What’s Possible with speechcollectr

Abbey L. Thomas

187th Meeting of the Acoustical Society of America (Virtual)

19 November 2024

Topics Covered in Today’s Talk

1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

Code: https://github.com/abbey-thomas/speechcollectr-demo2024

Topics Covered in Today’s Talk

1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

Code: https://github.com/abbey-thomas/speechcollectr-demo2024

What is *speechcollectr*?

- A fledgling R package for building experiment interfaces to collect speech production and perception data from participants

R Core Team. (2024). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

But… why??

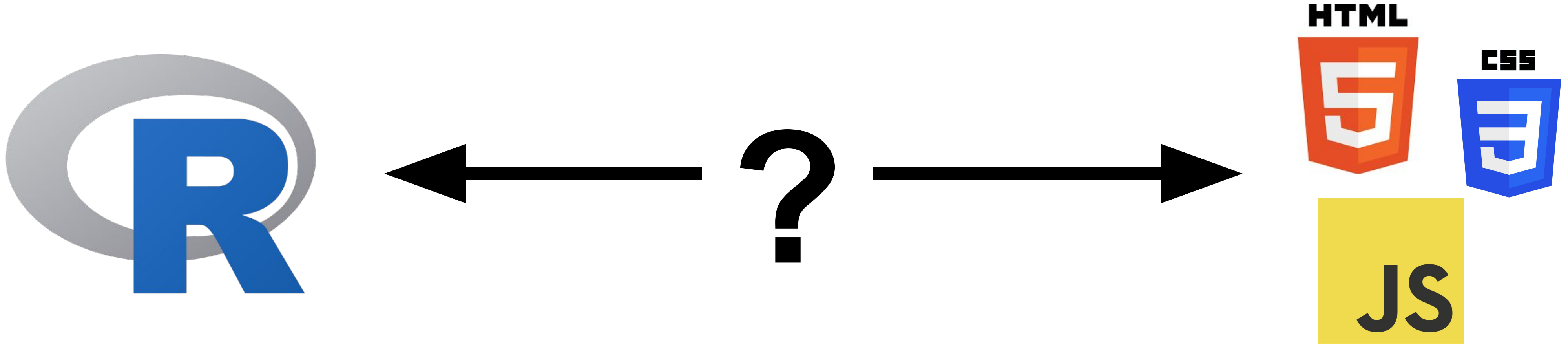
But… why??

- Allows those familiar with R to build on existing skills without the need to learn a new programming language or acquire additional software

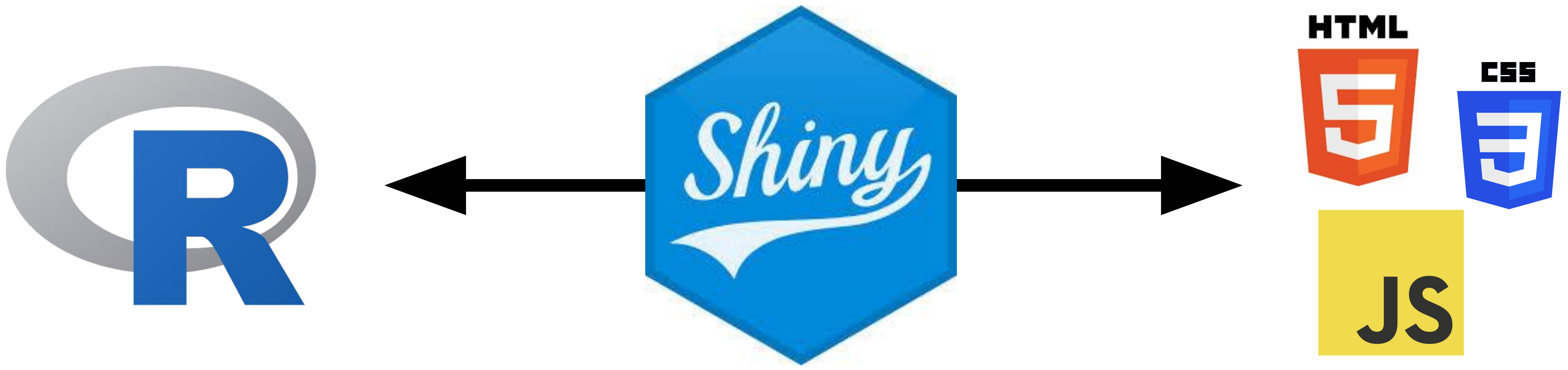
But… why??

- Allows those familiar with R to build on existing skills without the need to learn a new programming language or acquire additional software
- Experiments can be run locally on all major operating systems or distributed over the internet.

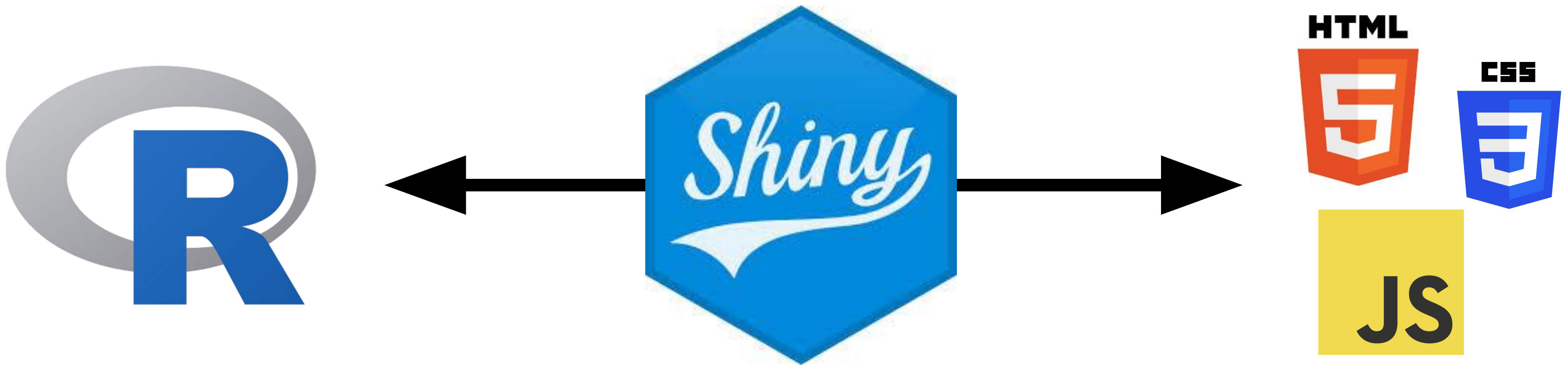
Programming for the Internet



Programming for the Internet

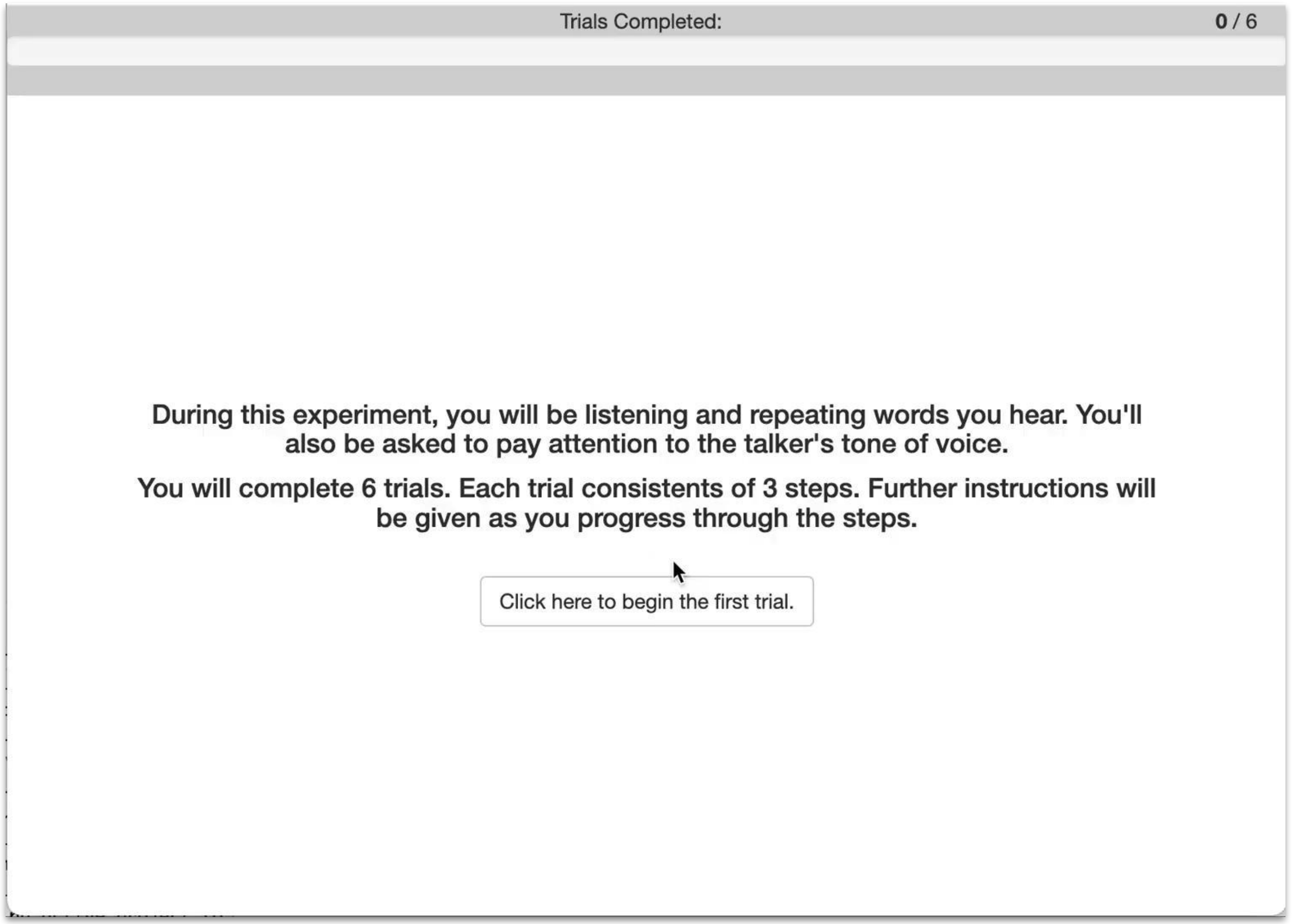


Programming for the Internet



+ speechcollectr

A Sample Experiment


Trials Completed: **0** / 6

During this experiment, you will be listening and repeating words you hear. You'll also be asked to pay attention to the talker's tone of voice.

