Different Approaches to Shiny App Development

By Andrés F. Quintero

Who am I?

- Software Engineer from Colombia
- Specialise in building Shiny Apps
- Currently work building Shiny Applications for use in healthcare institutions
- Free/Open Source advocate and enthusiast

andresquinterom

andyquinterom

What is this presentation about?

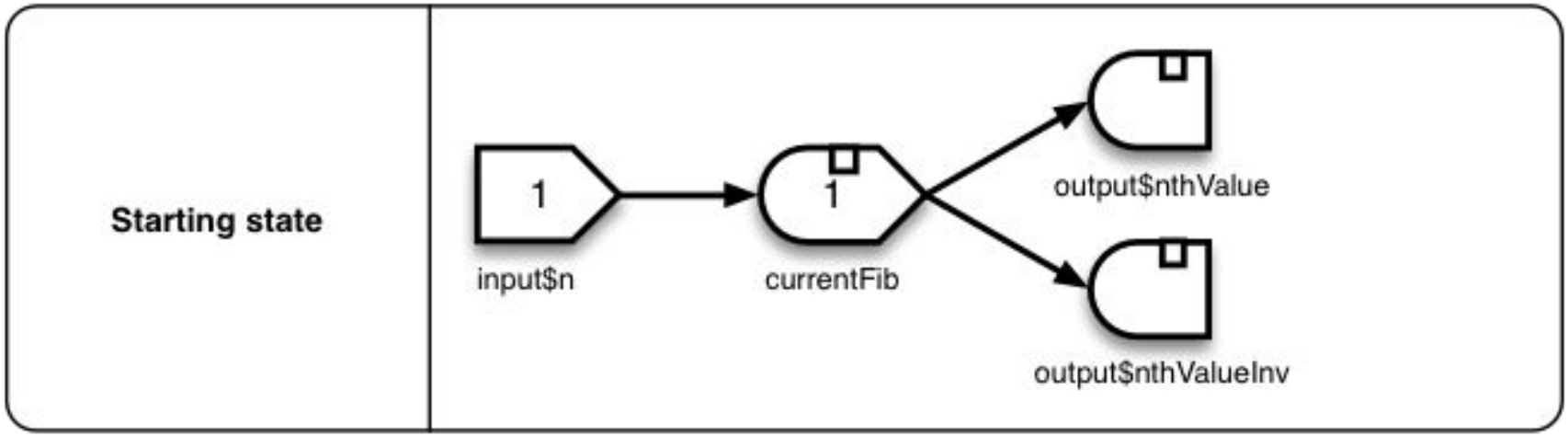
- The different approaches to server-side logic inside Shiny Applications.
- Geared toward Shiny developers looking to learn about how different teams or people could write Shiny Apps.
- Three different methods I have personally used extensively.

What do I mean by approach?

- Consistency and restrictions

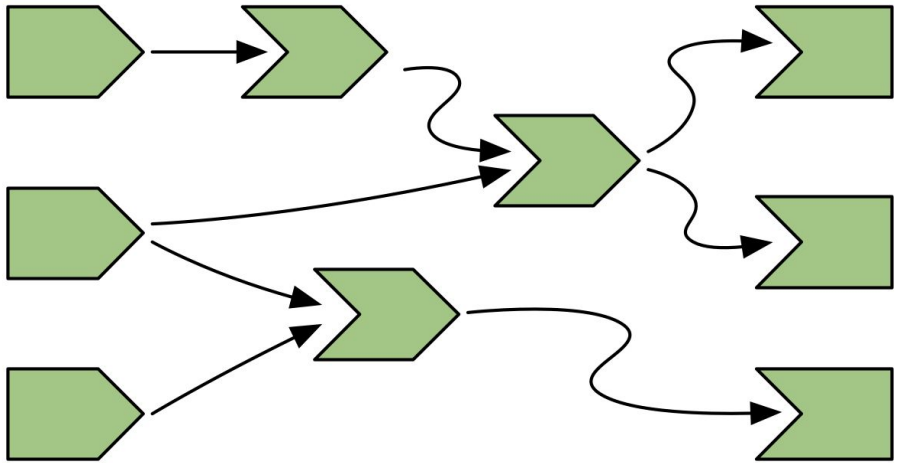
“Functional” approach

- Reactives are immutable and “pre-defined”.



