An Introduction to Shiny

A hands-on workshop

Andreas M. Brandmaier and Leonie Hagitte

7/1/23

6th ESLR Workshop, 2023: Social Learning in Complex Systems

Welcome

6th ESLR Workshop, 2023: Social Learning in Complex Systems

Ask questions anytime

Shiny

> Shiny is an R package that makes it easy to build interactive web apps in R

Apps can be

- standalone,
- deployed to a website,
- or be part of an interactive (Markdown) document

Required software

You need to install these software packages in order to follow along with the examples of today:

- **R**: https://cran.r-project.org
- **RStudio**: https://posit.co/download/rstudio-desktop/
- **shiny**
- **tidyverse packages** (and some others)

```
install.packages(c("shiny","tidyverse", "shinydashboard","palmerpenguins"))
```

Workshop materials

Please find the slides and code snippets here:

https://github.com/brandmaier/shiny-workshop-2023

What to expect

- This is a hands-on workshop; you'll get the most out of it if you download the materials and actively participate
- Introductory R coding skills are OK! We have exercises at varying levels of proficiency
- The workshop materials remain open and accessible after the workshop
- Feel free to team up!

Goals

Objectives of today

- Learn about the structure of a shiny application.
- Learn how to create shiny apps from a template.
- Learn how to think in terms of *inputs* and *outputs*.
- Write apps yourselves (using simulated data, real data or *your* data)

Content

Let’s talk about…

- User-interface / Layout
- Reactivity / Logic
- Awesome visualizations

Anatomy of a Shiny app

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) {  }
6 )
```

We first load the `shiny` package and define a `shinyApp`, which really is only a function call with two arguments.

Anatomy of a Shiny app

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) {  }
6 )
```

The `ui` specifies the *visible* user interface

- Dynamic elements *inputs* and *outputs*
- Static elements like headings, text, static images
- A layout how to arrange these things

Anatomy of a Shiny app

```
1 library(shiny)
2
3 shinyApp(
4   ui = list(),
5   server = function(input, output, session) {  }
6 )
```

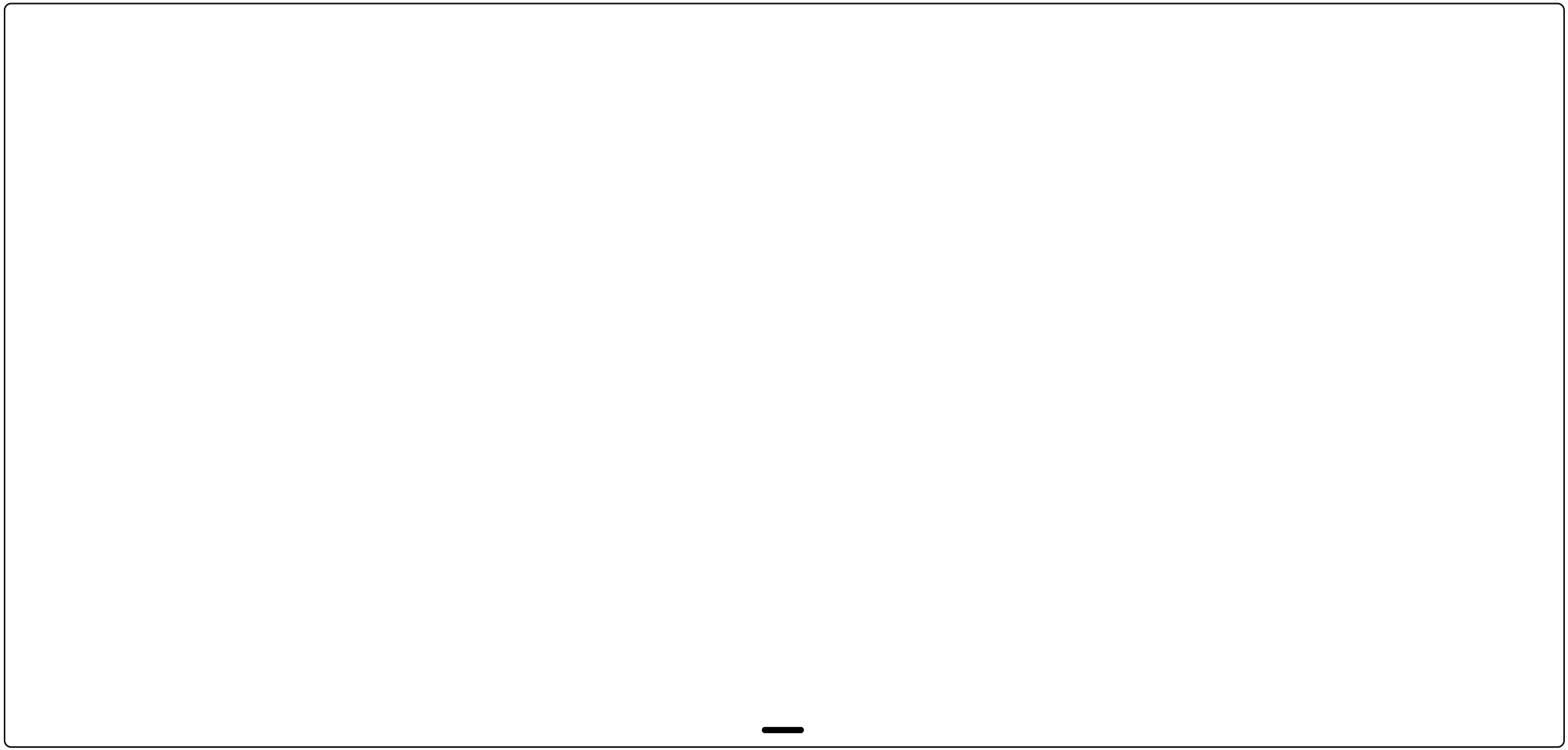
The `server` is *invisible* and is responsible for all computations