You will complete 6 trials. Each trial consistents of 3 steps. Further instructions will be given as you progress through the steps.

Code: https://github.com/abbey-thomas/speechcollectr-demo2024/blob/main/full_app/app.R

Process > Package

A contribution perhaps more important than the package itself is the **development of methods for collecting speech data with R** and the **demonstration of the feasibility** of doing so.

Process > Package

A contribution perhaps more important than the package itself is the **development of methods for collecting speech data with R** and the **demonstration of the feasibility** of doing so.

NOTE: *speechcollectr is not intended include all functions experimenters could possibly ever need. This presentation offers some minimal tools and examples, intended to encourage YOU to build the experiment interfaces YOU want.*

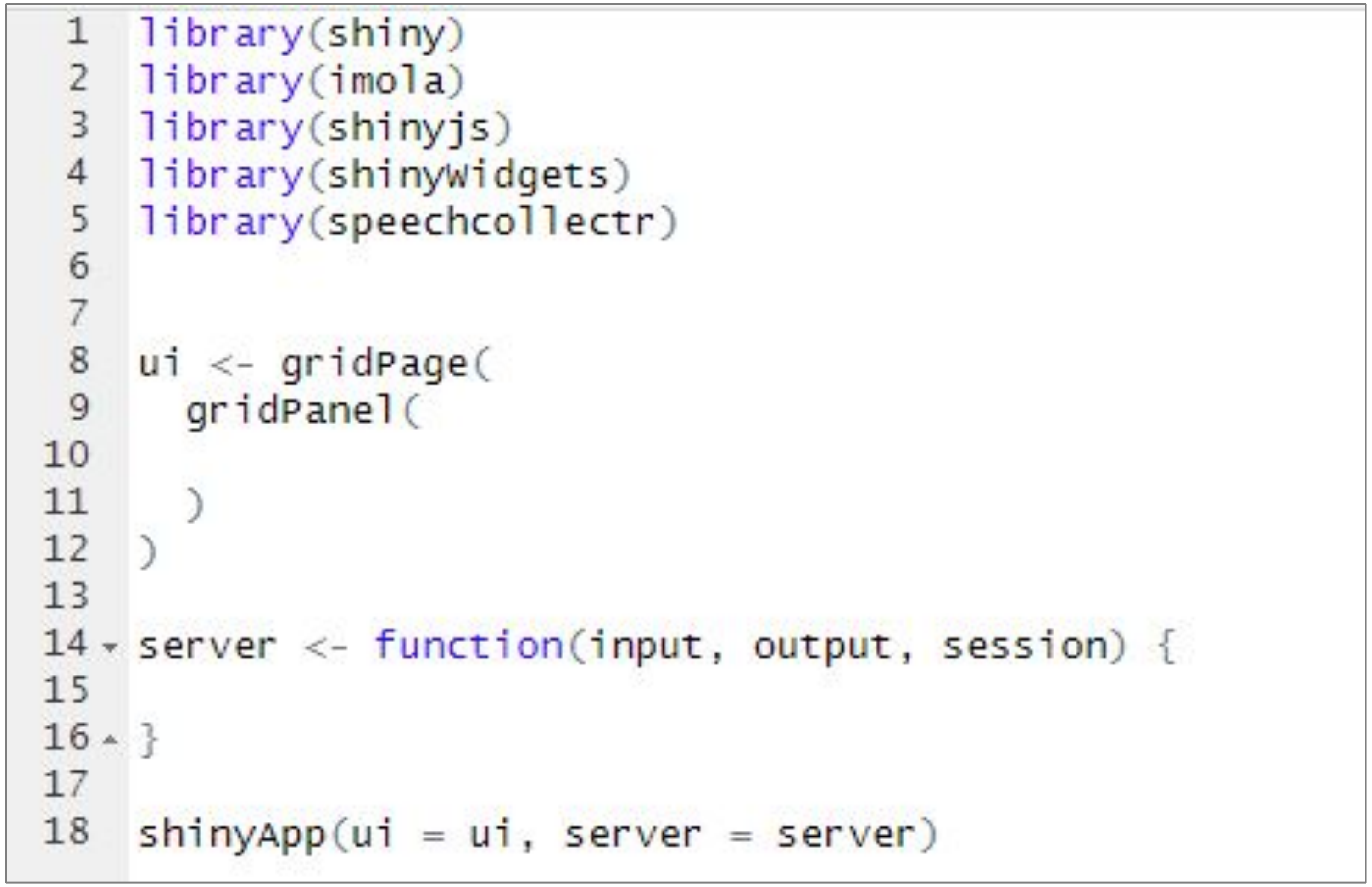
Topics Covered in Today’s Talk

1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

Basic Code for All Experiments

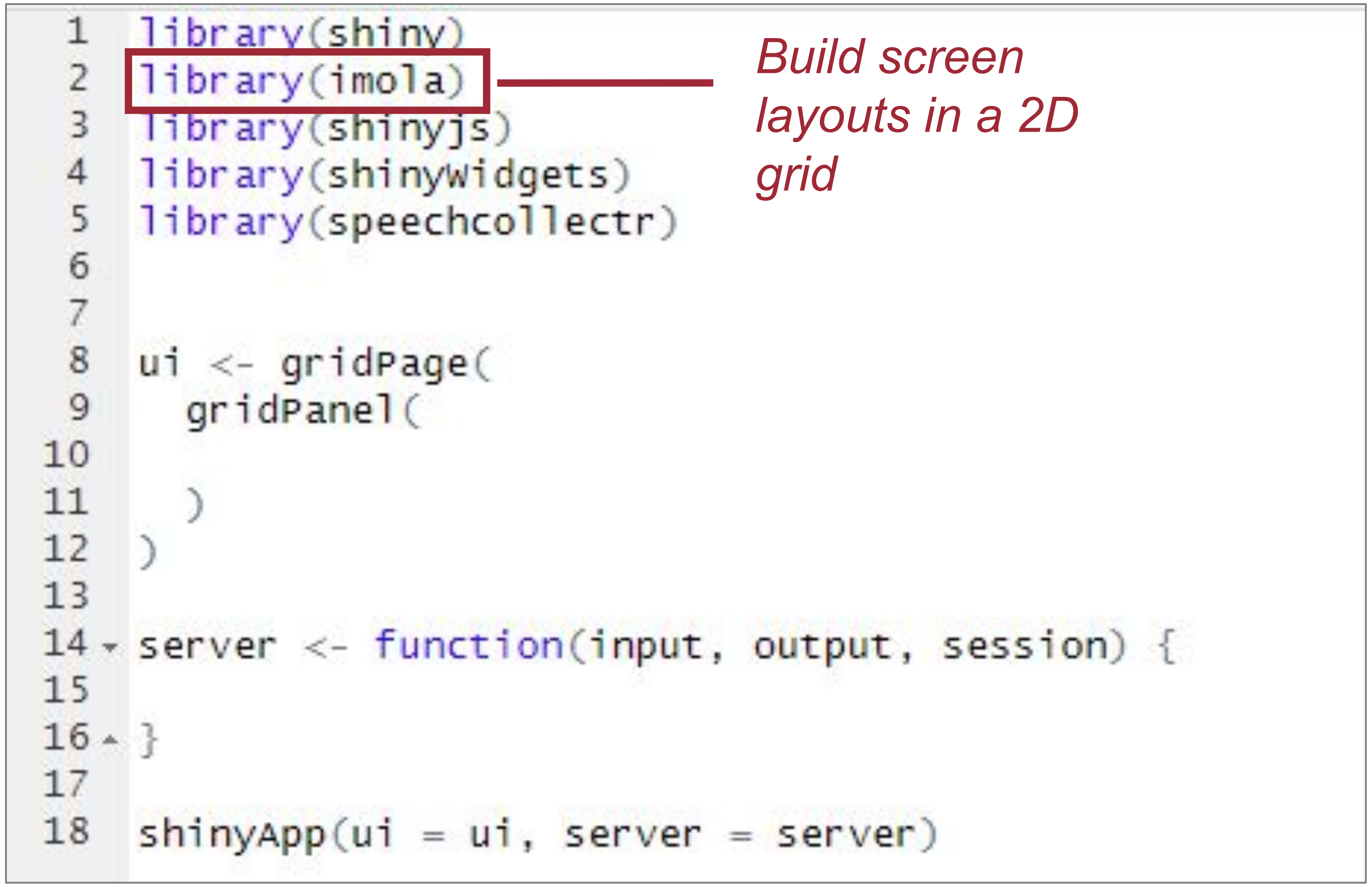
- best to start in a new working directory containing:
  - a subfolder called www
  - a file called app.R

