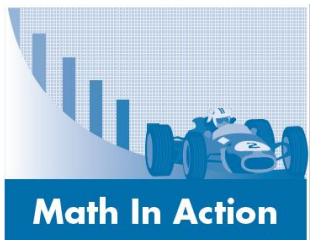
Create your own math apps

Daniel W. Adrian[1]

[1]Grand Valley State University
Department of Statistics

Math In Action
February 27, 2016



Outline

My website

`http://facweb.gvsu.edu/adriand1/mia.html`

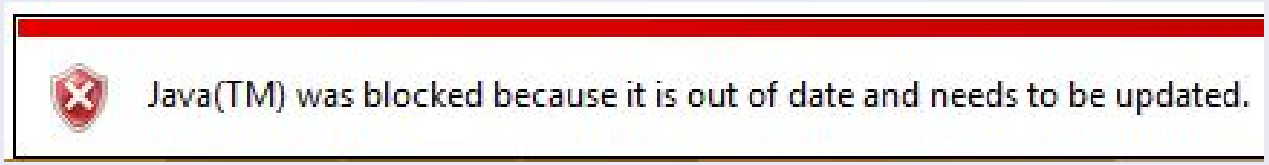
- PDF of these slides
- Links to apps
- Code to create apps

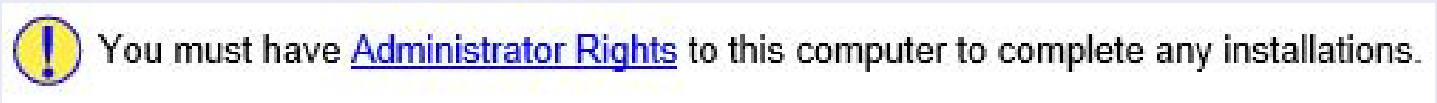
Note: All web addresses in this presentation are hyperlinks on the PDF.

Motivation

Problems with Java Applets

- Require updates or downloads





- Browsers no longer supporting Java Applets
  - Google Chrome (after version 45)
  - Microsoft Edge

The "Learning Curve"

Can't make your own without knowing

- Java
- HTML
- Javascript

"Shiny Apps" created by R

- Not based on Java Applets
- You don't need to know HTML or Javascript.

Written in R

- An "easier" programming language
- Free!
- Two Downloads:
  1. The Base System: `https://cran.rstudio.com/`
  2. User Interface: `https://www.rstudio.com/products/rstudio/#Desktop`
- Students do not need knowledge of R to use Shiny Apps.

Outline

Try them out!

- Login to your computer.
- Go to `http://facweb.gvsu.edu/adriand1/mia.html`
- Click on "shinyapps.io" or "GVSU server" next to the following apps:
  - Slope-intercept Form of a Line
  - The Unit Circle: sine and cosine functions (try the animation)
  - Equation of a circle
  - Equation of a parabola (vertex form)
  - Normal distribution

Discussion

- Shiny Apps provide for student learning that is
  - interactive
  - exploration-based
  - more fun
- Illustrate dynamic visual concepts very well

Question

How would YOU use them in your teaching?

Outline

No Free Lunch (except at MIA...)

- Shiny Apps need to be served by a computer (a server)
- Some options:

shinyapps.io: from Rstudio

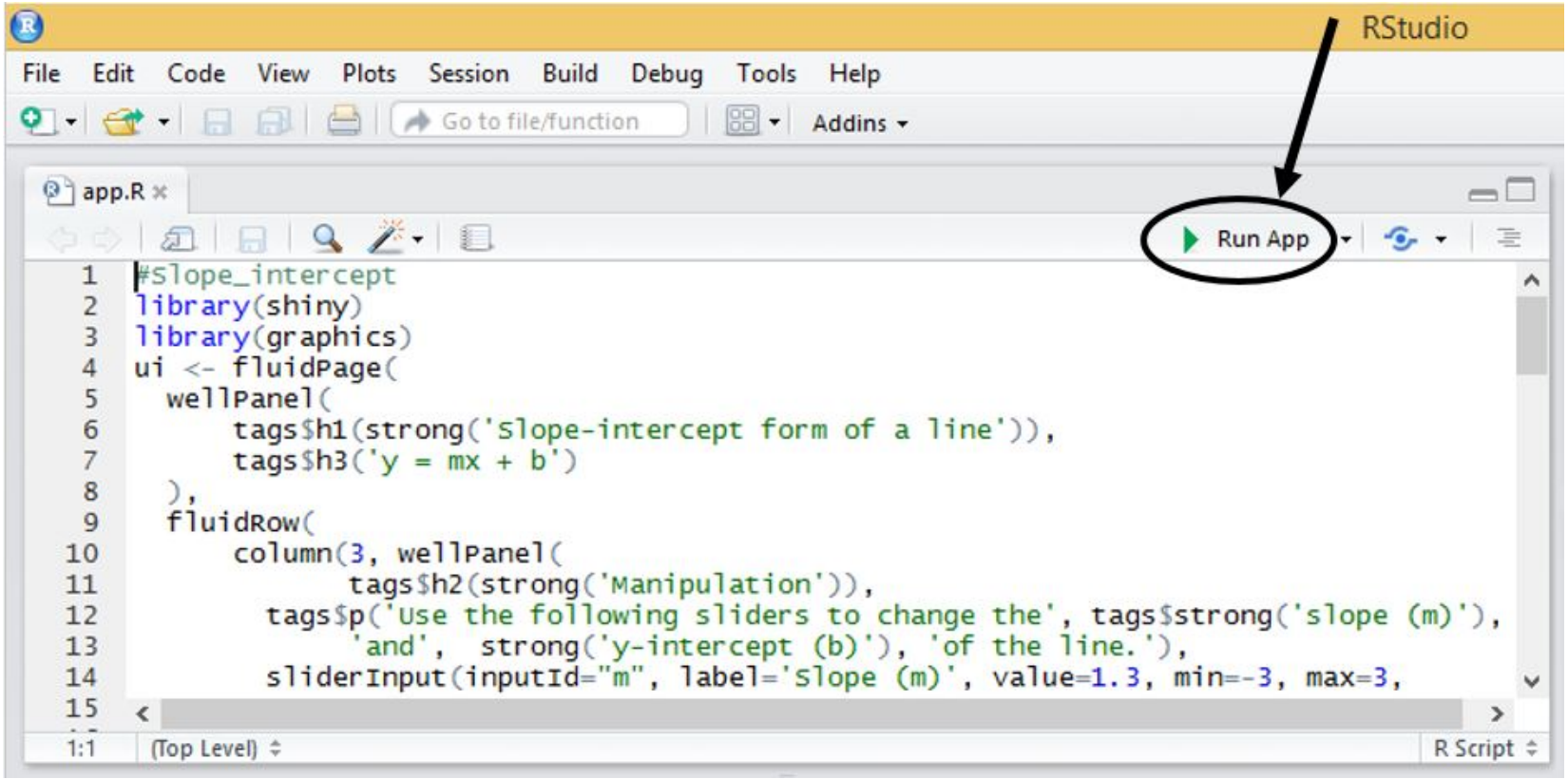
- Free Plan - Limitations:
  - Only 5 Apps
  - Only 25 active hours per month
- Other plans available for $$$.
- Info:
  `http://shiny.rstudio.com/articles/shinyapps.html`

Shiny Server: Linux

- Use Linux to create your own server.
- Not recommended unless you are an advanced Linux user.
- Info: `www.rstudio.com/products/shiny/shiny-server/`