- The `server` monitors *inputs*
- When inputs change, *outputs* are updated (*reactivity*)

User-interface

Shiny Widgets Gallery

shiny.rstudio.com/gallery/widget-gallery.html



Example

Inputs have unique ids that correspond to server-side variables, a label, a starting value and extra options (e.g., range restrictions, etc.)

```
textInput(inputId="familyname", label="Family name:",  value="Steve Miller"  )
```

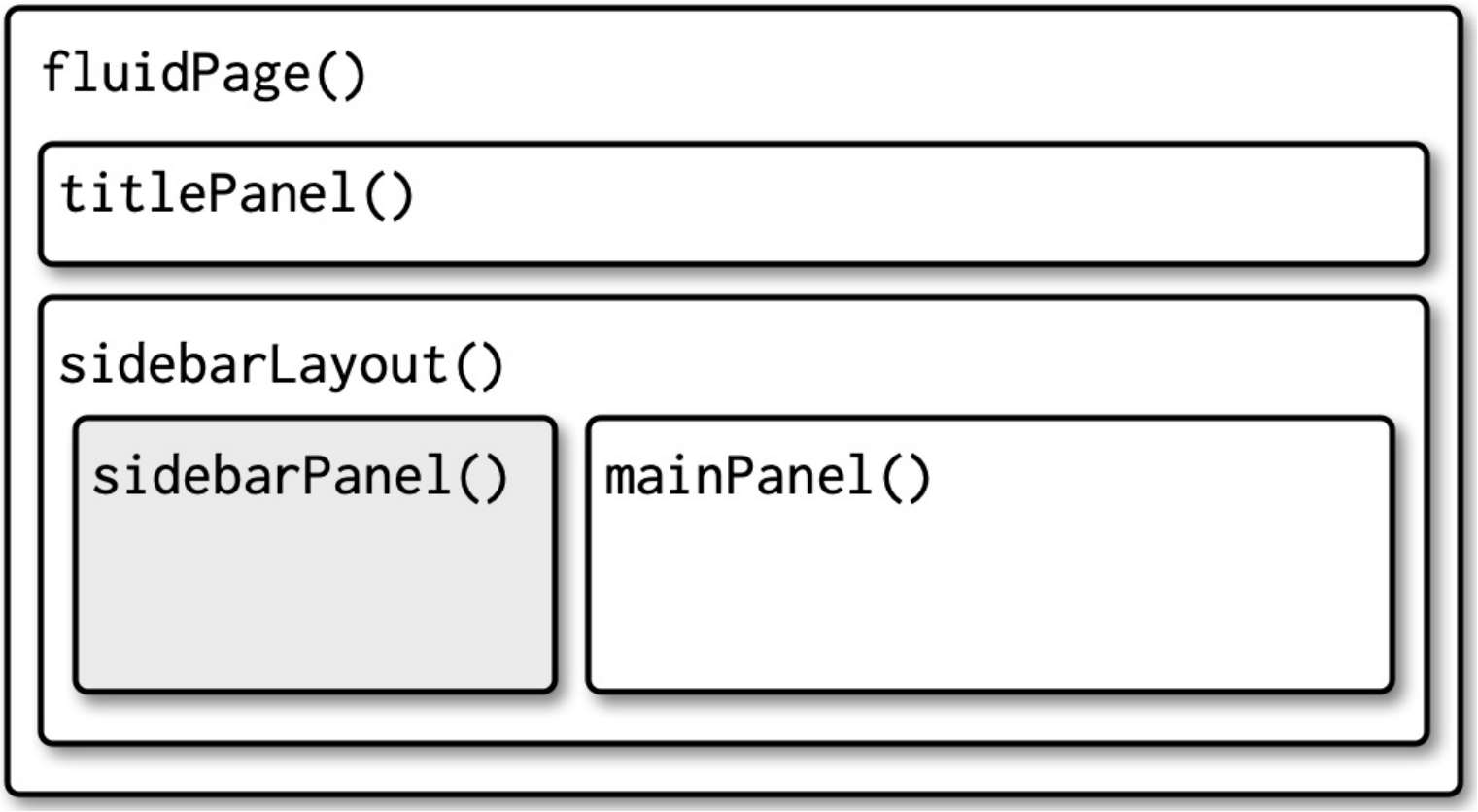
or

```
numericInput(inputId="age", label="Age (in years):",  value=1, min=0, max=150 )
```

