**Video Script: Section 6 Video 4 – understanding reactive programming**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Description | Action on screen | Narration |
| 1 | Introduction  (Outcome and why it is desirable)  This should give the viewer an idea of the outcome of the task at the beginning of the videos and set the stage and expectations of the viewer. | Refer to PPT | In this video, we’re going to have a closer look at reactive programming, on which shiny is based. |
| 2 | Context(Problem/Solution)  Present the viewer with a real-world solution and how the situation would pose as a challenge. It always helps to draw the viewer's attention using a use-case. Metadata template can be used here. |  | To program interactive graphics, you need to write some code that is executed only when some change occurs.  Using Shiny’s concept of reactive programming, you’ll see how to tackle this. |
| 3 | Guidance (How to do it and how it works): |  | With Shiny, you only need to declare an expression as ‘reactive’ for it to automatically pass messages between the UI and the server.  You don’t need to write all the necessary code: shiny does it for you behind the scenes. |
| 4 |  | Switch to RStudio. Run in the console:  library(“shiny”)  runApp(“activity\_06\_04”)  A description... | Open R and run the shiny app ‘activity\_06\_04’:  library(“shiny”)  runApp(“activity\_06\_04”) |
| 5 |  | Change the text in the text box. | Entering some text in the text box updates the content of the first text area automatically. |
| 6 |  | Select a different dataset | Selecting a dataset in the menu updates the second text area. |
| 7 |  |  | Note how every time you select a dataset, a new random value is shown in the first line of the text area. |
| 8 |  | Change the text in the text box | Changing the value in the text box updates the first text area only: the random value doesn’t change because it hasn’t been recalculated. |
| 9 |  |  | Only the parts that depend on changes in the UI are updated. Shiny doesn’t waste time updating everything. |
| 10 |  | Open UI.R and server.R in the editor. | Let’s look at the code. |
| 11 |  | Switch to server.R | The functions renderText and renderPlot are ‘reactive’, i.e. they listen to changes to input. |
| 12 |  |  | Consider output$dataSummary |
| 13 |  |  | The expression depends on input$dataset, which comes from the dropdown menu in UI.R. Any change in the menu will affect the expression, which will change the value of output$dataSummary, which will in turn be reflected in the webpage.  This can be represented by this diagram: you set the connections between the inputs and the outputs but all the message-passing is done for you automatically. |
| 14 |  | Copy and paste:  # you can also define your own reactive expression with reactive()  myReactiveExpression<- reactive({  switch(input$dataset,  "rock" = rock,  "pressure" = pressure,  "cars" = cars)  }) | You can write your own reactive expression. Uncomment the following code in the body of shinyServer.  This expression listens to input$dataset. |
| 15 |  | Replace  dataset <- switch(input$dataset,  "rock" = rock,  "pressure" = pressure,  "cars" = cars)  With  dataset <- myReactiveExpression() | You can use it for output$dataSummary and have the changes cascading from one reactive expression to the next.  Reactive functions are especially useful to pass on processed values to multiple outputs.  Replace get(input$dataset) by get(myReactiveExpression())  Note that you need the two parens after myReactiveExpression because it is a function.  Your code can be represented by this diagram. output$dataSummary now listens to myReactiveExpression, which in turn listens to input$dataset.  ~~Your code will typically contain a mixture of reactive and usual functions.~~  Run the app. It works just the same.  You could have defined myReactiveExpression as a normal function. The app would have run as well.  However, it is more efficient to declare functions as reactive when they depend on changes in the user interface and are used by multiple functions.  This is because reactive functions save their latest calculated values and can avoid repeating their calculations.  Consult the Shiny tutorial for more details about the inner workings of reactive programming. |
| 16 | Conclusion:The video concludes by showing the viewer that the goal has been achieved, and reminding them why they should be happy about that. A PowerPoint summary slide with the key points emphasized would make it easier for the viewer to remember what was covered in the video | Back to PPT | In this video, we have seen in more detail how information is passed on from the webpage to the server and how shiny only updates what is necessary.  In the next video, we’ll see what strategy to apply when re-generating the webpage after any change would be too slow. |