Use R as a local server

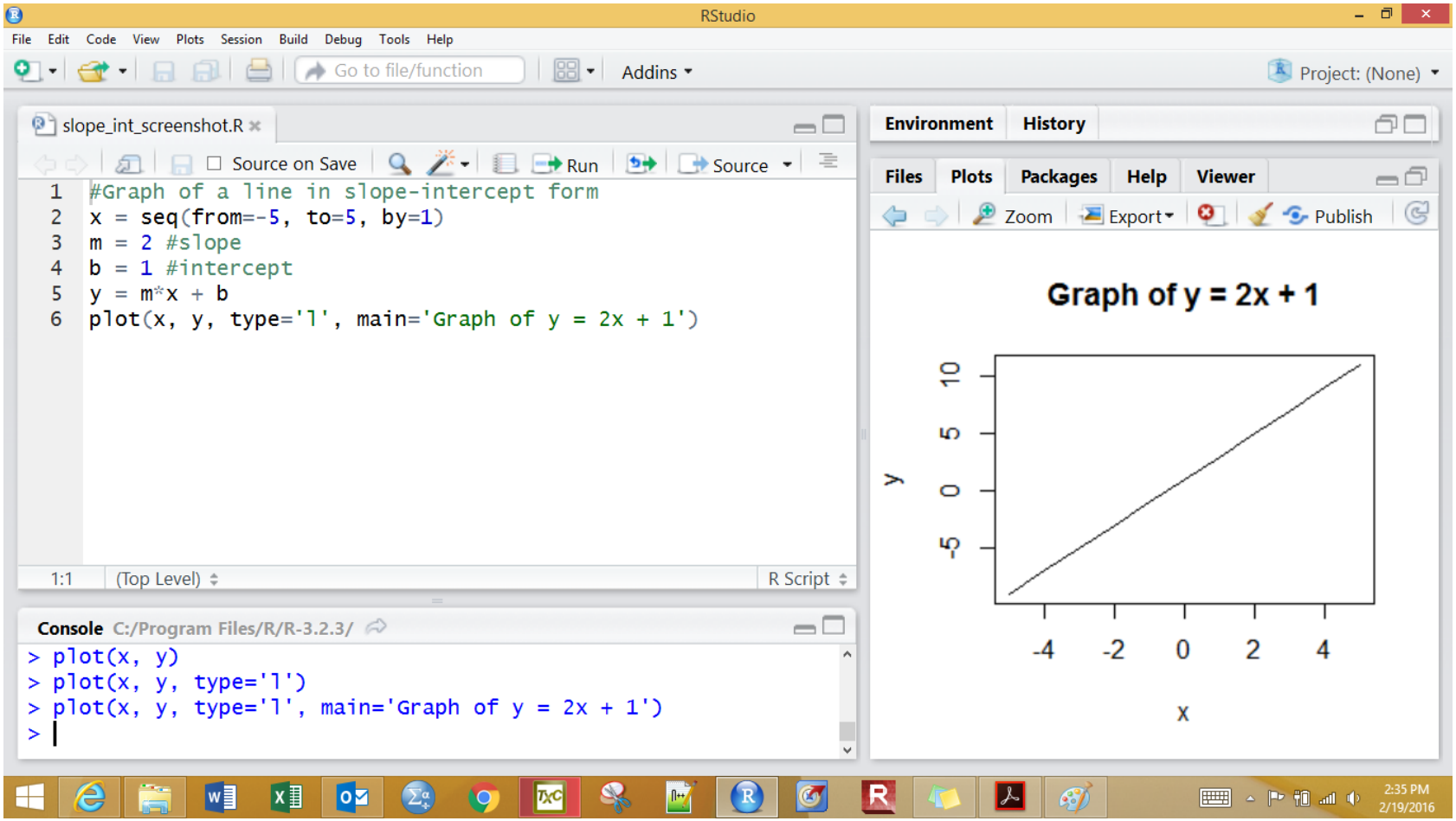
- Install R on school computers
- Save code from my website as the file `app.R`.
- Tell students to click on "Run App"



Outline

R Screenshot

- You can use R like a calculator
- Or you can write an R script (a program) – code for the app



Excellent Online Tutorial about Shiny Apps

- Website: `http://shiny.rstudio.com/tutorial/`
- I borrow from it in this presentation.

Two Parts of a Shiny App

User Interface (UI)

What is shown on the webpage

A computer (server) running R

Performs calculations to update the webpage

Outline

The Template

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- Every time you make a new app, you should start with this template.
- Available on webpage

library(shiny)

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- `library(shiny)` loads the package `shiny` into the current R session.
- Packages are extensions to the base R system.
- The package `shiny` must first be installed (Tools → Install Packages...)

ui <- fluidPage()

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

What goes between `()` on `ui <- fluidPage()` determines what is shown on the user interface (UI), i.e. webpage.

server <- function(input, output)

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

- Tells the server what to do to update the webpage.
- Takes `input` from the webpage (like slope and intercept from sliders)
- Produces `output` to the webpage (like updated graph of the line)

shinyApp(ui = ui, server = server)

```
library(shiny)
ui <- fluidPage()
server <- function(input, output) {}
shinyApp(ui = ui, server = server)
```

R does its "magic" to create the app from your code.

Outline

*Input() and *Output() functions

In general

- Go in the `ui <- fluidPage()` part.
- `*Input()`: input from user of webpage $\rightarrow$ server
- `*Output()`: output from server $\rightarrow$ user of webpage

My apps

- `sliderInput()`
- `plotOutput()`

Other *Input() functions

Buttons

Action

Submit

`actionButton()`
`submitButton()`

Single checkbox

Choice A

`checkboxInput()`

Checkbox group

Choice 1
Choice 2
Choice 3

`checkboxGroupInput()`

Date input

2014-01-01

`dateInput()`

Date range

2014-01-24 to 2014-01-24

`dateRangeInput()`

File input

Choose File No file chosen

`fileInput()`

Numeric input

1

`numericInput()`

Password Input

`passwordInput()`

Radio buttons

Choice 1
Choice 2
Choice 3

`radioButtons()`

Select box

Choice 1

`selectInput()`

Sliders

`sliderInput()`

Text input

Enter text...