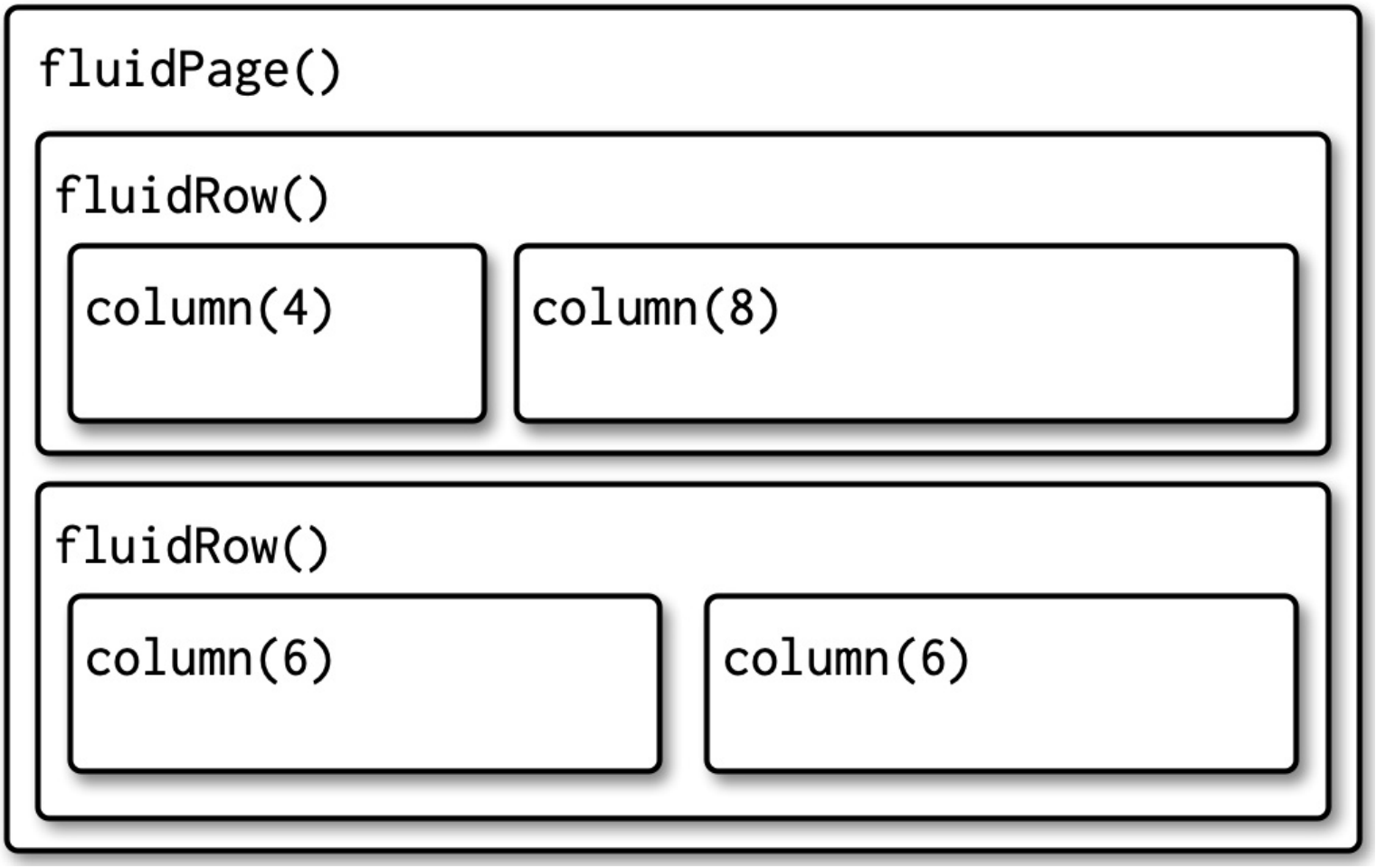
On the server, we will be able to access variables `input$familyname` and `input$age`