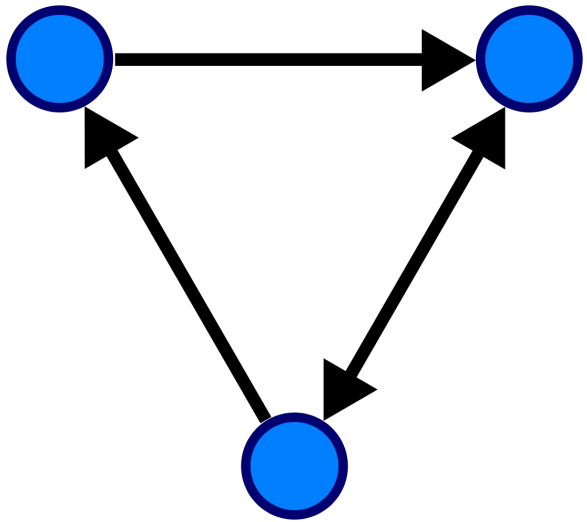
Benefits to the functional approach

- Most predictable
- Clean reactive graph
- “Cleaner” code



Downsides to functional approach

- Can be hard to manage non-linear reactive graphs.



```r
ui <- fluidPage(

  titlePanel("Functional"),
  sidebarLayout(
    sidebarPanel(
      selectizeInput(
        inputId = "file_name",
        label = "Dataset",
        choices = c("mtcars", "iris")
      ),
      selectizeInput(
        inputId = "variables",
        label = "Variables",
        choices = NULL,
        multiple = TRUE
      ),
      actionButton(
        inputId = "apply_changes",
        label = "Apply changes"
      )
    ),
    mainPanel(
      dataTableOutput("data")
    )
  )
)
```

```r
server <- function(input, output) {

  # Read dataset based on input file name
  dataset <- reactive({
    file.path("data", glue::glue("{ input$file_name }.csv")) %>%
      readr::read_csv()
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(dataset())
    )
  })

  # Select only the variables of the data set the user selects
  filtered_data <- reactive({
    dataset() %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    filtered_data()
  })

}
```

```r
ui <- fluidPage(

  titlePanel("Functional"),
  sidebarLayout(
    sidebarPanel(
      selectizeInput(
        inputId = "file_name",
        label = "Dataset",
        choices = c("mtcars", "iris")
      ),
      selectizeInput(
        inputId = "variables",
        label = "Variables",
        choices = NULL,
        multiple = TRUE
      ),
      actionButton(
        inputId = "apply_changes",
        label = "Apply changes"
      )
    ),
    mainPanel(
      dataTableOutput("data")
    )
  )
)
```

```r
server <- function(input, output) {

  # Read dataset based on input file name
  dataset <- reactive({
    file.path("data", glue::glue("{ input$file_name }.csv")) %>%
      readr::read_csv()
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(dataset())
    )
  })

  # Select only the variables of the data set the user selects
  filtered_data <- reactive({
    dataset() %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    filtered_data()
  })

}
```

```
ui <- fluidPage(

  titlePanel("Functional"),
  sidebarLayout(
    sidebarPanel(
      selectizeInput(
        inputId = "file_name",
        label = "Dataset",
        choices = c("mtcars", "iris")
      ),
      selectizeInput(
        inputId = "variables",
        label = "Variables",
        choices = NULL,
        multiple = TRUE
      ),
      actionButton(
        inputId = "apply_changes",
        label = "Apply changes"
      )
    ),
    mainPanel(
      dataTableOutput("data")
    )
  )
)
```