`textInput()`

Other *Output() functions

Function	Inserts
`dataTableOutput()`	an interactive table
`htmlOutput()`	raw HTML
`imageOutput()`	image
`plotOutput()`	plot
`tableOutput()`	table
`textOutput()`	text
`uiOutput()`	a Shiny UI element
`verbatimTextOutput()`	text

For documentation (help): `?plotOutput`

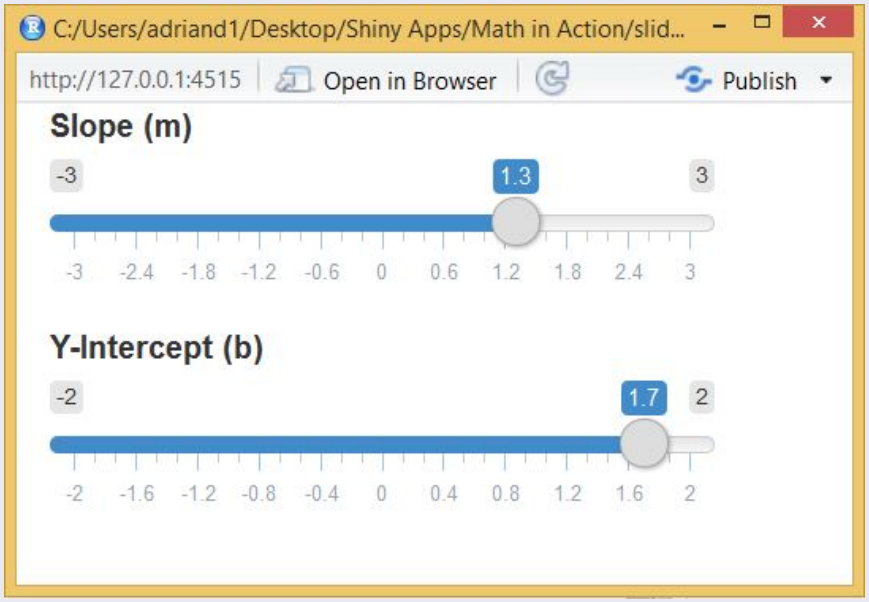
Outline

Example: Slope-intercept app

Code

```
ui <- fluidPage(
  sliderInput(inputId="m", label='Slope (m)',
              value=1.3, min=-3, max=3, step=.1),
  sliderInput(inputId="b", label='Y-Intercept (b)',
              value=1.7, min=-2, max=2, step=.1),
  plotOutput('myplot')
)
```

Webpage



Building the output (plot) in the `server` function

3 steps:

1. Save to `output$`
2. Use `render*()` functions to produce output – in this case, `renderPlot()`.
3. Incorporate inputs with `input$`

1. Save to output$

Code

```
ui <- fluidPage(
  sliderInput(inputId="m", label='Slope (m)',
              value=1.3, min=-3, max=3, step=.1),
  sliderInput(inputId="b", label='Y-Intercept (b)',
              value=1.7, min=-2, max=2, step=.1),
  plotOutput('myplot')
)

server <- function(input, output) {
  output$myplot <- #code
}
```

- Save to `output$`
- The name following `output$` needs to match the name in `plotOutput`.
- Here: `output$myplot` matches `plotOutput('myplot')`

2. Use render*()

Code

```
ui <- fluidPage(
  sliderInput(inputId="m", label='Slope (m)',
              value=1.3, min=-3, max=3, step=.1),
  sliderInput(inputId="b", label='Y-Intercept (b)',
              value=1.7, min=-2, max=2, step=.1),
  plotOutput('myplot')
)

server <- function(input, output) {
  output$myplot <- renderPlot({
    #code
  })
}
```

Other `render*()` functions:

- `renderTable()`
- `renderText()`

3. Use input$ to incorporate inputs

Code

```
ui <- fluidPage(
  sliderInput(inputId="m", label='Slope (m)',
              value=1.3, min=-3, max=3, step=.1),
  sliderInput(inputId="b", label='Y-Intercept (b)',
              value=1.7, min=-2, max=2, step=.1),
  plotOutput('myplot')
)

server <- function(input, output) {
  output$myplot <- renderPlot({
    x <- seq(from=-3, to=3, by=1)
    y <- input$m * x + input$b
    plot(x, y, type='l')
  })
}
```

Note:

- input$m matches sliderInput(inputId='m', ...)
- input$b matches sliderInput(inputId='b', ...)

Add to the plot

Code

```
server <- function(input, output) {
  output$myplot <- renderPlot({
    x <- seq(from=-3, to=3, by=1)
    y <- input$m * x + input$b
    plot(x, y, type='l', ylim=c(-3,3))
    abline(h=0)
    abline(v=0)
  })
}
```

- `ylim=c(-3,3)` fixes the y-axis limits (so we can see the effect of the slope)
- `abline(h=0)` and `abline(v=0)` add the x- and y-axes.

Resulting app (code on website – `example.R`)