Basic Code for All Experiments



```r
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

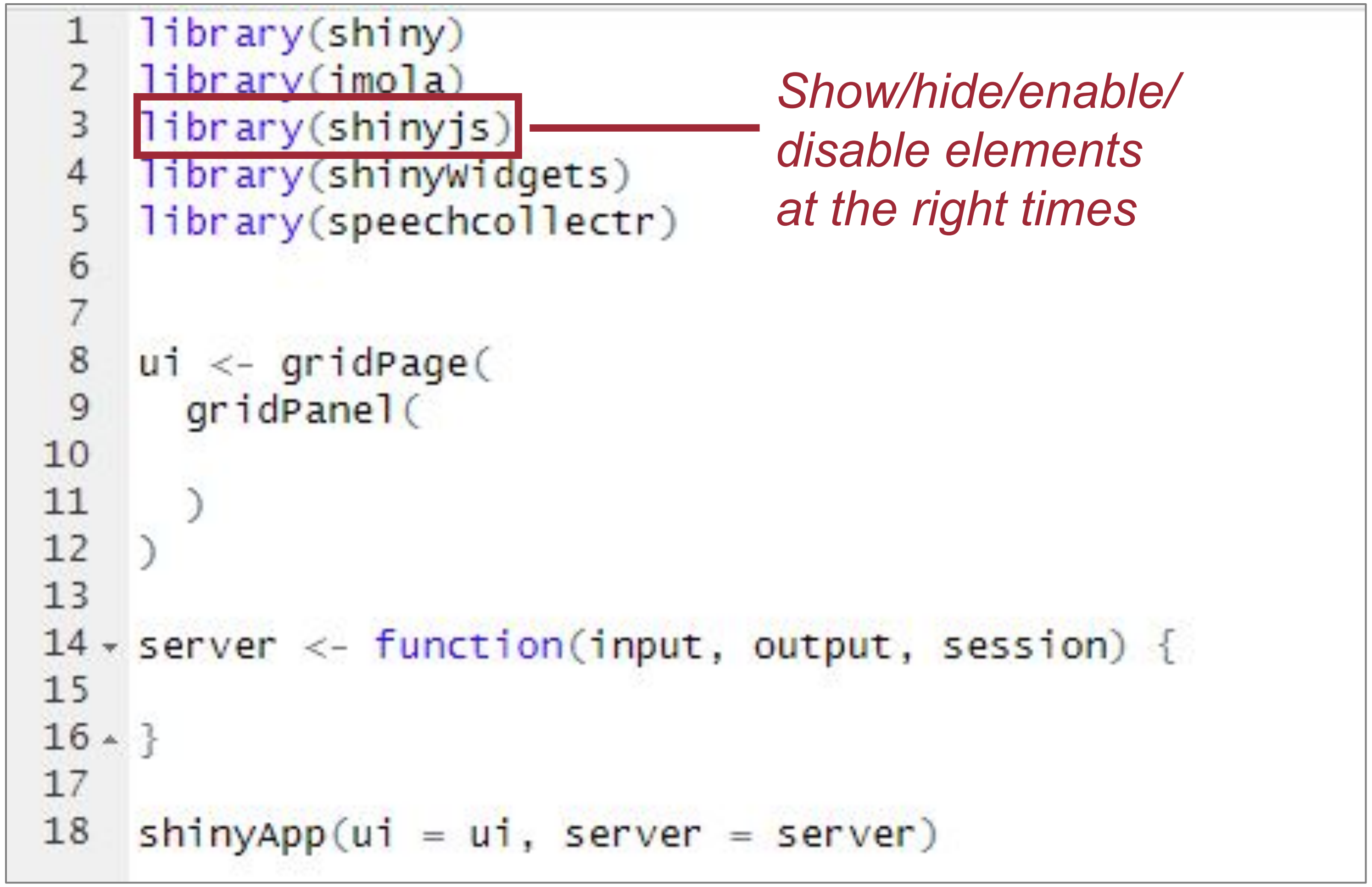
Basic Code for All Experiments



```r
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

Build screen layouts in a 2D grid

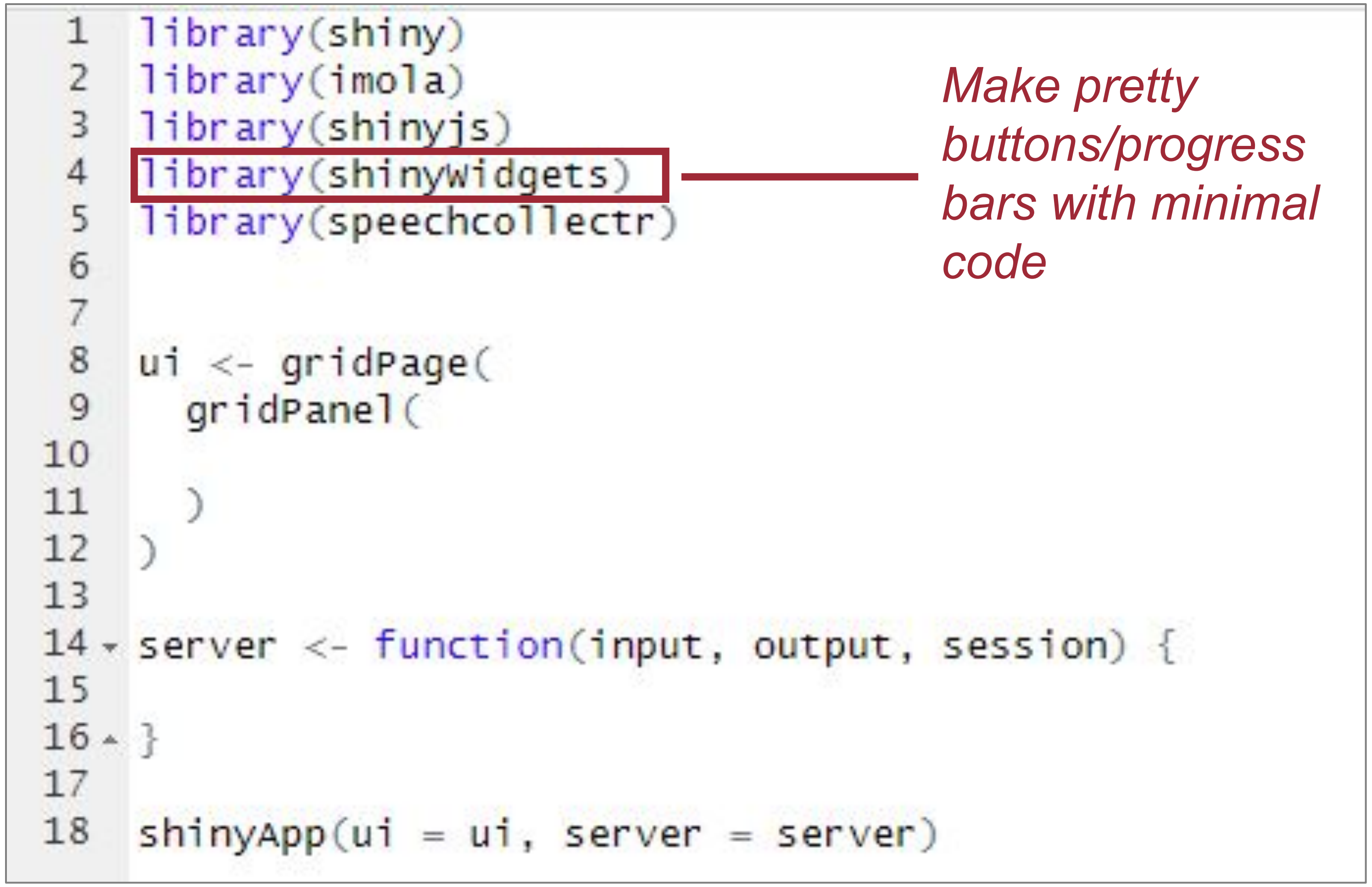
Basic Code for All Experiments



```r
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

Show/hide/enable/ disable elements at the right times

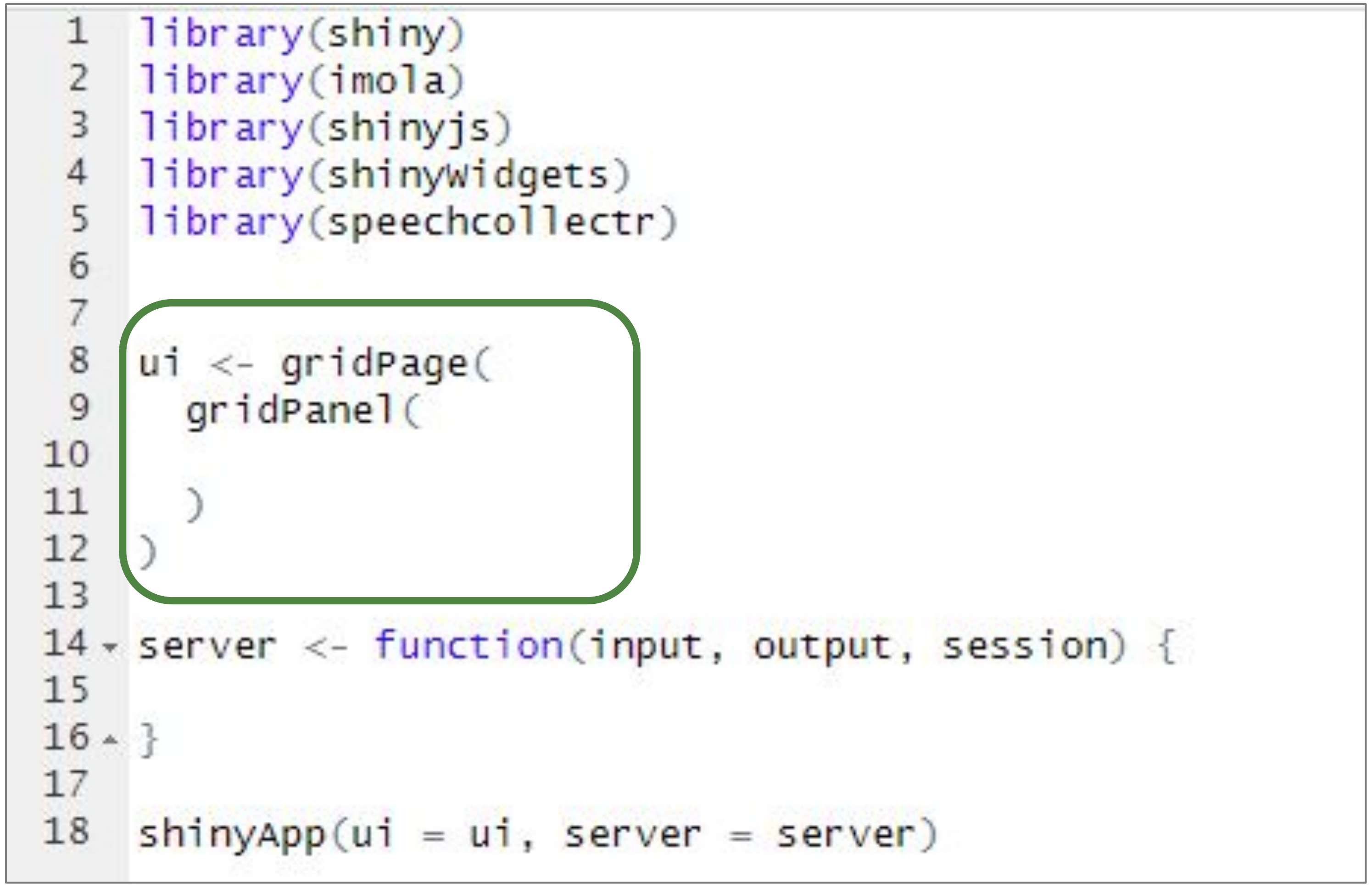
Basic Code for All Experiments