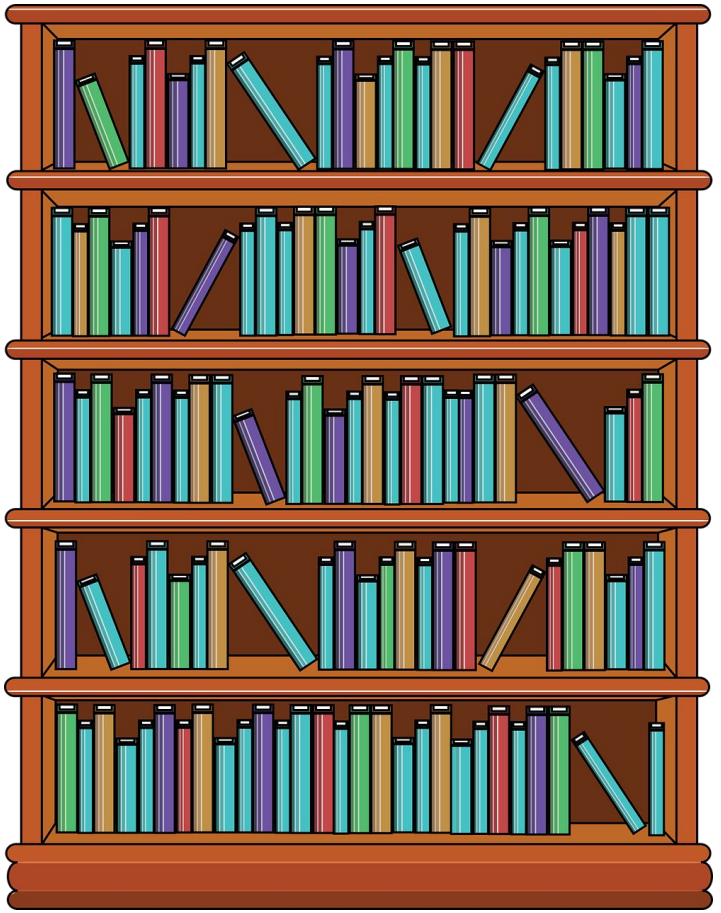
```
server <- function(input, output) {

  # Read dataset based on input file name
  dataset <- reactive({
    file.path("data", glue::glue("{ input$file_name }.csv")) %>%
      readr::read_csv()
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(dataset())
    )
  })

  # Select only the variables of the data set the user selects
  filtered_data <- reactive({
    dataset() %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    filtered_data()
  })

}
```


ui <- fluidPage(
titlePanel("Functional"),
sidebarLayout(
sidebarPanel(
selectizeInput(
inputId = "file_name",
label = "Dataset",
choices = c("mtcars", "iris")
),
selectizeInput(
inputId = "variables",
label = "Variables",
choices = NULL,
multiple = TRUE
),
actionButton(
inputId = "apply_changes",
label = "Apply changes"
)
),
mainPanel(
dataTableOutput("data")
)
)
)
server <- function(input, output) {
# Read dataset based on input file name
dataset <- reactive({
file.path("data", glue::glue("{ input$file_name }.csv")) %>%
readr::read_csv()
})
# Update select input based on the dataset selected by user
observe({
updateSelectizeInput(
inputId = "variables",
choices = colnames(dataset())
)
})
# Select only the variables of the data set the user selects
filtered_data <- reactive({
dataset() %>%
dplyr::select(!!!rlang::syms(input$variables))
}) %>%
bindEvent(input$apply_changes)
# Render de filtered dataset
output$data <- renderDataTable({
filtered_data()
})
}


ui <- fluidPage(
titlePanel("Functional"),
sidebarLayout(
sidebarPanel(
selectizeInput(
inputId = "file_name",
label = "Dataset",
choices = c("mtcars", "iris")
),
selectizeInput(
inputId = "variables",
label = "Variables",
choices = NULL,
multiple = TRUE
),
actionButton(
inputId = "apply_changes",
label = "Apply changes"
)
),
mainPanel(
dataTableOutput("data")
)
)
)
server <- function(input, output) {
# Read dataset based on input file name
dataset <- reactive({
file.path("data", glue::glue("{ input$file_name }.csv")) %>%
readr::read_csv()
})
# Update select input based on the dataset selected by user
observe({
updateSelectizeInput(
inputId = "variables",
choices = colnames(dataset())
)
})
# Select only the variables of the data set the user selects
filtered_data <- reactive({
dataset() %>%
dplyr::select(!!!rlang::syms(input$variables))
}) %>%
bindEvent(input$apply_changes)
# Render de filtered dataset
output$data <- renderDataTable({
filtered_data()
})
}

Imperative Approach

Saves state in reactive values.



