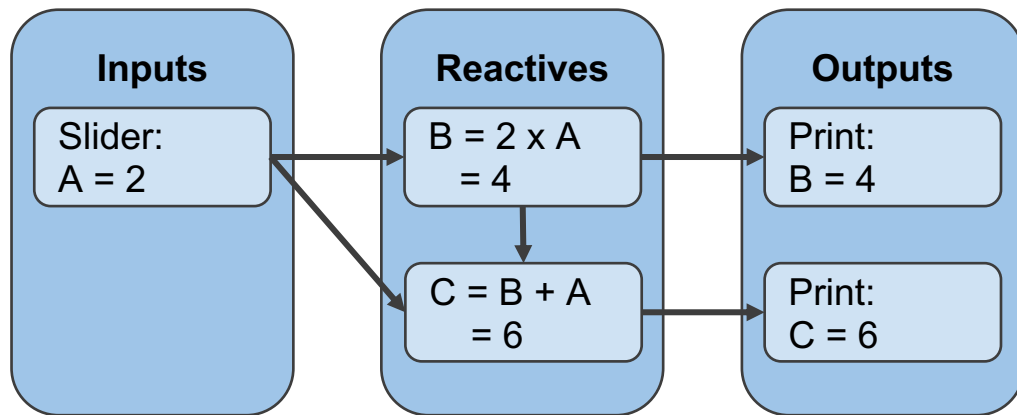


- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

3. Update nodes whose
parents are clean



- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



4. Done! Graph is clean;
wait for input

EXERCISE I

Add an input that allows the user to filter on neighborhood

Tips

- a. use `listings $ neighborhood_cleansed` to see what neighborhoods are in the dataset
- b. `selectInput(...)` will be useful
- c. Extra credit: only display neighborhoods with listings that satisfy all the other filters.
<https://shiny.rstudio.com/articles/dynamic-ui.html>

EXERCISE II

Add an interactive table that displays information about the listings.

Tip: You'll want to use `renderDataTable(...)` and `dataTableOutput(...)`

USEFUL SOURCES

- Full tutorial on Shiny: <https://shiny.rstudio.com/tutorial/>
- Detailed Gallery with templates and examples: <https://shiny.rstudio.com/gallery/>
- Many wrappers for fancy JS viz libraries: <http://gallery.htmlwidgets.org>
- Bootstrap themes: <https://rstudio.github.io/shinythemes/>
- You can use `browser()` to debug