Intro to Shiny

What is Shiny?

- Want to build a Web app but only know R? Use Shiny!
- Interactively explore data:
  - Change model parameters
  - Dashboarding
  - Filter datasets
  - Add extra “dimensions” to your visualizations
- Probably the easiest way to impress stakeholders (but don’t impress them too much)
- Deploy easily to web via ShinyApps.io
- Some examples:
  - Revenue management game: https://atzheng.shinyapps.io/retailer-game/
  - COVID-19 epidemic modeling: https://alhill.shinyapps.io/COVID19seir/

Anatomy of Shiny App

- **UI**: defines webpage layout, buttons, plots, etc.
- **Server**: Processes inputs into outputs (i.e., everything else)
- Minimal app:

```
ui <- fluidPage()

server <- function(input, output) { }

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

Anatomy of a Shiny App: **UI**

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

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

Anatomy of a Shiny App: **UI**

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")
    )
)
```

Anatomy of a Shiny App: **UI**

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")
    )
)
```

Anatomy of a Shiny App: **UI**

Outputs define things to display

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

Anatomy of a Shiny App: **UI**

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")
    )
)
```

Anatomy of a Shiny App: **Server**

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())

    })

}
```

Anatomy of a Shiny App: **Server**

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())

    })

}
```

Anatomy of a Shiny App: **Server**

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())

    })

}
```

Anatomy of a Shiny App: **Server**

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())

    })

}
```

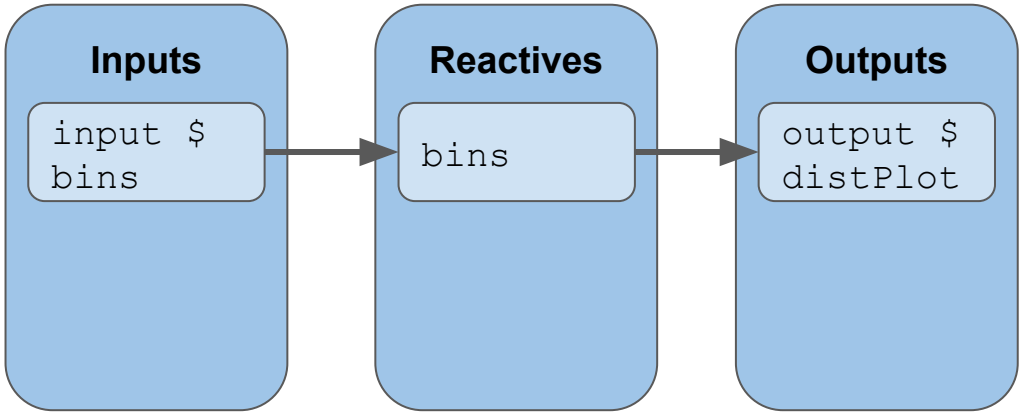
Anatomy of a Shiny App: **Server**

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())

    })