Layout

Sidebar layout



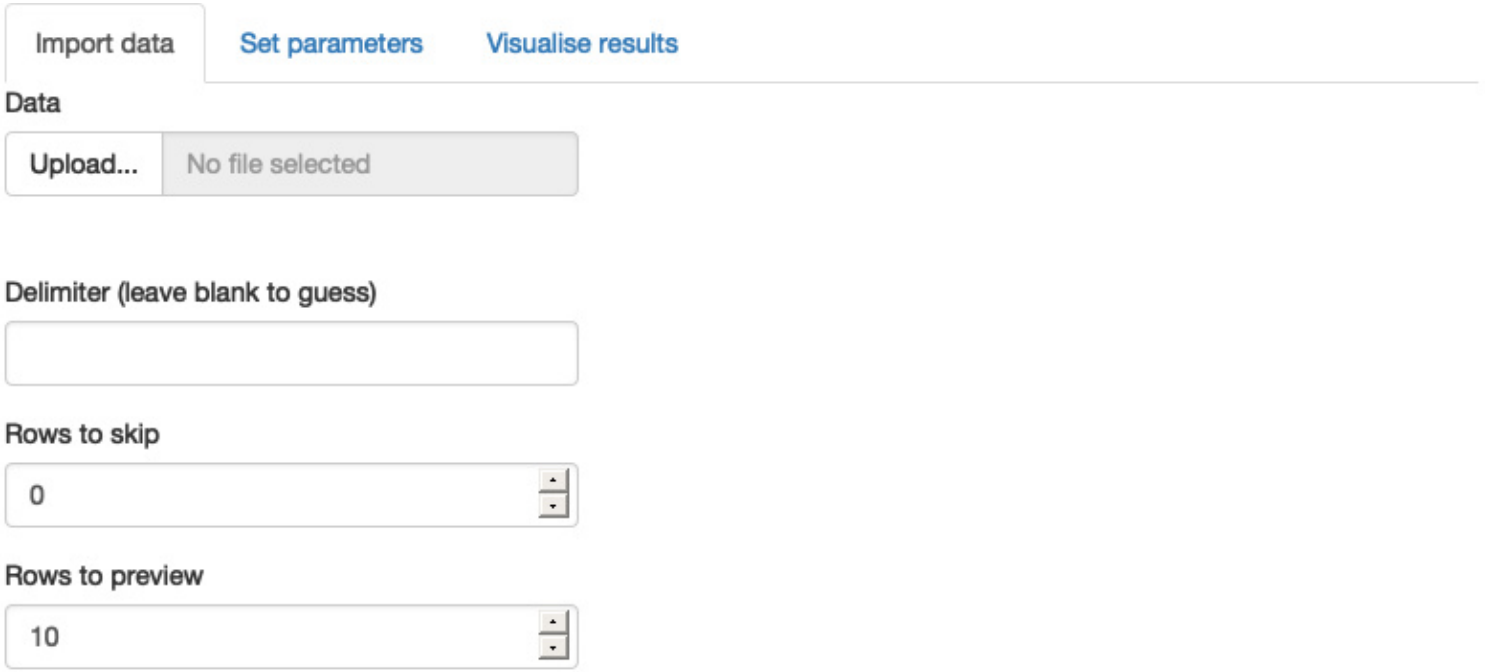
Multi-row layout



From Mastering Shiny

Other layouts

Many more, e.g. Tabsets - see `tabsetPanel()`



From Mastering Shiny

Outputs

Example output elements (placeholders for dynamic content):

- `textOutput()` or `htmlOutput()`
- `plotOutput()`
- `tableOutput()`

You can use

```
1 help.search("Output", package = "shiny")
```

to find other output functions in shiny.

—

Outputs and Renderers

Each `*Output()` function has a corresponding `render*()` server-side function. For example:

- `textOutput()` → `renderText()`
- `plotOutput()` → `renderPlot()`
- `tableOutput()` → `renderTable()`

Server logic: Accessing inputs

```
shinyApp(
  ui = list(),
  server = function(input, output, session) {  }
)
```

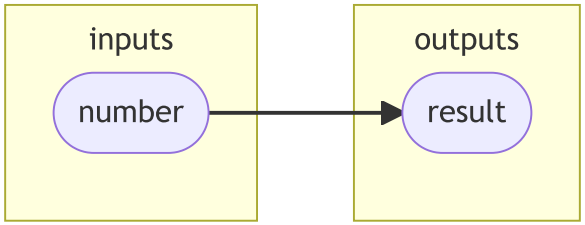
- Inputs are accessed in the *server* function via the *input* argument.
- Inputs are *reactive*, meaning that changes trigger updates to outputs.

Example: A pocket calculator

Demo 1 - Plus One

Demo: We write a simple calculator that adds +1 to a number we enter.

The simplest structure of a reactive program involves just a source and an endpoint:



Demo 1 - Plus One

R/demo1.R

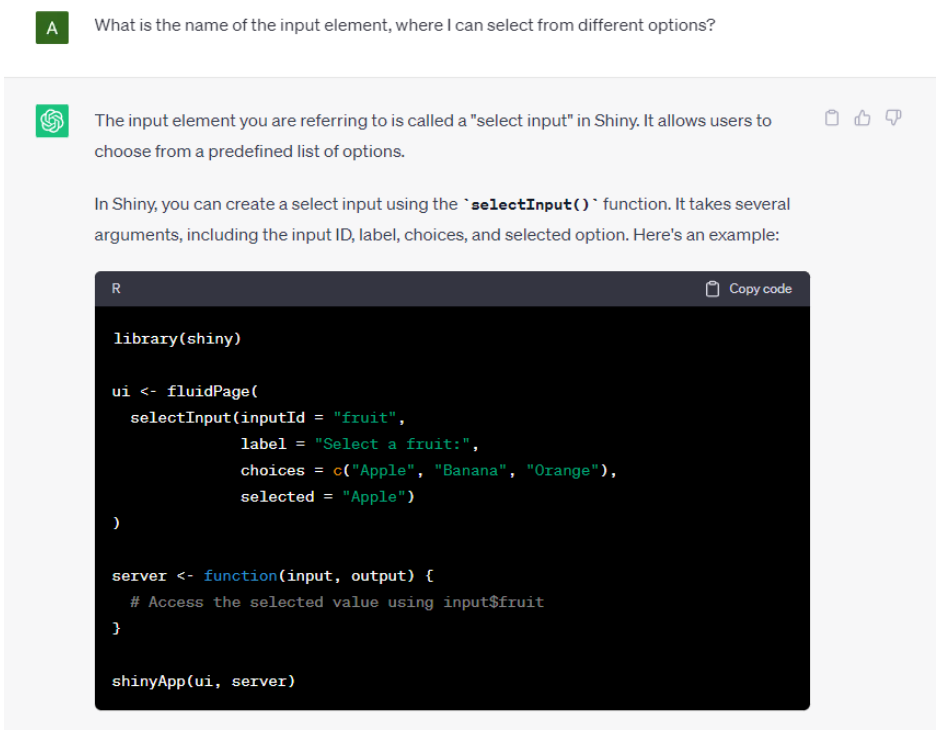
```
 1 library(shiny)
 2
 3 # Define UI for application that draws a histogram
 4 ui <- fluidPage(
 5
 6   # Application title
 7   titlePanel("Calculator"),
 8
 9   # Sidebar with a slider input for number of bins
10   sidebarLayout(
11     sidebarPanel(
12       numericInput("number",
13                    "Number", value=0)
14     ),
15
16     # Show a plot of the generated distribution
17     mainPanel(
18       h3("Result"),
19       textOutput("result")
```

