



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

<code>fluidPage()</code>	Creates a fluid layout that adapts to different screen sizes.
<code>sidebarLayout(yout())</code>	Creates a layout with a sidebar and a main panel.
<code>sidebarPanel(...)</code>	Contains the elements in the sidebar.
<code>mainPanel(...)</code>	Contains the main content of the app.
<code>titlePanel(...)</code>	Adds a title to the page.
<code>br()</code>	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.
<code>reactive()</code> : Creates a reactive expression that only re-evaluates when its dependencies change.
<code>observe()</code> : Executes code in response to reactive changes, but doesn't return a value.
<code>observeEvent()</code> : Executes code only when a specific event occurs (e.g., a button click).

Input Components

Common Input Widgets

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

Output Components

Rendering Outputs

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

Displaying Outputs in UI

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

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)
    }
  )
}
```

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")
}
```

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")
  })
}
```