Benefits of imperative approach

- Easier to manage non-linear workflows
- Reactives are mutable and can be created dynamically

Downsides to imperative approach

- “Complicated” reactive graphs.
- If not managed carefully, bugs can creep in.
- Keeping track of messy data gets hard.

```r
ui <- fluidPage(

  titlePanel("Imperative"),
  sidebarLayout(
    sidebarPanel(
      selectizeInput(
        inputId = "file_name",
        label = "Dataset",
        choices = c("mtcars", "iris")
      ),
      selectizeInput(
        inputId = "variables",
        label = "Variables",
        choices = NULL,
        multiple = TRUE
      ),
      actionButton(
        inputId = "apply_changes",
        label = "Apply changes"
      )
    ),
    mainPanel(
      dataTableOutput("data")
    )
  )
)
```

```r
server <- function(input, output) {

  globals <- reactiveValues()

  # Read dataset based on input file name
  observe({
    globals$dataset <- readr::read_csv(
      file.path("data", glue::glue("{ input$file_name }.csv"))
    )
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(globals$dataset)
    )
  })

  # Select only the variables of the data set the user selects
  observe({
    req(globals$dataset)
    globals$filtered_data <- globals$dataset %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    req(globals$filtered_data)
    globals$filtered_data
  })

}
```

```r
ui <- fluidPage(

  titlePanel("Imperative"),
  sidebarLayout(
    sidebarPanel(
      selectizeInput(
        inputId = "file_name",
        label = "Dataset",
        choices = c("mtcars", "iris")
      ),
      selectizeInput(
        inputId = "variables",
        label = "Variables",
        choices = NULL,
        multiple = TRUE
      ),
      actionButton(
        inputId = "apply_changes",
        label = "Apply changes"
      )
    ),
    mainPanel(
      dataTableOutput("data")
    )
  )
)
```

```r
server <- function(input, output) {

  globals <- reactiveValues()

  # Read dataset based on input file name
  observe({
    globals$dataset <- readr::read_csv(
      file.path("data", glue::glue("{ input$file_name }.csv"))
    )
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(globals$dataset)
    )
  })

  # Select only the variables of the data set the user selects
  observe({
    req(globals$dataset)
    globals$filtered_data <- globals$dataset %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    req(globals$filtered_data)
    globals$filtered_data
  })

}
```


ui <- fluidPage(
titlePanel("Imperative"),
sidebarLayout(
sidebarPanel(
selectizeInput(
inputId = "file_name",
label = "Dataset",
choices = c("mtcars", "iris")
),
selectizeInput(
inputId = "variables",
label = "Variables",
choices = NULL,
multiple = TRUE
),
actionButton(
inputId = "apply_changes",
label = "Apply changes"
)
),
mainPanel(
dataTableOutput("data")
)
)
)
server <- function(input, output) {
globals <- reactiveValues()
# Read dataset based on input file name
observe({
globals$dataset <- readr::read_csv(
file.path("data", glue::glue("{ input$file_name }.csv"))
)
})
# Update select input based on the dataset selected by user
observe({
updateSelectizeInput(
inputId = "variables",
choices = colnames(globals$dataset)
)
})
# Select only the variables of the data set the user selects
observe({
req(globals$dataset)
globals$filtered_data <- globals$dataset %>%
dplyr::select(!!!rlang::syms(input$variables))
}) %>%
bindEvent(input$apply_changes)
# Render de filtered dataset
output$data <- renderDataTable({
req(globals$filtered_data)
globals$filtered_data
})
}


ui <- fluidPage(
titlePanel("Imperative"),
sidebarLayout(
sidebarPanel(
selectizeInput(
inputId = "file_name",
label = "Dataset",
choices = c("mtcars", "iris")
),
selectizeInput(
inputId = "variables",
label = "Variables",
choices = NULL,
multiple = TRUE
),
actionButton(
inputId = "apply_changes",
label = "Apply changes"
)
),
mainPanel(
dataTableOutput("data")
)
)
)
server <- function(input, output) {
globals <- reactiveValues()
# Read dataset based on input file name
observe({
globals$dataset <- readr::read_csv(
file.path("data", glue::glue("{ input$file_name }.csv"))
)
})
# Update select input based on the dataset selected by user
observe({
updateSelectizeInput(
inputId = "variables",
choices = colnames(globals$dataset)
)
})
# Select only the variables of the data set the user selects
observe({
req(globals$dataset)
globals$filtered_data <- globals$dataset %>%
dplyr::select(!!!rlang::syms(input$variables))
}) %>%
bindEvent(input$apply_changes)
# Render de filtered dataset
output$data <- renderDataTable({
req(globals$filtered_data)
globals$filtered_data
})
}