}
```

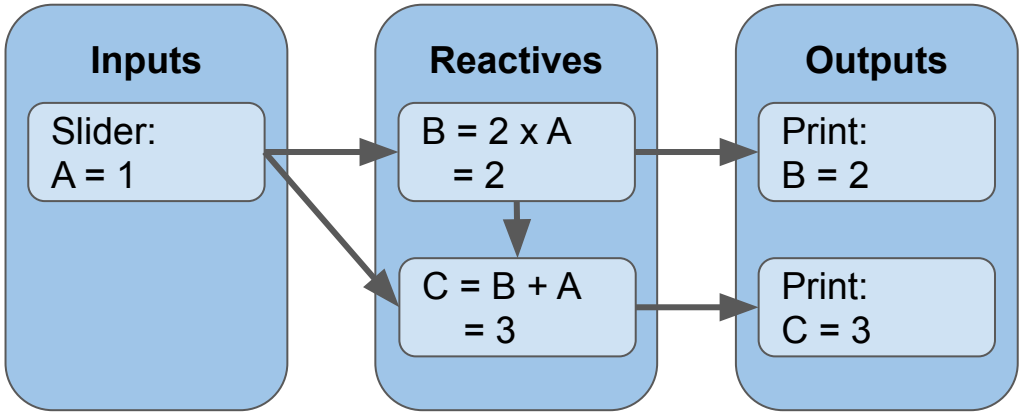
Reactivity

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



Reactivity

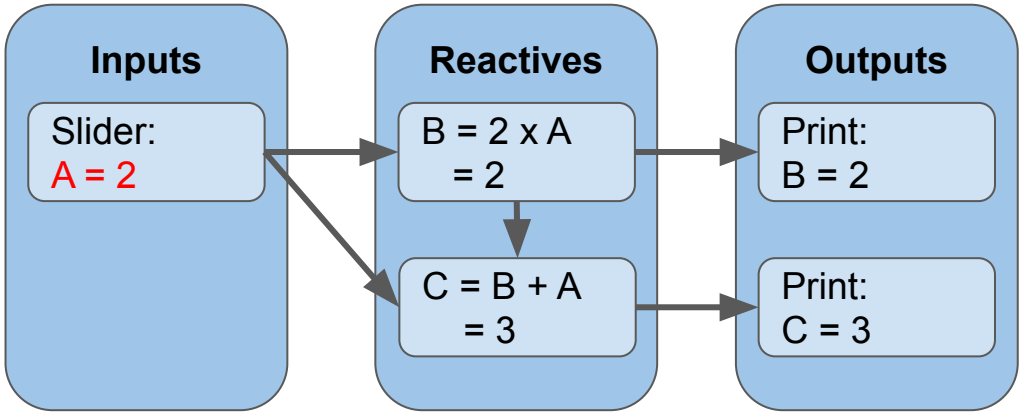
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)



Reactivity

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

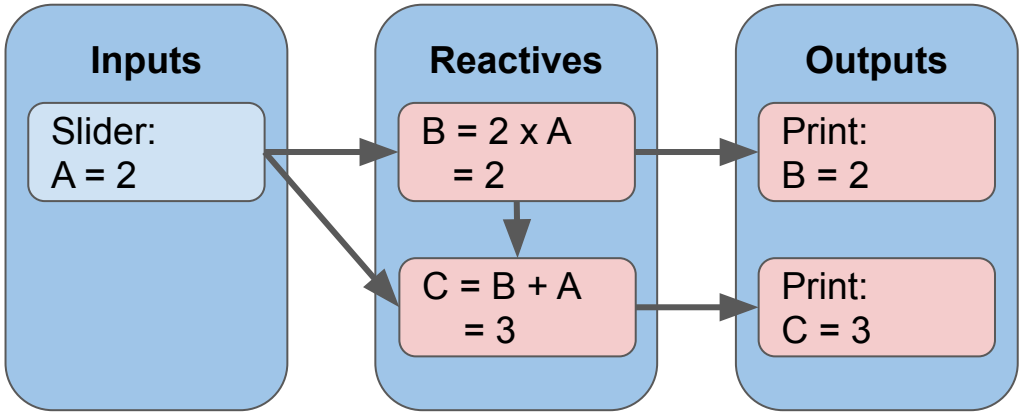
1. Change input: A -> 2



Reactivity

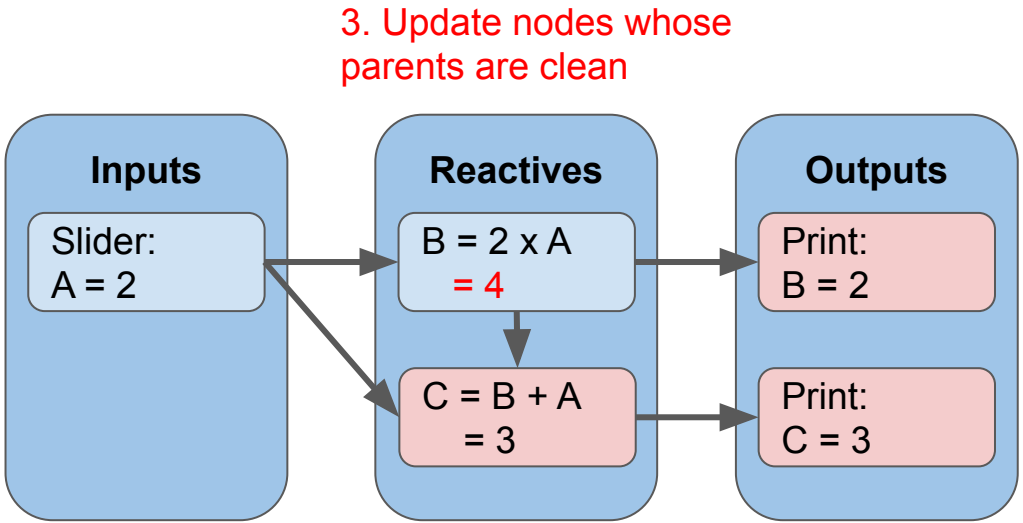
- **Inputs**, **Outputs** and **Reactives** are nodes in a (directed, acyclic) graph
- Changing something upstream updates values downstream (like Excel!)

2. Mark all downstream nodes as “dirty”



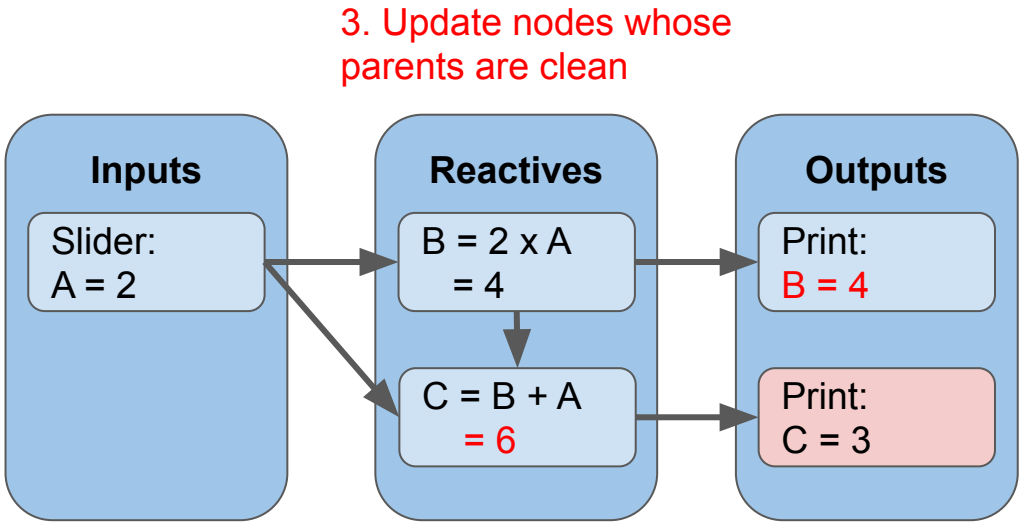
Reactivity

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