```r
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

Make pretty buttons/progress bars with minimal code

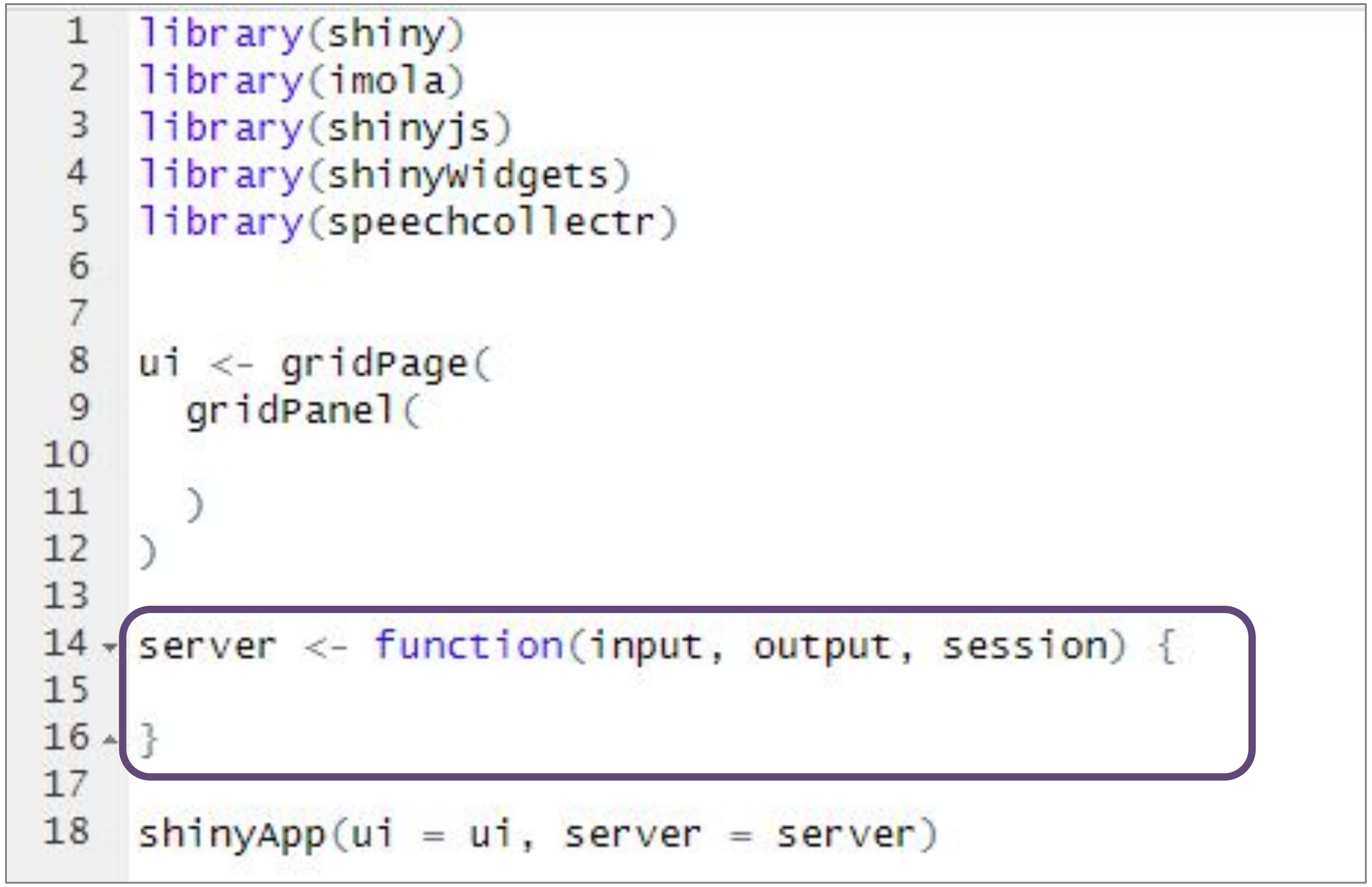
The ui Object



```
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

- The ui object contains code for the **static** elements of the experiment interface.
- Order of commands defines the order of elements on the screen.
- The page layout will be constructed on a 2D grid

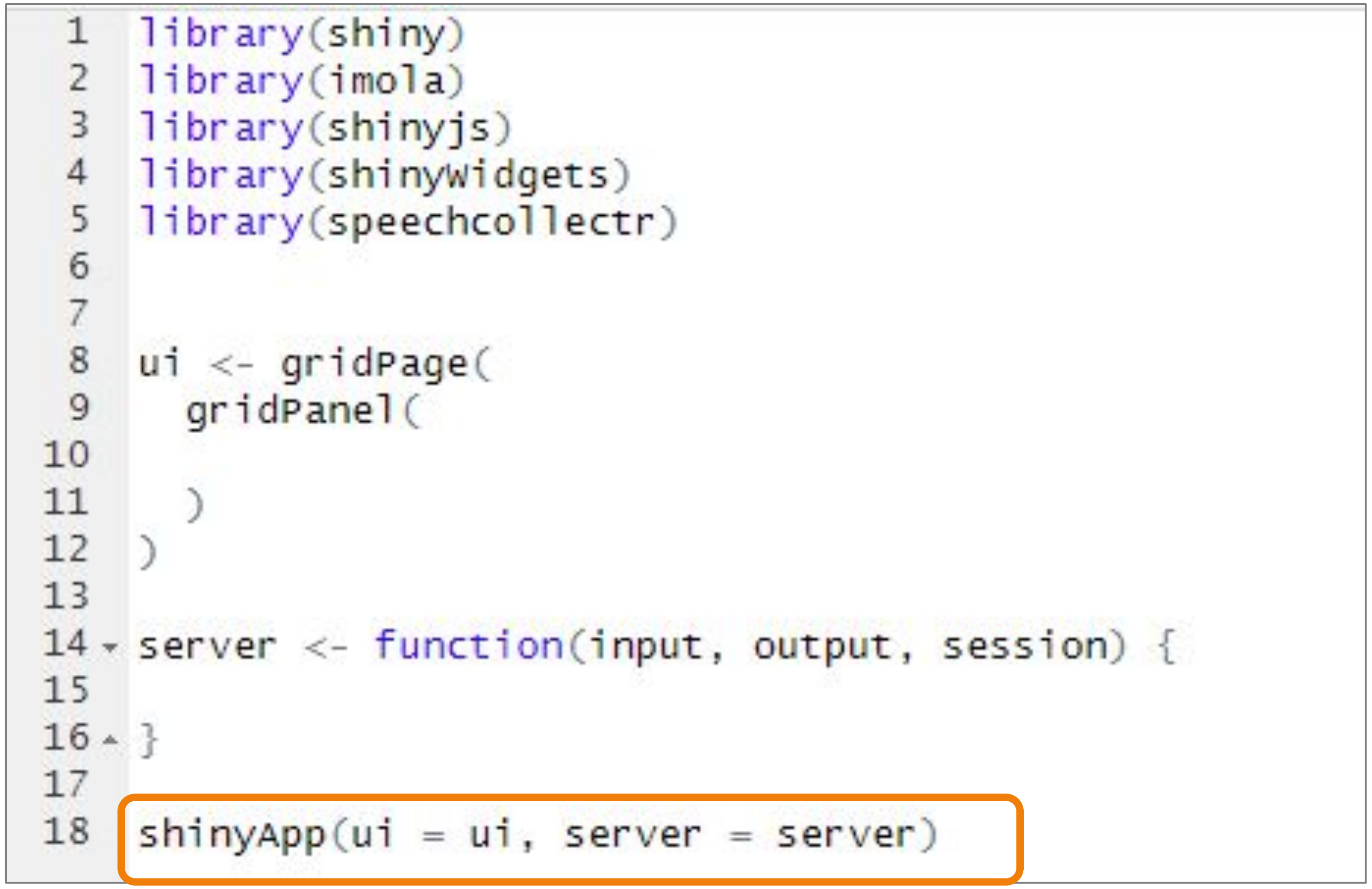
The server function



```r
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

- The server function makes the elements programmed in ui **reactive**.
- Tells R what to do when users interact with elements in the ui
- Keeps track of values that change during the experiment
- MUST have the **input**, **output**, & **session** arguments

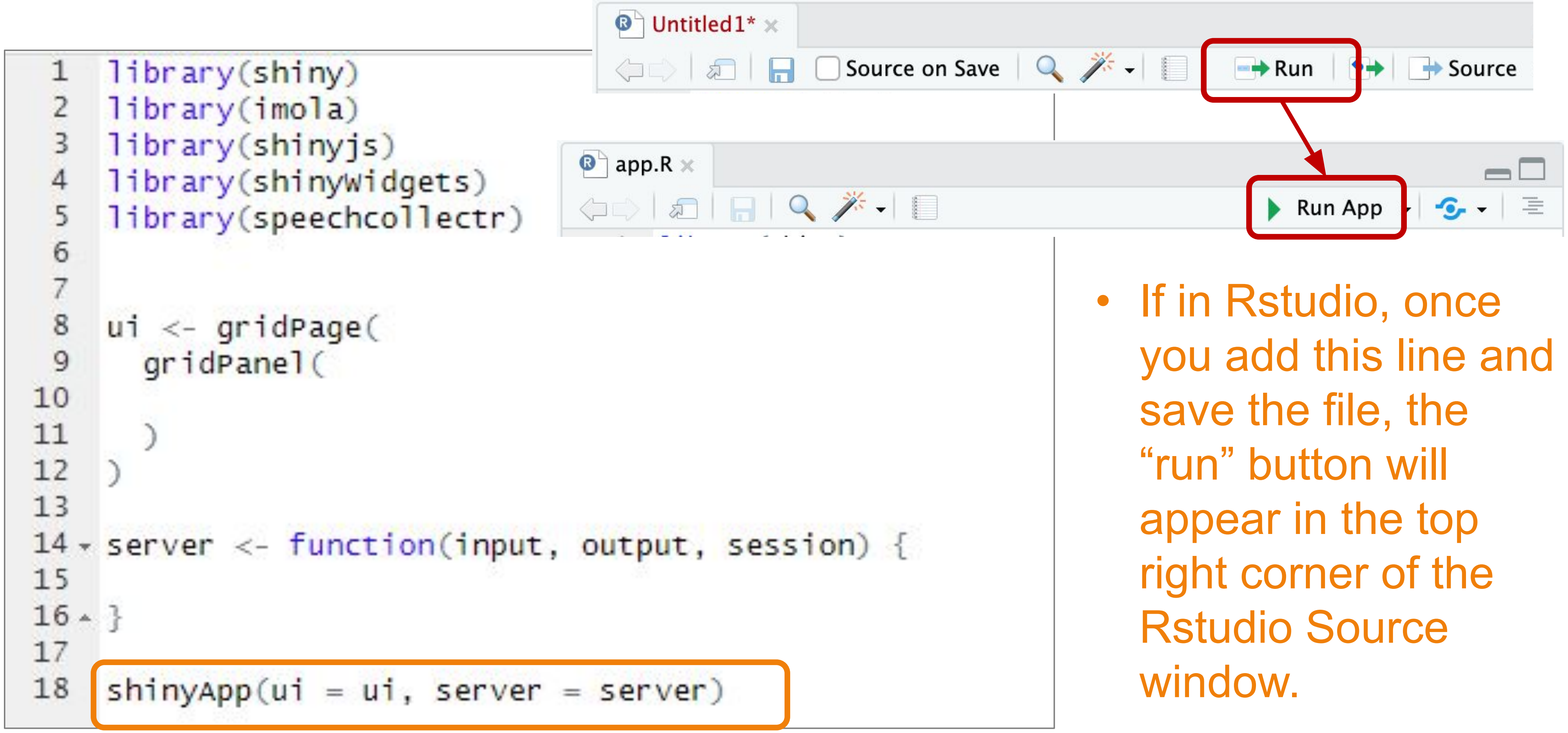
Run the Experiment



```
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

- Let’s R know this code is an application
- Application is rendered in a local browser window

Run the Experiment