ui <- fluidPage(
titlePanel("Imperative"),
sidebarLayout(
sidebarPanel(
selectizeInput(
inputId = "file_name",
label = "Dataset",
choices = c("mtcars", "iris")
),
selectizeInput(
inputId = "variables",
label = "Variables",
choices = NULL,
multiple = TRUE
),
actionButton(
inputId = "apply_changes",
label = "Apply changes"
)
),
mainPanel(
dataTableOutput("data")
)
)
)
server <- function(input, output) {
globals <- reactiveValues()
# Read dataset based on input file name
observe({
globals$dataset <- readr::read_csv(
file.path("data", glue::glue("{ input$file_name }.csv"))
)
})
# Update select input based on the dataset selected by user
observe({
updateSelectizeInput(
inputId = "variables",
choices = colnames(globals$dataset)
)
})
# Select only the variables of the data set the user selects
observe({
req(globals$dataset)
globals$filtered_data <- globals$dataset %>%
dplyr::select(!!!rlang::syms(input$variables))
}) %>%
bindEvent(input$apply_changes)
# Render de filtered dataset
output$data <- renderDataTable({
req(globals$filtered_data)
globals$filtered_data
})
}

```r
server <- function(input, output) {

  # Read dataset based on input file name
  dataset <- reactive({
    file.path("data", glue::glue("{ input$file_name }.csv")) %>%
      readr::read_csv()
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(dataset())
    )
  })

  # Select only the variables of the data set the user selects
  filtered_data <- reactive({
    dataset() %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    filtered_data()
  })

}
```

```r
server <- function(input, output) {

  globals <- reactiveValues()

  # Read dataset based on input file name
  observe({
    globals$dataset <- readr::read_csv(
      file.path("data", glue::glue("{ input$file_name }.csv"))
    )
  })

  # Update select input based on the dataset selected by user
  observe({
    updateSelectizeInput(
      inputId = "variables",
      choices = colnames(globals$dataset)
    )
  })

  # Select only the variables of the data set the user selects
  observe({
    req(globals$dataset)
    globals$filtered_data <- globals$dataset %>%
      dplyr::select(!!!rlang::syms(input$variables))
  }) %>%
    bindEvent(input$apply_changes)

  # Render de filtered dataset
  output$data <- renderDataTable({
    req(globals$filtered_data)
    globals$filtered_data
  })

}
```

Object oriented approach

Data is stored in objects which can be accessed from anywhere in the app (Objects are not reactive).