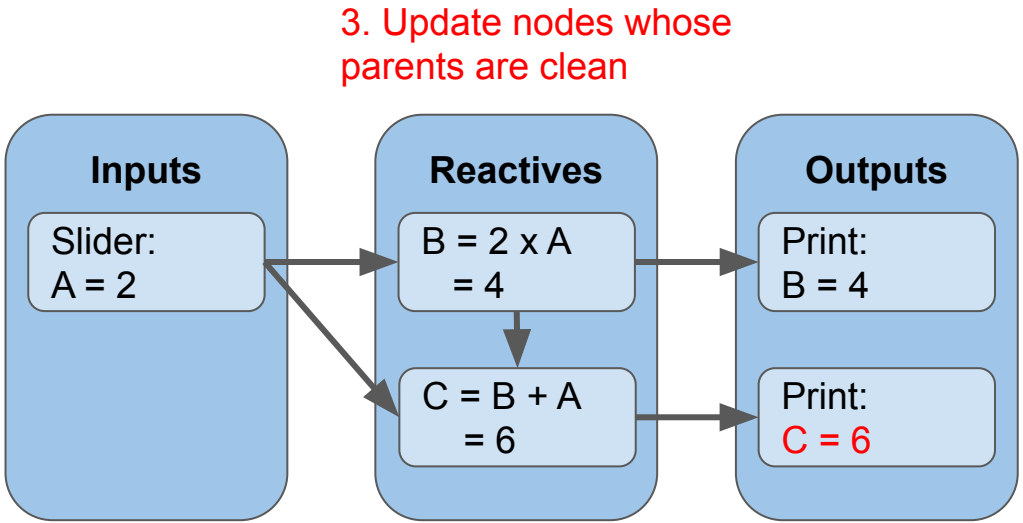
Reactivity

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



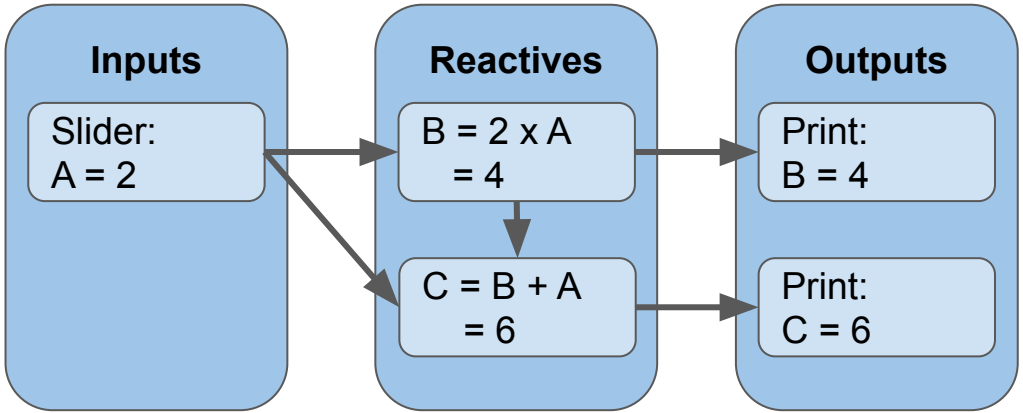
Reactivity

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



Reactivity

- **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

Demo: Building Airbnb’s UI

Exercise 1

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 2

Add an interactive table that displays information about the listings.

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

Final Tips / Tricks

- Many wrappers for fancy JS viz libraries: http://gallery.htmlwidgets.org
- Bootstrap themes: https://rstudio.github.io/shinythemes/
- You can use `browser()` to debug