```
library(shiny)
library(imola)
library(shinyjs)
library(shinyWidgets)
library(speechcollectr)


ui <- gridPage(
  gridPanel(

  )
)

server <- function(input, output, session) {

}

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

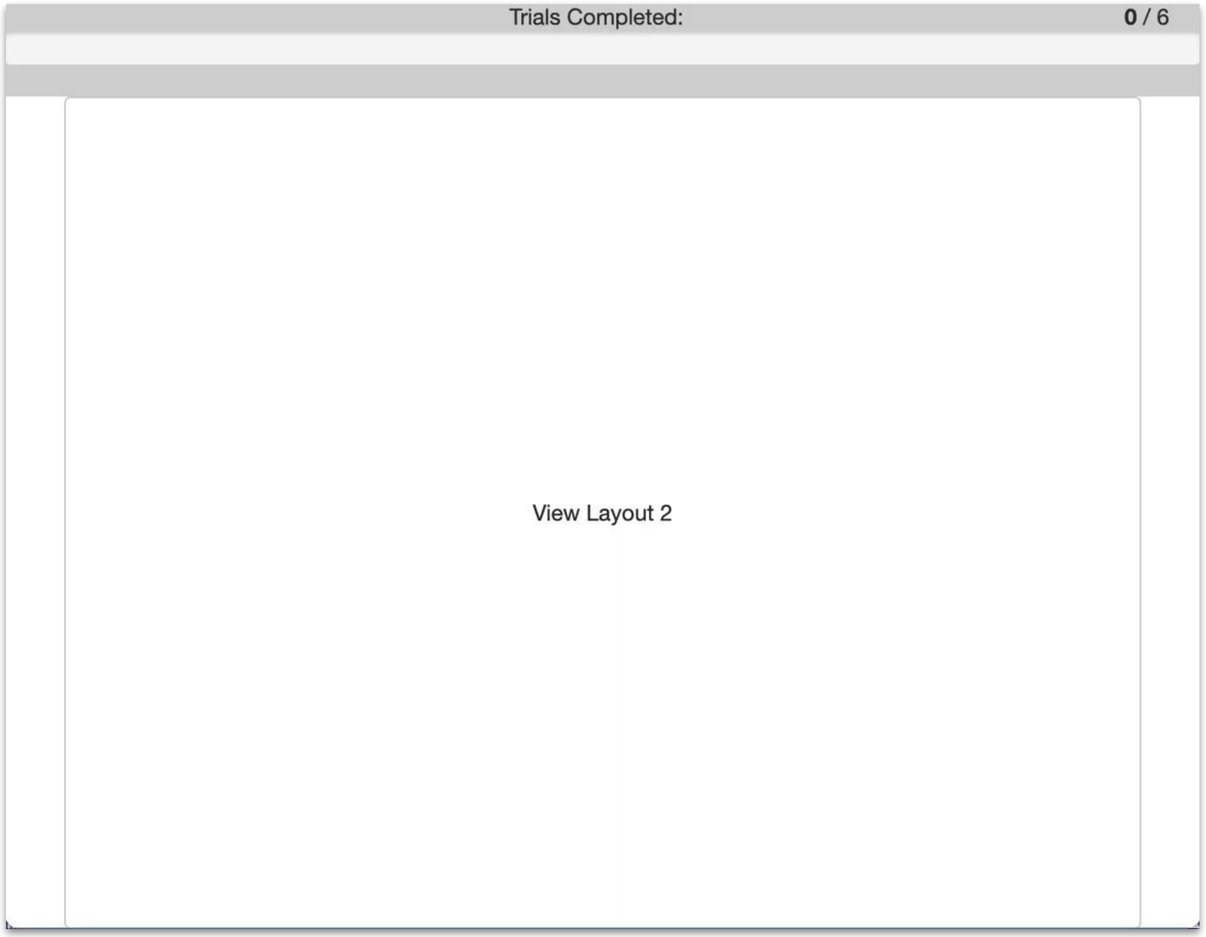
- If in Rstudio, once you add this line and save the file, the "run" button will appear in the top right corner of the Rstudio Source window.

Topics Covered in Today’s Talk

1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

Some Example ui Layouts

3 calls to gridPanel() inside a single call to gridPage()



Code: https://github.com/abbey-thomas/speechcollectr-demo2024/blob/main/app2.R

Topics Covered in Today's Talk

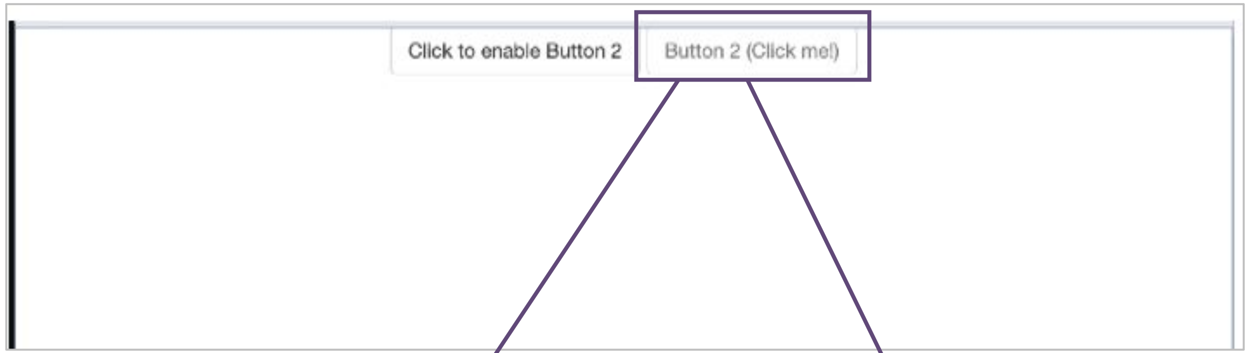
1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

From ui to server

- How do we make graphical elements react to user inputs?

Code: https://github.com/abbey-thomas/speechcollectr-demo2024/blob/main/app3.R

Using InputId and id



ui

```
div(id = "view1",
    actionButton(inputId = "button1",
                 label = "Click to enable Button 2"),

    disabled(actionButton(inputId = "button2",
                          label = "Button 2 (Click me!)"))),
```

server

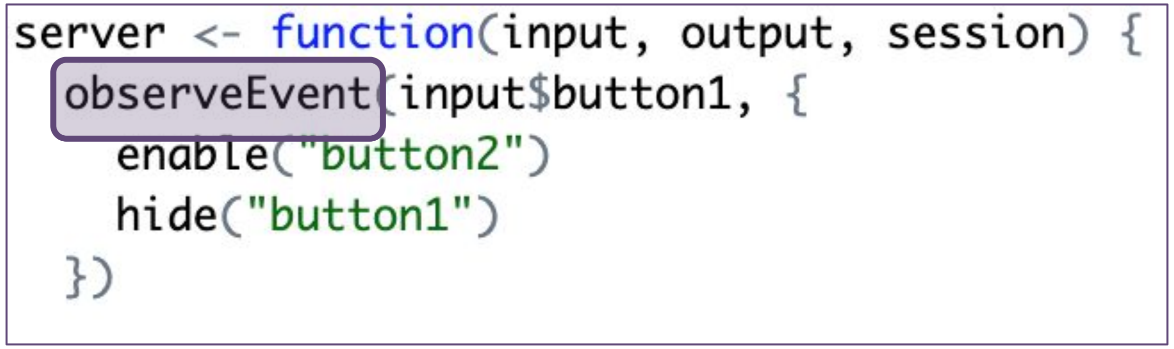
```
enable("button2")
hide("button1")
```

Reacting at the Right Time

- the server code = a chain of **EVENTS** like…
    - a change in the current trial number
    - a click on a button
    - an audio recording
    - an entry of some text in a text field

Reacting at the Right Time

- the server code = a chain of **EVENTS**

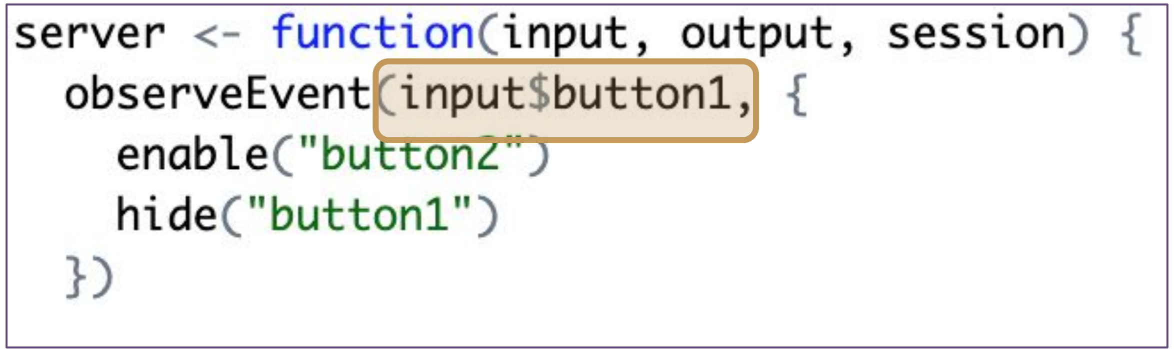


```
server <- function(input, output, session) {
  observeEvent(input$button1, {
    enable("button2")
    hide("button1")
  })
```