Seeking AI help

Large language models are great companions for programming

Here is a ChatGPT link (requires Microsoft or Google account) to answer your questions (but please ask us as well any time)

ChatGPT companion for Shiny



Your turn - Exercise 1

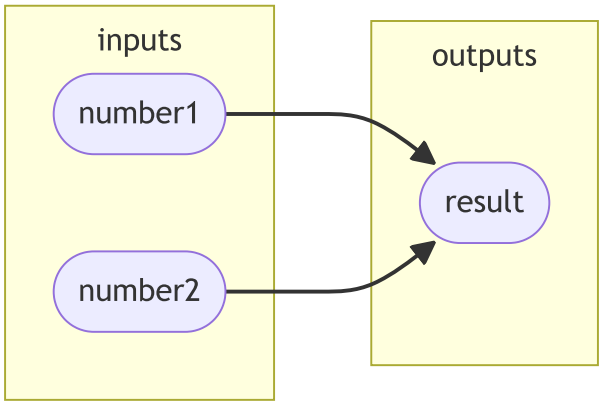
Copy the code from the previous slide (or open `R/demo1.R`) and run it in R

Check that you are able successfully run the shiny app and are able to interact with it.

- If everything is working try modifying the code (e.g. try adding a second number input and change the logic so that both numbers are added).

Reactive diagram

The reactive diagram of this solution shows two inputs and one output:



Solution

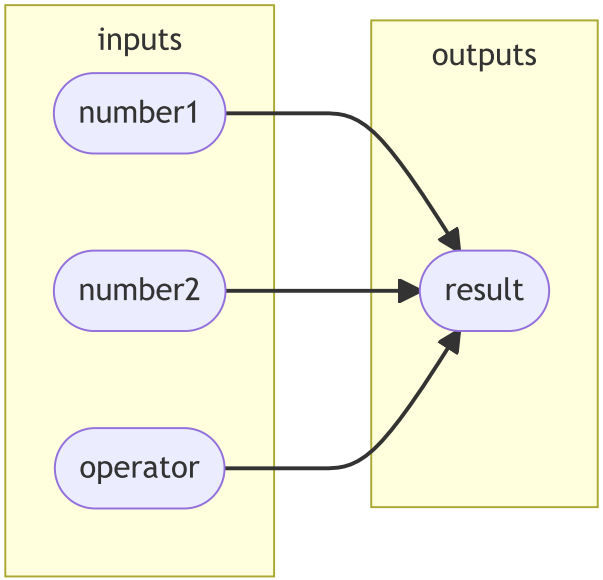
R/solution1_1.R

```
 1 library(shiny)
 2
 3 # Define UI for application that draws a histogram
 4 ui <- fluidPage(
 5
 6     # Application title
 7     titlePanel("Calculator"),
 8
 9     # Sidebar with a slider input for number of bins
10     sidebarLayout(
11         sidebarPanel(
12             numericInput("n1",
13                          "Number", value=0),
14             numericInput("n2",
15                          "Number", value=0)
16         ),
17
18         # Show a plot of the generated distribution
```

Your Turn - Exercise 2

- Continue with your code (or from `R/solution1_1.R`) and add a menu to choose different operators (e.g., plus, minus, …)
- For example, add a `selectInput(inputId, label, choices)`
- Add server-side logic to implement the different operators

Reactive diagram



Solution

R/solution1_2.R

```
 1 library(shiny)
 2
 3 # Define UI for application that draws a histogram
 4 ui <- fluidPage(
 5
 6     # Application title
 7     titlePanel("Calculator"),
 8
 9     # Sidebar with a slider input for number of bins
10     sidebarLayout(
11         sidebarPanel(
12             numericInput("n1",
13                          "Number", value=0),
14             numericInput("n2",
15                          "Number", value=0),
16             selectInput("operator","Operator",c("+","-","/","*"))
17         ),
18
19         # Show a plot of the generated distribution
```

Formatting text

We can use HTML elements to style text. E.g.,

```
<b>Bold</b> or <i>Italics</i>,h1>First-level heading</h> <h2>Second-level
heading</h2>, ...
```

In UI as static or dynamic elements:

```
1      h2("Title"),
2      htmlOutput(outputId = "result")
```

On the server:

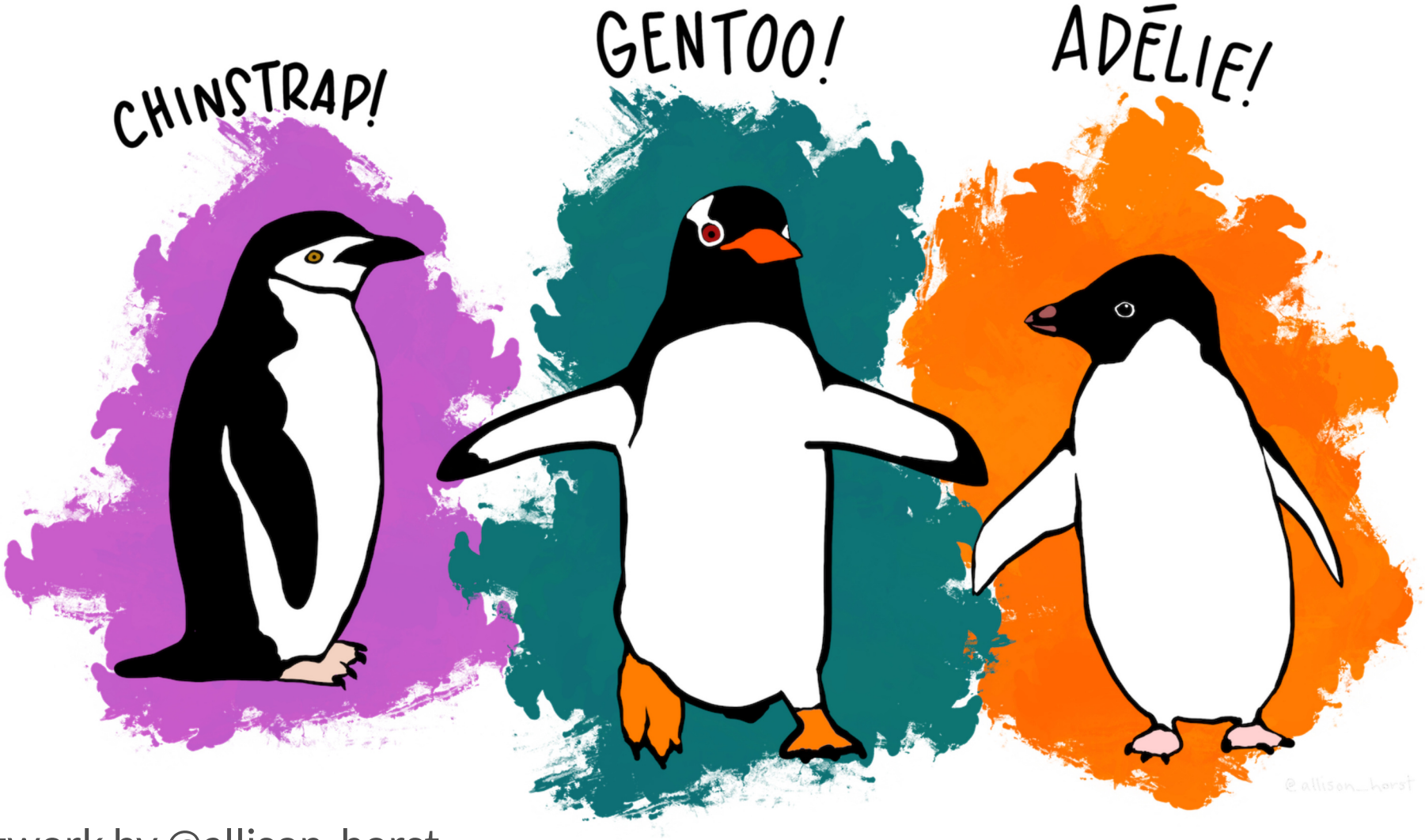
```
1  output$result <- renderText({ "<h2>Headline</h2>" })
```

Solution

R/solution1_3.R

```
 1 library(shiny)
 2
 3 # Define UI for application that draws a histogram
 4 ui <- fluidPage(
 5
 6     # Application title
 7     titlePanel("Calculator"),
 8
 9     # Sidebar with a slider input for number of bins
10     sidebarLayout(
11         sidebarPanel(
12             numericInput("n1",
13                          "Number", value=0),
14             selectInput("operator","Operator",c("+","-","/","*")),
15             numericInput("n2",
16                          "Number", value=0)
17         ),
18
19         # Show a plot of the generated distribution
```

Who doesn't like penguins?



Artwork by @allison_horst

—

Palmer Penguins

We are going to use the `penguins` dataset from `palmerpenguins`

species	island	bill_length_mm	bill_depth_mm	fli
Adelie	Torgersen	39.1	18.7	
Adelie	Torgersen	39.5	17.4	
Adelie	Torgersen	40.3	18.0	
Adelie	Torgersen	NA	NA	
Adelie	Torgersen	36.7	19.3	
Adelie	Torgersen	39.3	20.6	

Reactive expression

R/challenge2.R

```
 1 library(shiny)
 2 library(tidyverse)
 3 library(palmerpenguins)
 4
 5 # Define UI for application that draws a histogram
 6 ui <- fluidPage(
 7
 8     # Application title
 9     titlePanel("Penguins"),
10
11     # Sidebar with a slider input for number of bins
12     sidebarLayout(
13         sidebarPanel(
14             # <-------   here go input elements
15         ),
16
17         # Show a plot of the generated distribution
18         mainPanel(
```

