Workshop Series:
Reusable Research Data Made Shiny

Ontario Dairy Research Centre | Online
February 21st - 24th, 2023

University of Guelph
Improve Life.

Compute Ontario

ADC

Let’s do a quick recap with Kahoot!

Your turn!

Given this UI:

```
ui <- fluidPage(
  textInput("name", "What's your name?"),
  textOutput("greeting")
)
```

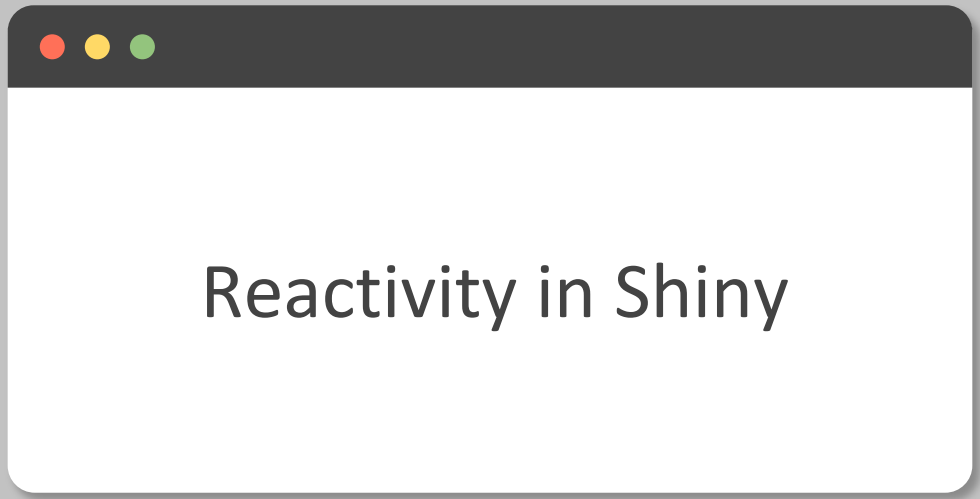
** Fix and run them to make sure they work properly

What is wrong in each of these server functions?

```
server1 <- function(input, output, server) {
  input$greeting <- renderText(paste0("Hello ", name))
}
```

```
server2 <- function(input, output, server) {
  greeting <- paste0("Hello ", input$name)
  output$greeting <- renderText(greeting)
}
```

```
server3 <- function(input, output, server) {
  output$greeting <- paste0("Hello", input$name)
}
```


Reactivity in Shiny

Reactivity in Shiny

Reactivity means that outputs automatically update as inputs change.

This is the big idea in Shiny: you don't need to tell an output when to update, because Shiny automatically figures it out for you.

Let's take a closer look on the greetings example. Notice what happens when you type letter by letter on the textInput.

Reactivity in Shiny

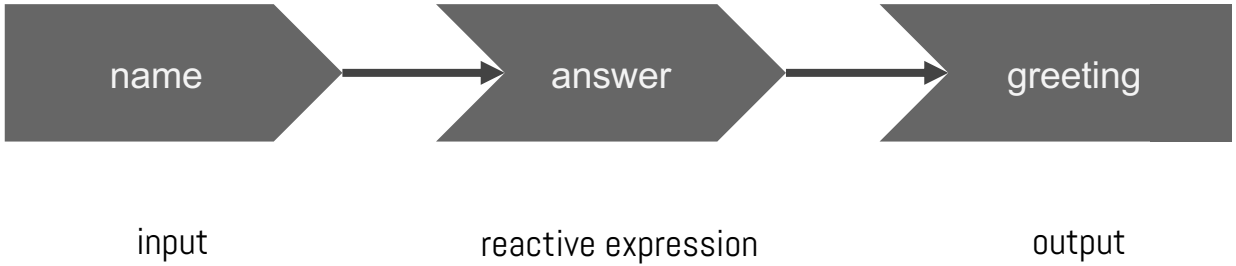
Shiny is lazy!