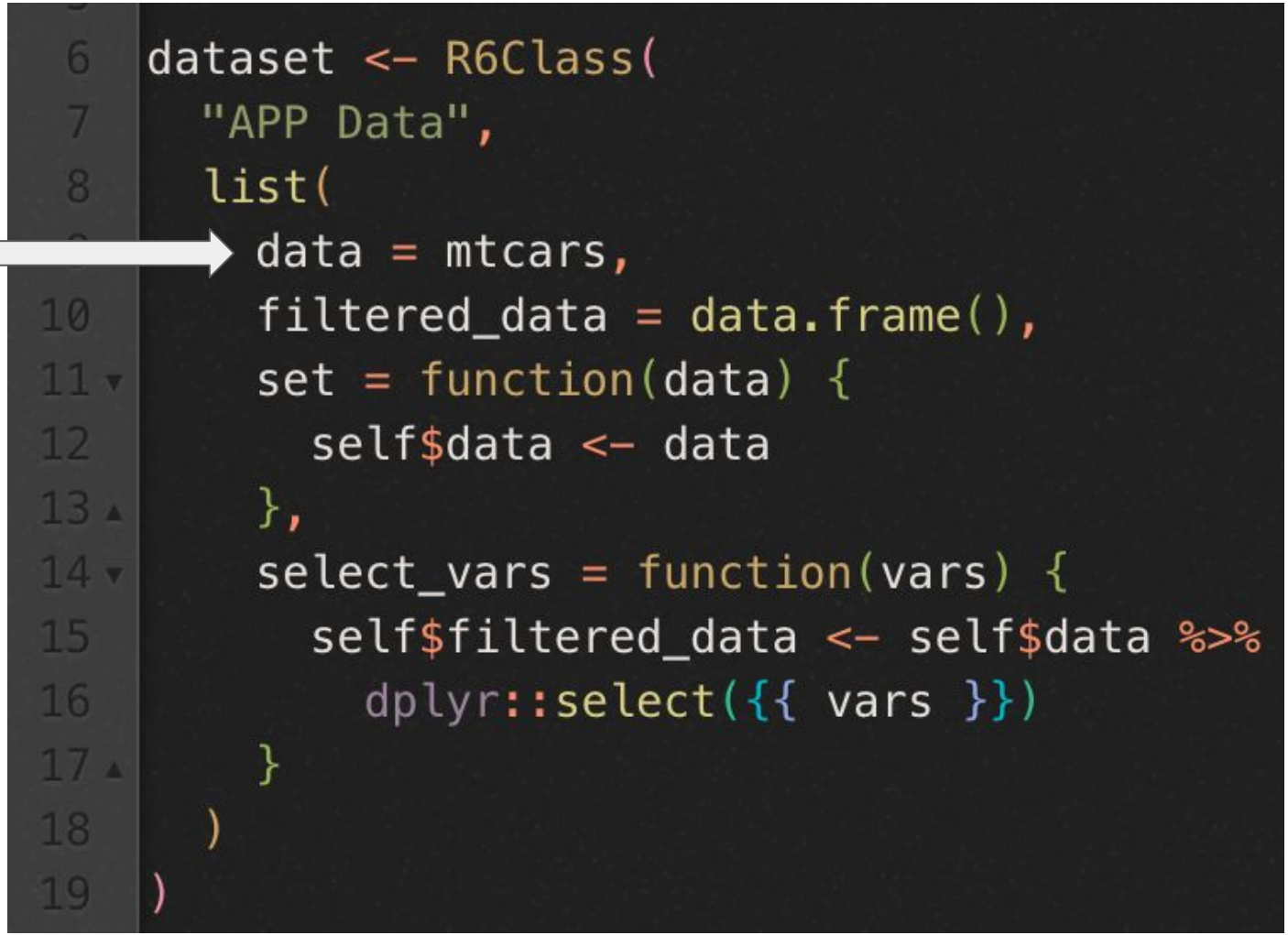
Benefits of the object oriented approach