Reacting at the Right Time

- the server code = a chain of **EVENTS**

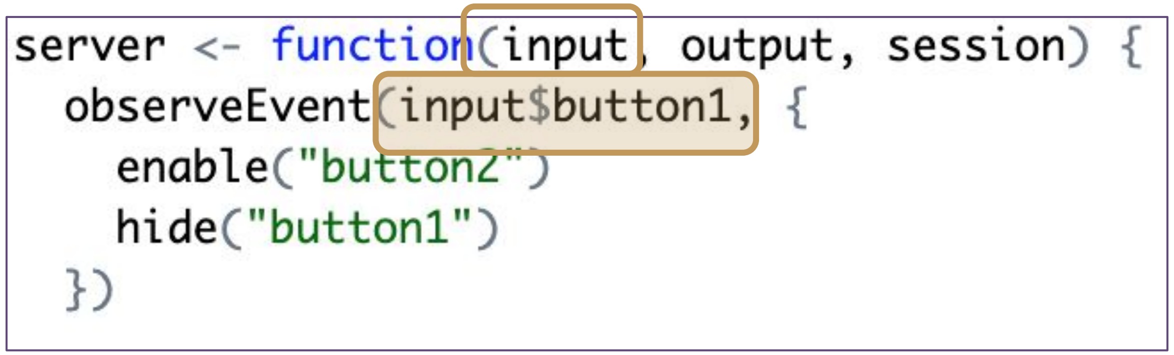
The event we're looking for


```
server <- function(input, output, session) {
  observeEvent(input$button1, {
    enable("button2")
    hide("button1")
  })
```

Reacting at the Right Time

- the server code = a chain of **EVENTS**

The event we're looking for

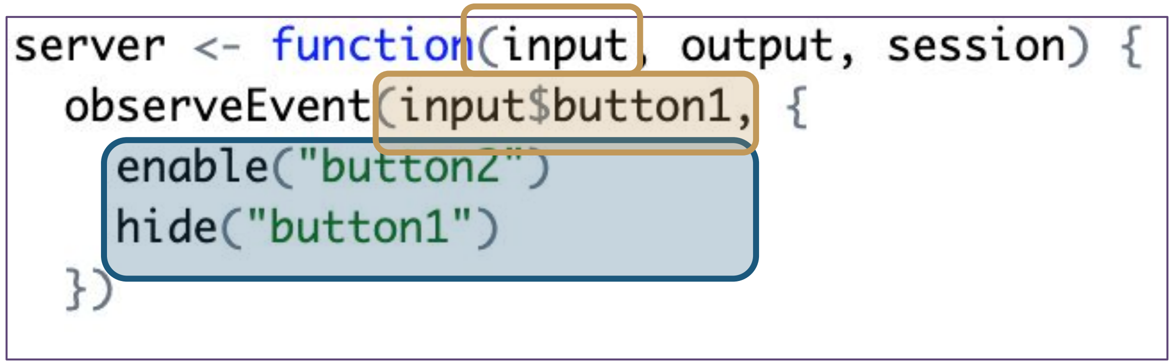


```
server <- function(input, output, session) {
  observeEvent(input$button1, {
    enable("button2")
    hide("button1")
  })
```

Reacting at the Right Time

- the server code = a chain of **EVENTS**

The event we're looking for



```
server <- function(input, output, session) {
  observeEvent(input$button1, {
    enable("button2")
    hide("button1")
  })
```

What will happen in response

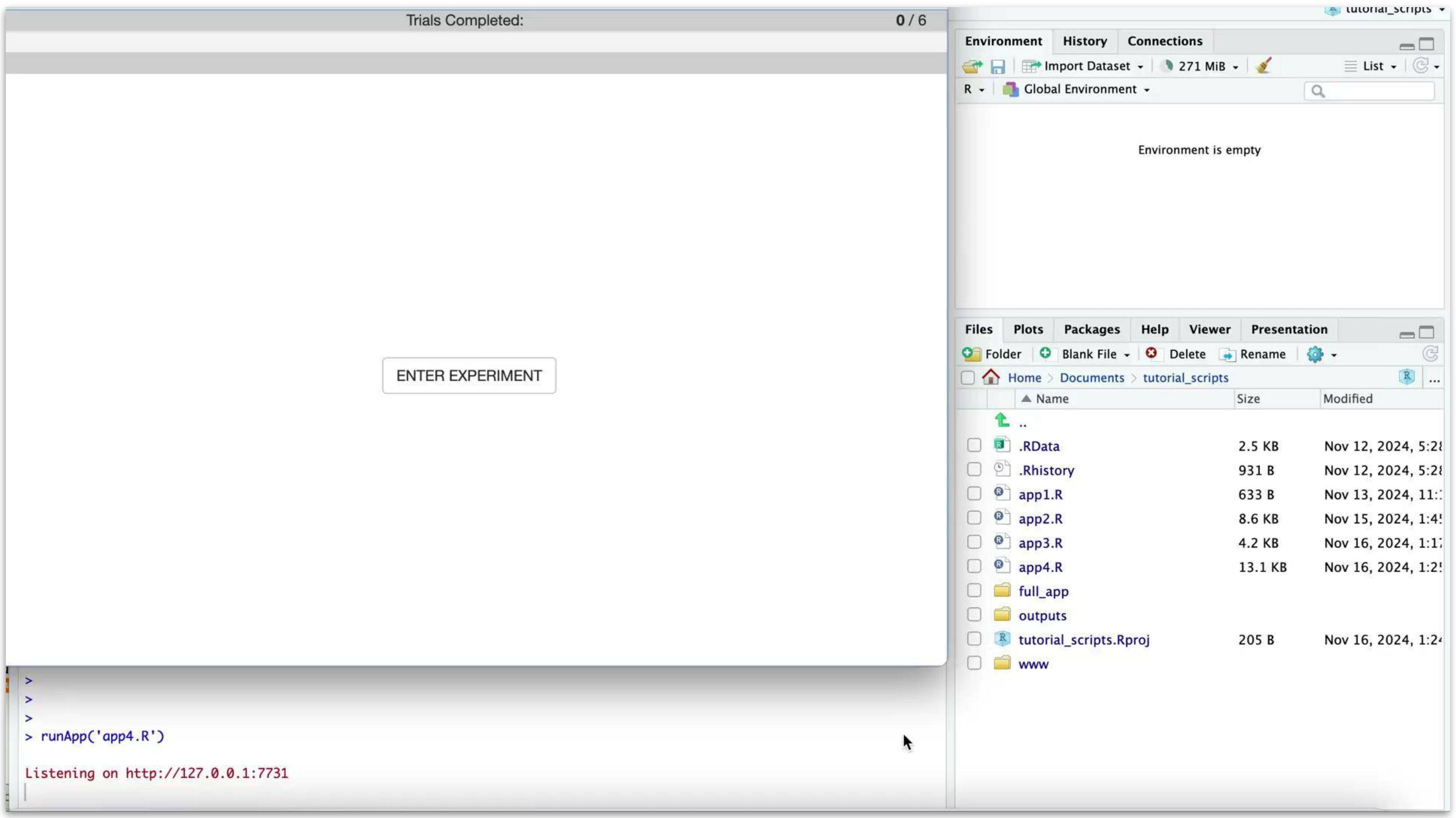
Topics Covered in Today's Talk

1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
   (especially audio recording)
6. speechcollectr modules for common experimental tasks

More server-side tasks

- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including audio files)
- Recording reaction times

More server-side tasks



```
>
>
>
> runApp('app4.R')

Listening on http://127.0.0.1:7731
```

Name	Size	Modified
..		
.RData	2.5 KB	Nov 12, 2024, 5:2
.Rhistory	931 B	Nov 12, 2024, 5:2
app1.R	633 B	Nov 13, 2024, 11:
app2.R	8.6 KB	Nov 15, 2024, 1:4
app3.R	4.2 KB	Nov 16, 2024, 1:1
app4.R	13.1 KB	Nov 16, 2024, 1:2
full_app		
outputs		
tutorial_scripts.Rproj	205 B	Nov 16, 2024, 1:2
www		

Code: https://github.com/abbey-thomas/speechcollectr-demo2024/blob/main/app4.R

More server-side tasks

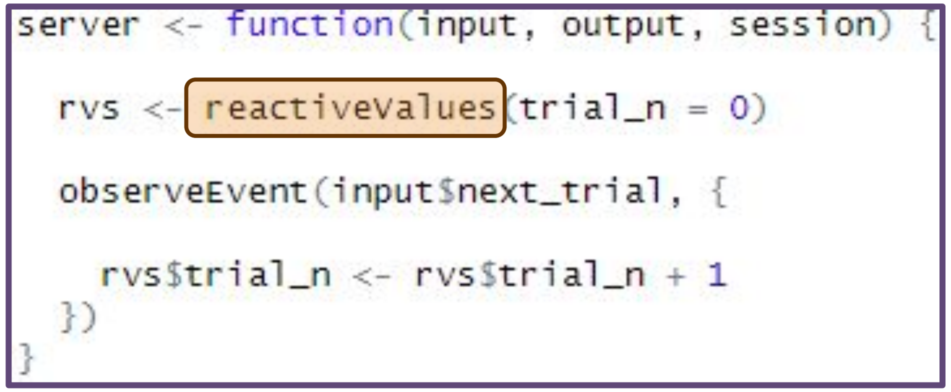
- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including audio files)
- Recording reaction times

Brief Introduction to Reactive Values & Contexts

- Reactive values: values that get updated in response to user inputs

Brief Introduction to Reactive Values & Contexts

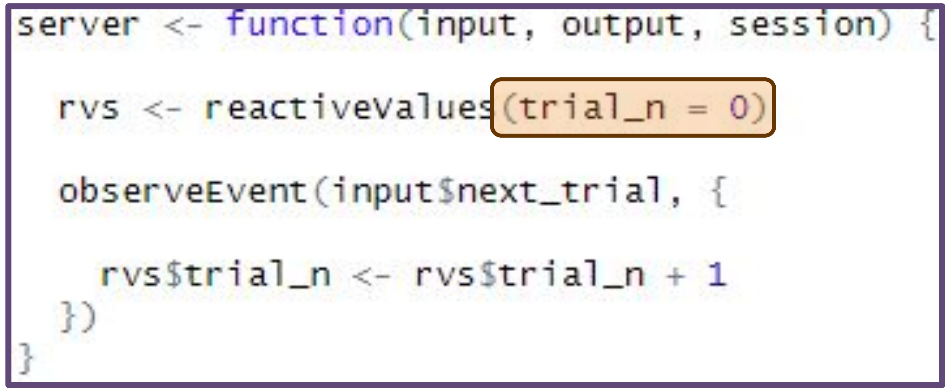
- Reactive values: values that get updated in response to user inputs
    - Can be created in the server function several ways, but an easy method is to use reactiveValues() in the server to create an object to hold reactive values.



```
server <- function(input, output, session) {

  rvs <- reactiveValues(trial_n = 0)

  observeEvent(input$next_trial, {

    rvs$trial_n <- rvs$trial_n + 1
  })
}
```