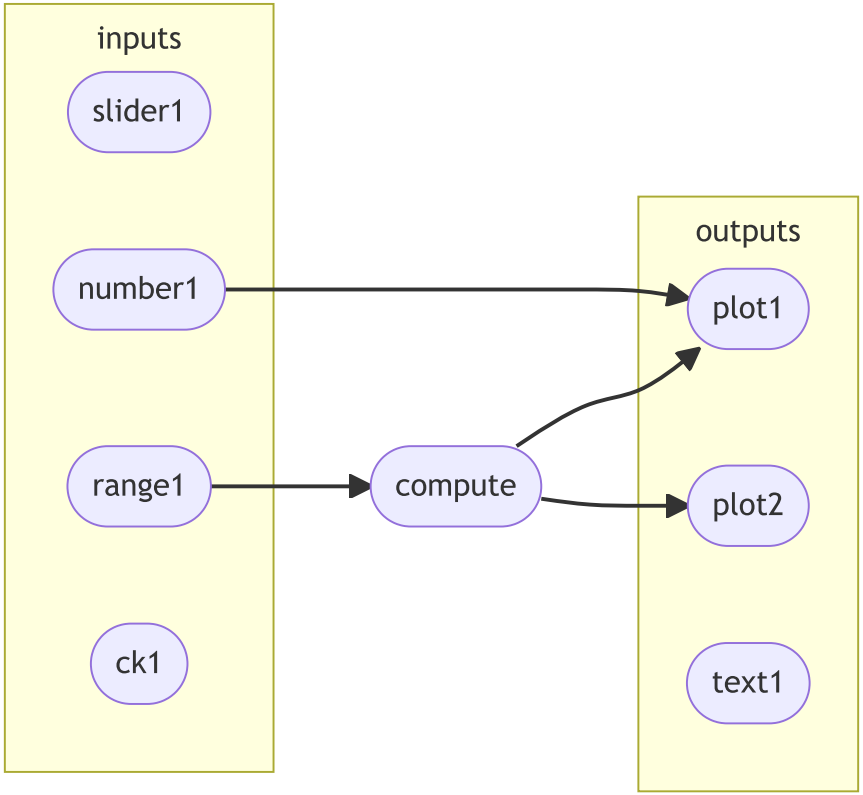
Your Turn - Exercise 3

- Copy the code from the previous slide (or open `R/challenge2.R`) and run it in R
- Add logic to create a second plot as output `plot2` on the server
- Add extra inputs (e.g., add a `selectInput` for subgroup selection of penguin species) or add a `rangeInput` to display only certain ranges of years, or make point size adjustable by a given variable (`selectInput` or a `checkboxInput`).

DRY - Don’t repeat yourself

- Assume a range input (`sliderInput(value=c(0,10))`) that filters data
- Filter logic should be executed only once for every relevant output
- Never copy&paste server logic, instead use a `reactive` element

DRY - Don’t repeat yourself



Reactives

Their primary use is similar to a function in an R script, they help to

- avoid repeating yourself
- decompose complex computations into smaller / more modular steps
- can improve computational efficiency by breaking up / simplifying reactive dependencies

DRY - Solution

R/demo3.R

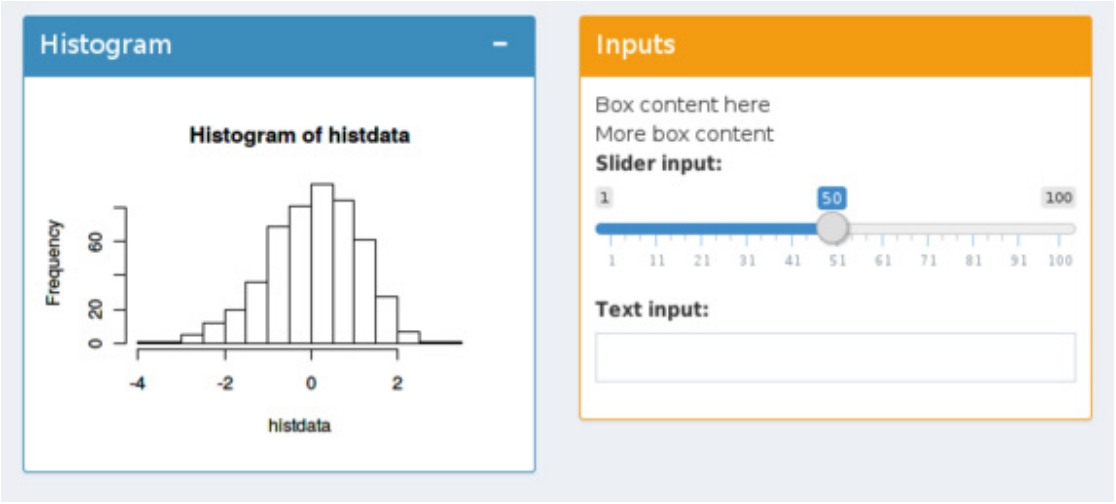
```
 1 library(shiny)
 2 library(tidyverse)
 3 library(palmerpenguins)
 4
 5 # Define UI for application that draws a histogram
 6 ui <- fluidPage(
 7
 8     # Application title
 9     titlePanel("Penguins"),
10
11     # Sidebar with a slider input for number of bins
12     sidebarLayout(
13         sidebarPanel(
14             sliderInput("rng", "Range ",value=c(3000,5000),min=2700, max=63
15             selectInput("size", label="Size", choices=c("flipper_length_mm"
16             checkboxInput("grp",label="Subgroups", value=TRUE)
17         ),
18
19         # Show a plot of the generated distribution
```