- A Shiny App will only ever do the **minimal amount of work needed** to update the output controls that you can currently see
- Saves you resources by running just what you need, BUT…. let's go back to the greeting example and see what happens if we introduce a typo to the outputId "greeting"

Reactivity in Shiny

Shiny is lazy!

- A Shiny App will only ever do the **minimal amount of work needed** to update the output controls that you can currently see
- Saves you resources by running just what you need, BUT…. let's go back to the greeting example and see what happens if we introduce a typo to the outputId "greeting"



Reactivity in Shiny

Shiny has it's own execution order

- The order you write your reactive/render functions **do not matter**

```
server <- function(input, output, session) {
  answer <- reactive(paste0("Hello ", input$name, "!"))
  output$greeting <- renderText(answer())
}
```

```
server <- function(input, output, session) {
  output$greeting <- renderText(answer())
  answer <- reactive(paste0("Hello ", input$name, "!"))
}
```

Both orders work just fine!

Reactivity in Shiny

But I want to control my app, what now?!

You can use a variant of the reactive expression: eventReactive()

Add an action button to your UI:

```
actionButton("show", "Show Answer")
```

And slightly modify your reactive expression answer() to:

```
answer <- eventReactive(input$show, paste0("Hello ", input$name, "!"))
```

Reactivity in Shiny

Sometimes you need to run a code that is not meant to be rendered, nor to become a reactive expression, but you need to access input variables.

Observer is your solution!

Add this line anywhere inside the server function.

```
observe(print(paste("The value of input$name now is:", input$name)))
```

> Look at the console and see what happens when you start your app, and when you start typing.

Reactivity in Shiny

Just like reactive() has an eventReactive() to control the execution order, observe() has its counterpart:

observeEvent()

Can you guess how we would print that message on the console only after we click on the show button?

Reactivity in Shiny

What is the main difference between an **observer** and a **reactive expression**?

Reactivity in Shiny

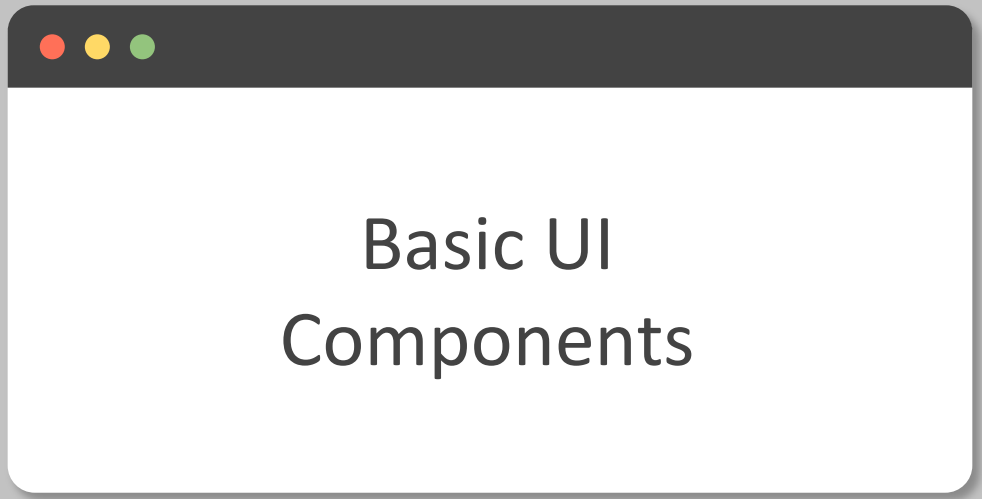
What is the main difference between an **observer** and a **reactive expression**?

observer: does not assign variables

```
observe(print("this text came from an observer"))
```

reactive expression: assigns variables

```
message <- reactive("this text came from a reactive expression")
observe(print(message()))
```


Basic UI
Components

Basic UI Components

Let's take a look at some basic UI components

Coffee Break!