Brief Introduction to Reactive Values & Contexts

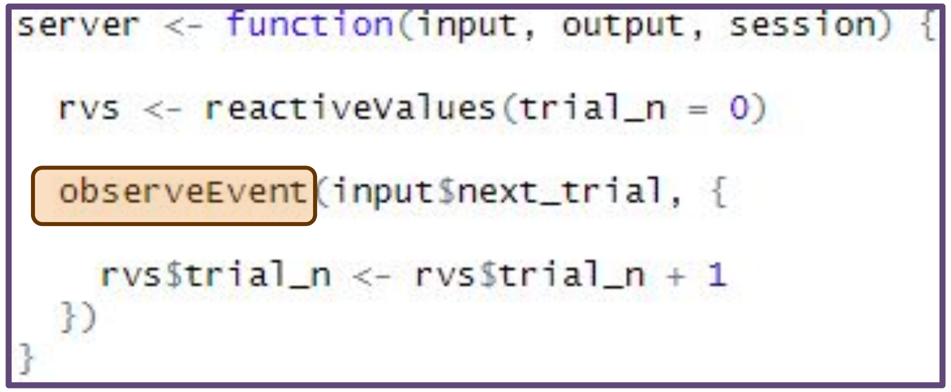
- Reactive values: values that get updated in response to user inputs
  - Set trial number as reactive value in the object rvs
  - The trial number has an initial value of 0.



```
server <- function(input, output, session) {

  rvs <- reactiveValues(trial_n = 0)

  observeEvent(input$next_trial, {

    rvs$trial_n <- rvs$trial_n + 1
  })
}
```

Brief Introduction to Reactive Values & Contexts

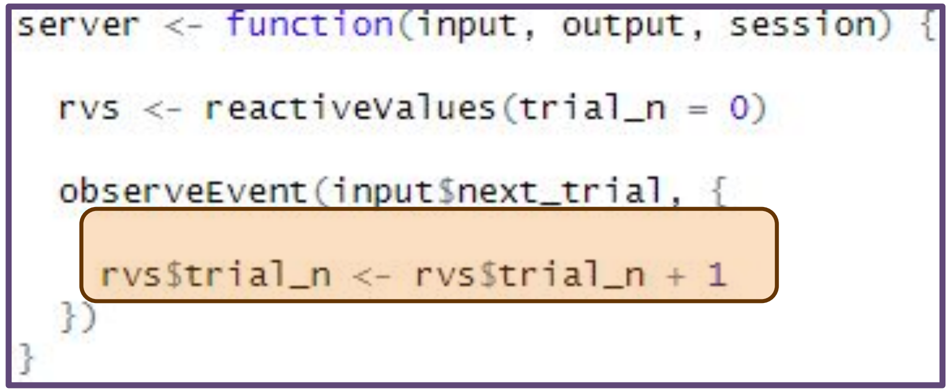
- Reactive values: values that get updated in response to user inputs
- Reactive contexts: contexts in which reactive values can be accessed or updated
    - Created with the functions **reactiveValues**() and **observeEvent**() in this snippet



```
server <- function(input, output, session) {

  rvs <- reactiveValues(trial_n = 0)

  observeEvent(input$next_trial, {

    rvs$trial_n <- rvs$trial_n + 1
  })
}
```

Brief Introduction to Reactive Values & Contexts

- Reactive values: values that get updated in response to user inputs
- Reactive contexts: contexts in which reactive values can be accessed or updated
  - Created with the functions **reactiveValues**() and **observeEvent**() in this snippet



```
server <- function(input, output, session) {

  rvs <- reactiveValues(trial_n = 0)

  observeEvent(input$next_trial, {

    rvs$trial_n <- rvs$trial_n + 1
  })
}
```

More server-side tasks: speechcollectr contributions

- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including recorded audio files)
- Recording reaction times

More server-side tasks: speechcollectr contributions

- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including recorded audio files)
- Recording reaction times

The pinGen() function

- **GEN**erates unique **PIN**s for participants
- If it doesn't exist, creates a file in the www folder called "pinlist.rds" = a list of already used PINs
- Returns a numeric code with the number of digits specified
- Generated PINs can be random or ordered according to the time of generation

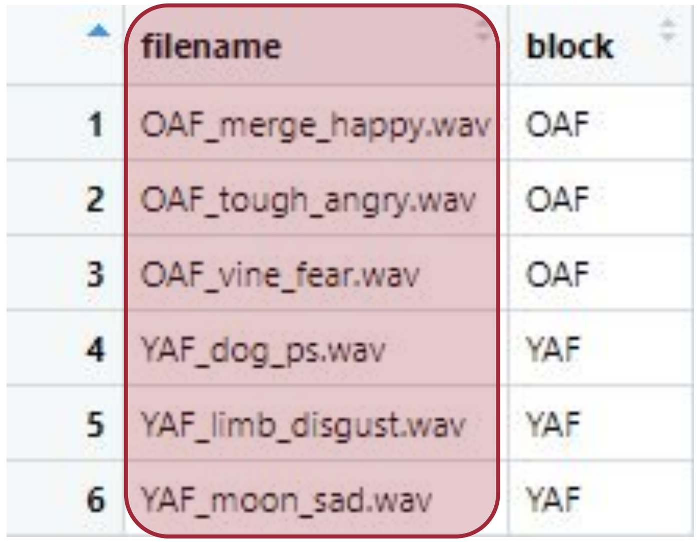
(NOTE: The name of this function rhymes if you're from Texas)

More server-side tasks: speechcollectr contributions

- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including recorded audio files)
- Recording reaction times

The randomStim() function

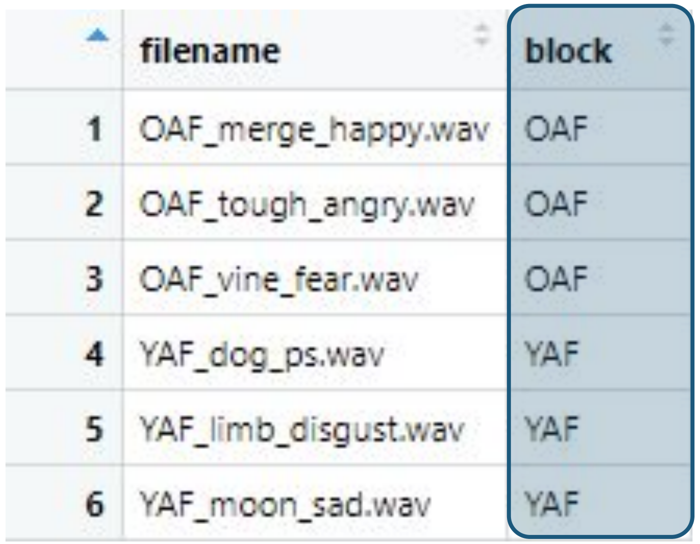
- Takes a dataframe or CSV table containing
  - a **column of the stimuli** to be randomized, one stimulus per row



	filename	block
1	OAF_merge_happy.wav	OAF
2	OAF_tough_angry.wav	OAF
3	OAF_vine_fear.wav	OAF
4	YAF_dog_ps.wav	YAF
5	YAF_limb_disgust.wav	YAF
6	YAF_moon_sad.wav	YAF

The randomStim() function

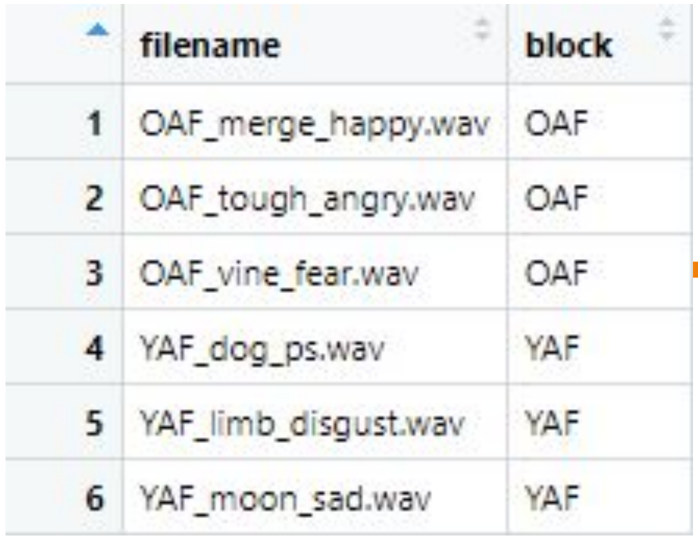
- Takes a dataframe or CSV table containing
  - a **column of the stimuli** to be randomized, one stimulus per row
  - (optionally) a **column defining which block** a stimulus should be presented in



	filename	block
1	OAF_merge_happy.wav	OAF
2	OAF_tough_angry.wav	OAF
3	OAF_vine_fear.wav	OAF
4	YAF_dog_ps.wav	YAF
5	YAF_limb_disgust.wav	YAF
6	YAF_moon_sad.wav	YAF

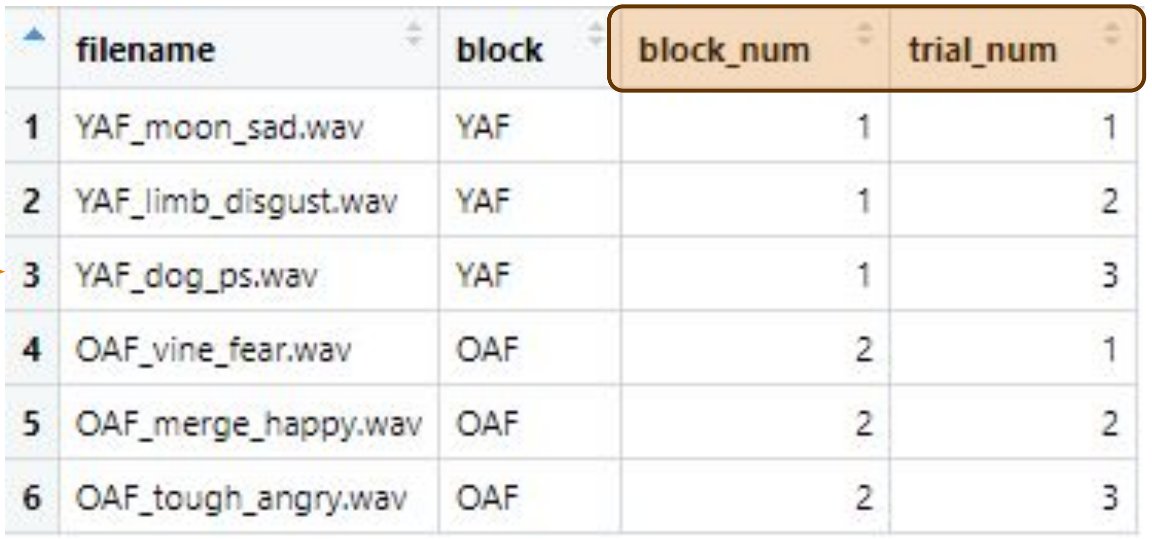
The randomStim() function

- Randomization performed with the base R sample() function
- Randomizes block order, order of trials within blocks, or both



	filename	block
1	OAF_merge_happy.wav	OAF
2	OAF_tough_angry.wav	OAF
3	OAF_vine_fear.wav	OAF
4	YAF_dog_ps.wav	YAF
5	YAF_limb_disgust.wav	YAF
6	YAF_moon_sad.wav	YAF