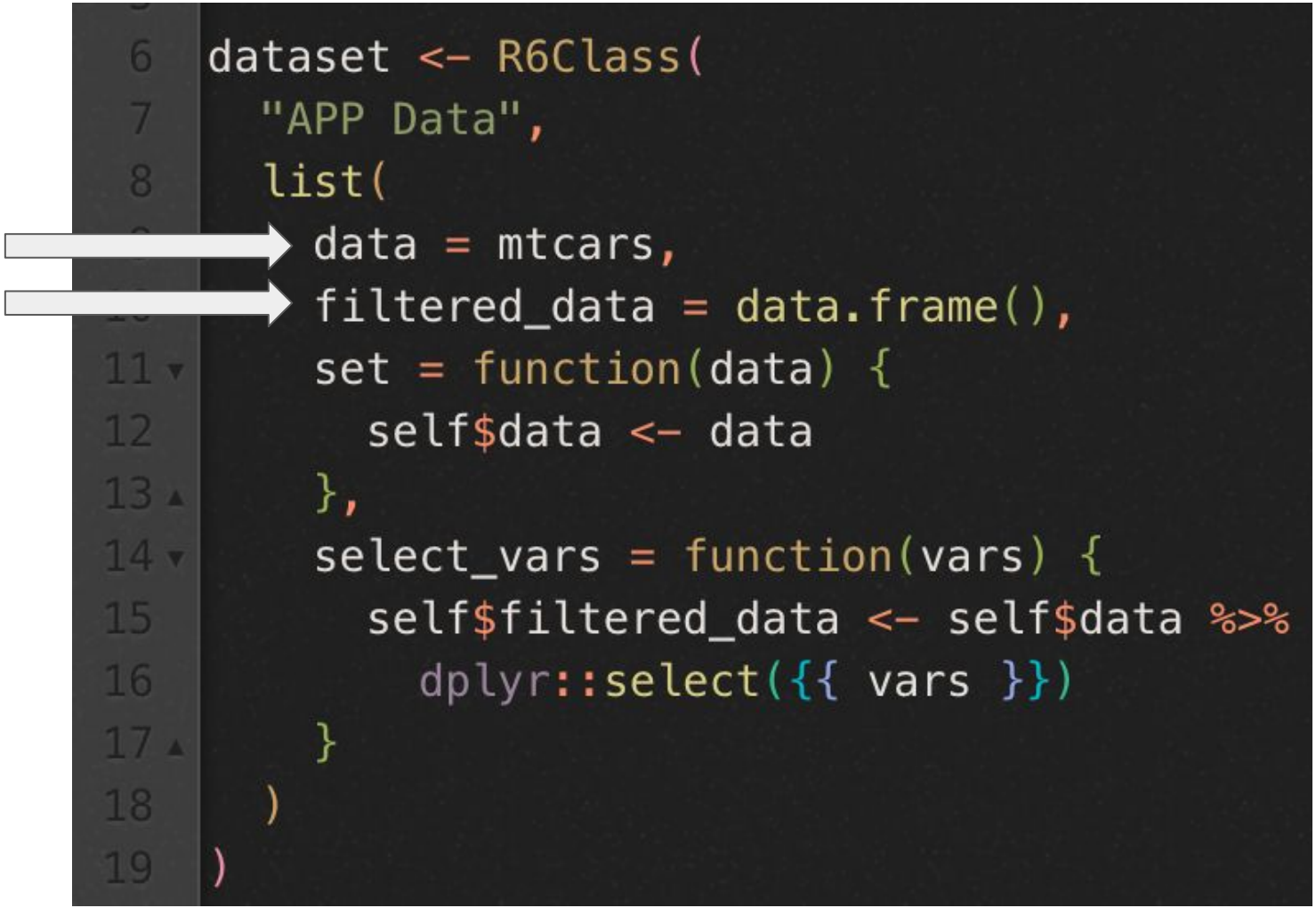
- “Easier” to organize data
- Data can be shared throughout the application without interfering with the reactive graph
- Great for bigger apps with many data sources and types.

