Shiny Cheat Sheet

A comprehensive cheat sheet covering essential Shiny concepts, functions, and best practices for building interactive web applications with R.

Core Concepts

Basic Structure

A Shiny app consists of two main parts: a user interface (`ui`) and a server function (`server`).

```
library(shiny)

ui <- fluidPage(
  # UI elements go here
)

server <- function(input, output) {
  # Server logic goes here
}

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

The `ui` defines the layout and appearance of the app. It uses functions like `fluidPage`, `sidebarLayout`, and various input/output elements.

The `server` function contains the logic that makes the app interactive. It uses `input` to access values from the UI and `output` to render content.

UI Elements

`fluidPage()`	Creates a fluid layout that adapts to different screen sizes.
`sidebarLayout()`	Creates a layout with a sidebar and a main panel.
`sidebarPanel()`	Contains the elements in the sidebar.
`mainPanel()`	Contains the main content of the app.
`titlePanel()`	Adds a title to the page.
`br()`	Adds a line break.

Reactivity

Shiny uses a reactive programming model. When an input value changes, Shiny automatically re-executes the code that depends on that value.

`reactive()`: Creates a reactive expression that only re-evaluates when its dependencies change.

`observe()`: Executes code in response to reactive changes, but doesn't return a value.

`observeEvent()`: Executes code only when a specific event occurs (e.g., a button click).

Input Components

Common Input Widgets

`textInput(inputId, label, value = "")`	Creates a text input box.
`numericInput(inputId, label, value, min = NA, max = NA, step = NA)`	Creates a numeric input box with optional min, max, and step.
`sliderInput(inputId, label, min, max, value, step = NULL)`	Creates a slider input.
`selectInput(inputId, label, choices, selected = NULL, multiple = FALSE)`	Creates a dropdown select box.
`radioButtons(inputId, label, choices, selected = NULL)`	Creates a set of radio buttons.
`checkboxGroupInput(inputId, label, choices, selected = NULL)`	Creates a group of checkboxes.
`checkboxInput(inputId, label, value = FALSE)`	Creates a single checkbox.
`dateInput(inputId, label, value = NULL, min = NULL, max = NULL, format = "yyyy-mm-dd", language = "en")`	Creates a date input.
`fileInput(inputId, label, multiple = FALSE, accept = NULL, width = NULL, buttonLabel = "Browse...", placeholder = "No file selected")`	Creates a file upload input.

Output Components

Rendering Outputs

`renderPlot()`	Renders a plot (e.g., from `ggplot2`).
`renderTable()`	Renders a static table (e.g., from a data frame).
`renderDataTable()`	Renders an interactive table (using DataTables library).
`renderText()`	Renders text output.
`renderPrint()`	Renders the output of `print()`.
`renderUI()`	Renders arbitrary HTML or Shiny UI elements. Useful for dynamic UIs.

Displaying Outputs in UI

`plotOutput(outputId, width = "100%", height = "400px")`	Displays a plot rendered by `renderPlot()`.
`tableOutput(outputId)`	Displays a table rendered by `renderTable()`.
`dataTableOutput(outputId)`	Displays an interactive table rendered by `renderDataTable()`.
`textOutput(outputId, inline = FALSE)`	Displays text rendered by `renderText()`.
`verbatimTextOutput(outputId, placeholder = FALSE)`	Displays text (verbatim) rendered by `renderPrint()`.
`uiOutput(outputId)`	Displays UI elements rendered by `renderUI()`.

Advanced Features

Download Handlers

Use `downloadHandler()` to create download buttons. Specify the filename and the content to be downloaded.

```
downloadButton("downloadData", "Download")

server <- function(input, output) {
  output$downloadData <- downloadHandler(
    filename = function() { paste("data-",
Sys.Date(), ".csv", sep="") },
    content = function(file) {
      write.csv(data, file)
    }
  )
}
```

Session Management

The `session` object provides information about the user's session and allows you to control the app's behavior.

```
server <- function(input, output, session) {
  observe({
    # Update input values
    updateTextInput(session, "text", value =
"New Value")
  })
}
```

JavaScript Integration

You can integrate JavaScript code into your Shiny apps using `tags$script` in the UI or by sending messages from R to JavaScript.

```
# UI
tags$script("alert('Hello from
JavaScript!');")
```

Modules

Shiny modules allow you to encapsulate UI and server logic into reusable components. This promotes code organization and reusability.

```
# Define a module
myModuleUI <- function(id) {
  ns <- NS(id)
  tagList(
    textInput(ns("textInput"), "Enter text:"),
    textOutput(ns("textOutput"))
  )
}

myModuleServer <- function(id) {
  moduleServer(
    id,
    function(input, output, session) {
      output$textOutput <-
renderText(input$textInput)
    }
  )
}

# Use the module in the main app
ui <- fluidPage(
  myModuleUI("myModule")
)

server <- function(input, output, session) {
  myModuleServer("myModule")
}
```