Deployment

- Free online deployment at https://www.shinyapps.io/ after registration
- Free account limited (e.g., 25h operating hours, 5 apps; more plans available)
- Sharing your app for others to run it locally (e.g., via OSF)
- Reproducibility! Make sure that everything is contained, no absolute file paths were used (see `here` package) and that all dependencies are loaded

Dashboards

Package `shinydashboard` has some nice GUI elements for dashboards:



Demo Dashboard

R/demo7.R

```
 1 library(shinydashboard)
 2
 3 ui <- dashboardPage(
 4   dashboardHeader(title = "Value boxes"),
 5   dashboardSidebar(),
 6   dashboardBody(
 7     fluidRow(
 8       # A static valueBox
 9       valueBox(20, "New Orders", icon = icon("credit-card")),
10
11       # Dynamic valueBox
12       valueBoxOutput("progressBox"),
13
14     ),
15     fluidRow(
16       # Clicking this will increment the progress amount
17       box(width = 4, actionButton("count", "Do some work"))
18     )
```

Simulation

Shiny is useful for simulating data (multivariate distributions, network graphs, agents, …)

- Inputs allow us to vary simulation parameters
- Outputs display simulation results
- We use a `reactive()` to generate our dataset, so that it can be reused in different places
- `downloadButton` and `downloadHandler` allow us to download the simulated data files for later analyses

—

Simulation Stub

R/demo6.R

```
 1 library(shiny)
 2
 3 # Define UI for application that draws a histogram
 4 ui <- fluidPage(
 5
 6     # Application title
 7     titlePanel("Simulation"),
 8
 9     # Sidebar with a slider input for number of bins
10     sidebarLayout(
11         sidebarPanel(
12             numericInput("N",
13                          "Sample Size", value=100),
14             downloadButton("download")
15
16         ),
17
18         # Show a plot of the generated distribution
```

Your Turn - Exercise 4

Copy the code from the previous slide (or open `R/demo6.R`) and run it in R

- Add logic to simulate data (e.g., using `rnorm` or `MASS::mvrnorm`)
- Add a plot to show the simulation results (e.g., a scatterplot)
- Add extra features to make the simulation interactive

Simulation Solution

R/solution6.R

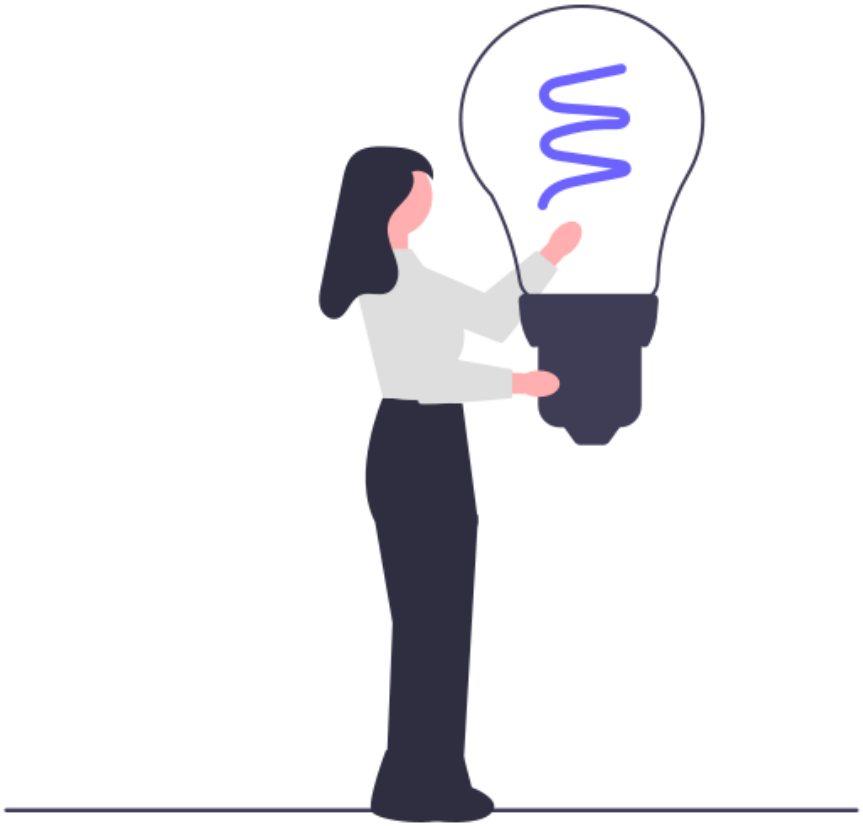
```r
library(shiny)

# Define UI for application
ui <- fluidPage(

    # Application title
    titlePanel("Simulation"),

    # Sidebar with a slider input for number of bins
    sidebarLayout(
        sidebarPanel(
            numericInput("N",
                         "Sample Size", value=100),
            numericInput("r",
                         "Correlation", value=0),
            downloadButton("download")

        ),
```

Inspiration

shiny.rstudio.com/gallery/

> The Shiny User Showcase is comprised of contributions from the Shiny app developer community.

Your turn - go wild!



License

Thanks

Thank you for being on this journey with us!

Andreas (find me on Twitter)

Leonie (find me on LinkedIn)