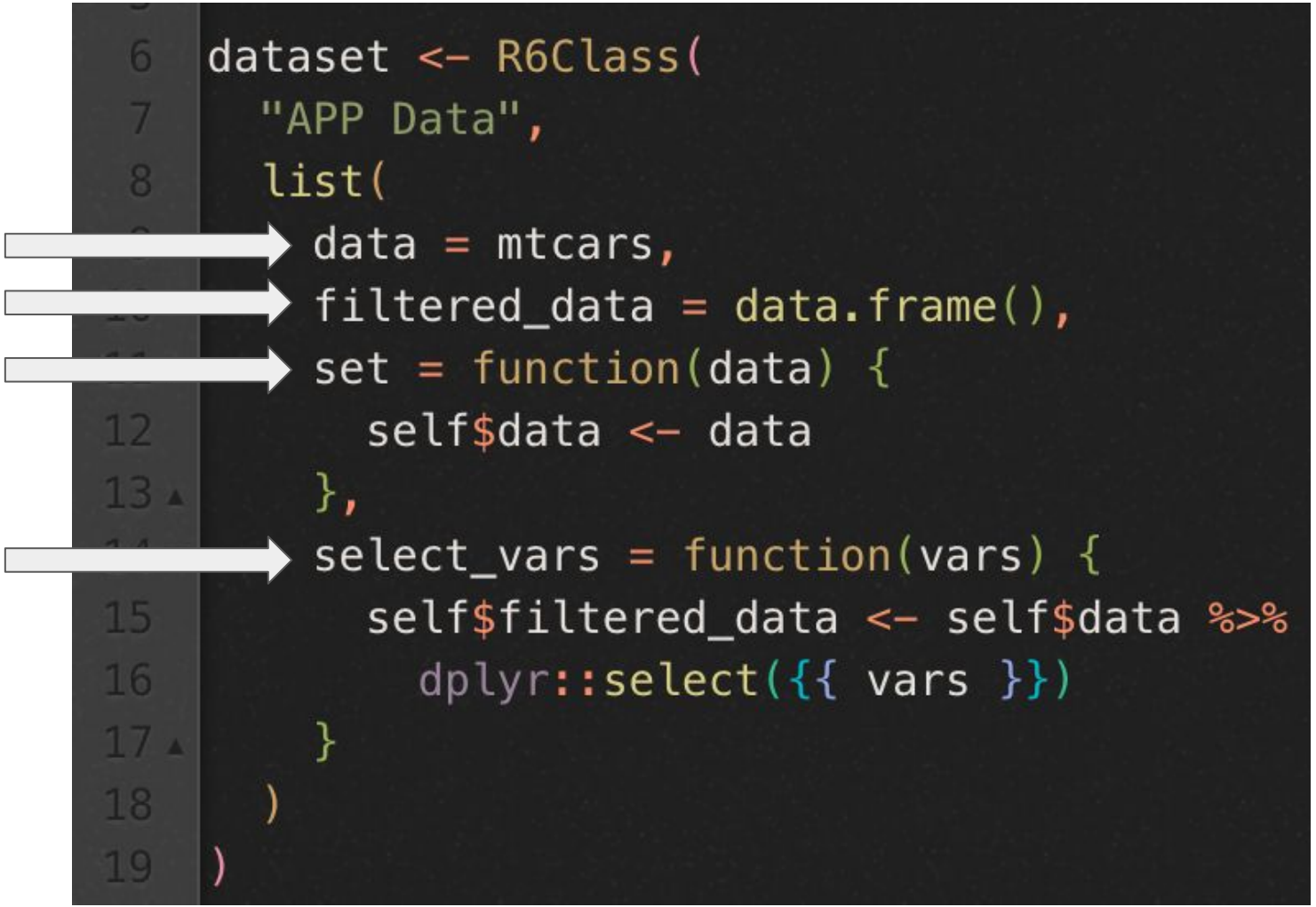
Downsides to the object oriented approach

- Can be more work.
- Works against the general notion of how R code is usually written.
- Does not interfere the reactive graph.
- Harder to mix with other approaches.

```r
dataset <- R6Class(
  "APP Data",
  list(
    data = mtcars,
    filtered_data = data.frame(),
    set = function(data) {
      self$data <- data
    },
    select_vars = function(vars) {
      self$filtered_data <- self$data %>%
        dplyr::select({{ vars }})
    }
  )
)
```


dataset <- R6Class(
"APP Data",
list(
data = mtcars,
filtered_data = data.frame(),
set = function(data) {
self$data <- data
},
select_vars = function(vars) {
self$filtered_data <- self$data %>%
dplyr::select({{ vars }})
}
)
)


dataset <- R6Class(
"APP Data",
list(
data = mtcars,
filtered_data = data.frame(),
set = function(data) {
self$data <- data
},
select_vars = function(vars) {
self$filtered_data <- self$data %>%
dplyr::select({{ vars }})
}
)
)


dataset <- R6Class(
"APP Data",
list(
data = mtcars,
filtered_data = data.frame(),
set = function(data) {
self$data <- data
},
select_vars = function(vars) {
self$filtered_data <- self$data %>%
dplyr::select({{ vars }})
}
)
)

Initialize a new object

Init triggers

Set the data to that which the user selected

Trigger “selected_dataset”

Run when “selected_dataset” is triggered

Set selected variables to those in input$variables
and trigger “selected_variables”

Render the data when “selected_variables” is triggered

Conclusion

- Most approaches are completely valid for production grade applications.
- It is important to determine which approach is easier to work with for a particular project.
- Consistency is key, code is understandable as long as it is consistent.

- Software Engineer
- Specialise in building Shiny Apps
- Currently work building Shiny Applications for use in healthcare institutions
- Free/Open Source advocate and enthusiast

andresquinterom

andyquinterom