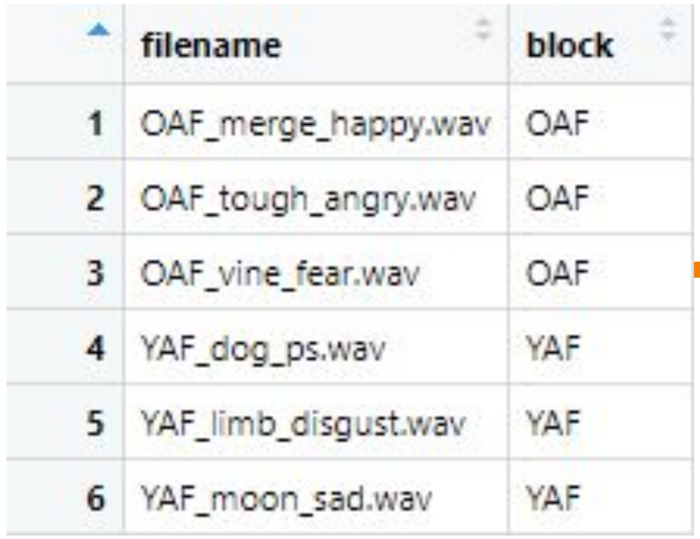
outFile = “stim01.csv”



	filename	block	block_num	trial_num
1	YAF_moon_sad.wav	YAF	1	1
2	YAF_limb_disgust.wav	YAF	1	2
3	YAF_dog_ps.wav	YAF	1	3
4	OAF_vine_fear.wav	OAF	2	1
5	OAF_merge_happy.wav	OAF	2	2
6	OAF_tough_angry.wav	OAF	2	3

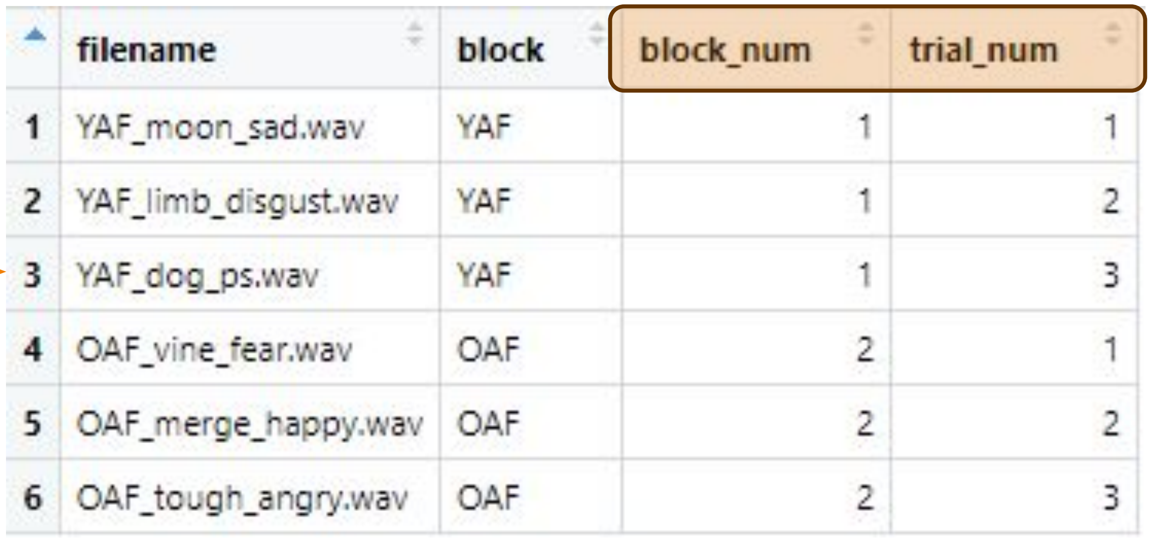
The randomStim() function

- Randomization performed with the base R sample() function
- Randomizes block order, order of trials within blocks, or both
- Can run inside or outside a shiny application



	filename	block
1	OAF_merge_happy.wav	OAF
2	OAF_tough_angry.wav	OAF
3	OAF_vine_fear.wav	OAF
4	YAF_dog_ps.wav	YAF
5	YAF_limb_disgust.wav	YAF
6	YAF_moon_sad.wav	YAF

outFile = “stim01.csv”



	filename	block	block_num	trial_num
1	YAF_moon_sad.wav	YAF	1	1
2	YAF_limb_disgust.wav	YAF	1	2
3	YAF_dog_ps.wav	YAF	1	3
4	OAF_vine_fear.wav	OAF	2	1
5	OAF_merge_happy.wav	OAF	2	2
6	OAF_tough_angry.wav	OAF	2	3

More server-side tasks: speechcollectr contributions

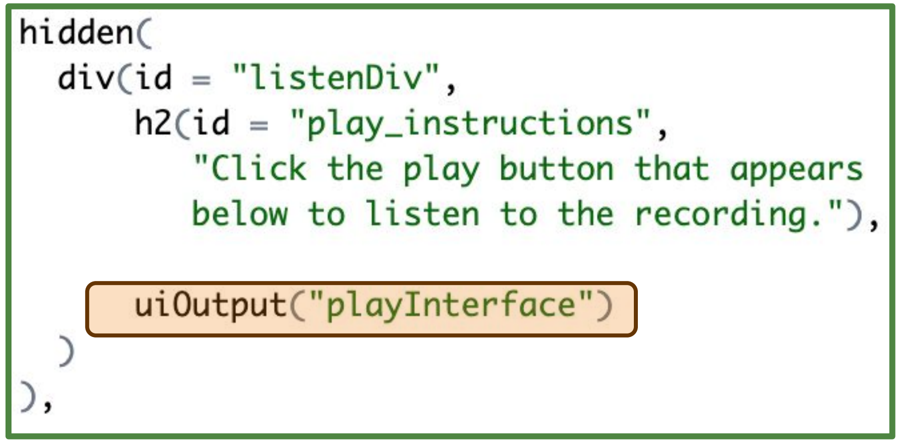
- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including recorded audio files)
- Recording reaction times

The playBttn() Function

- Actually a ui function but…
  - Common problem: The ui object only deals with static elements. The audio file will change on each trial, so can’t be hard-coded in ui.

The playBttn() Function

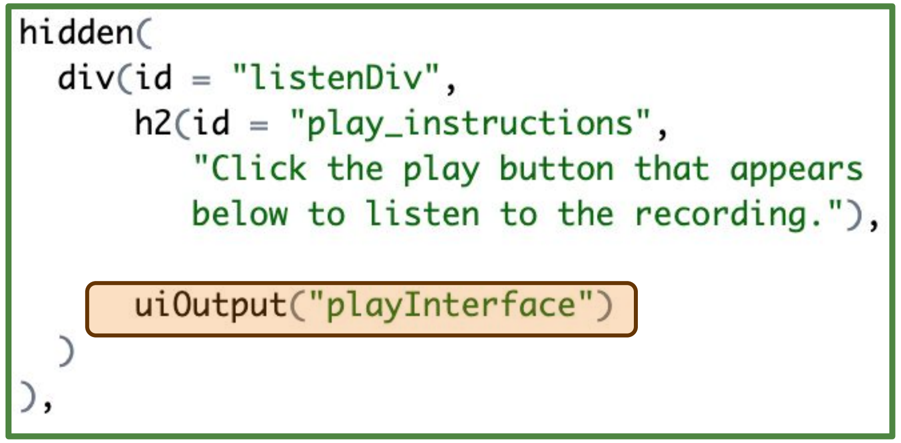
- Actually a ui function but…
    - Common problem: The ui object only deals with static elements. The audio file will change on each trial, so can't be hard-coded in ui.
- Solution: add a placeholder with uiOutput()



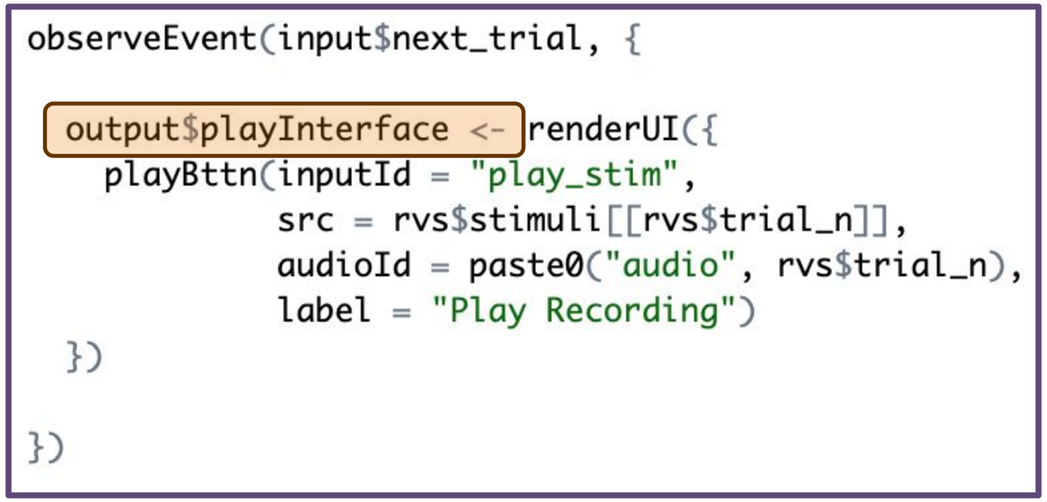
```
hidden(
  div(id = "listenDiv",
      h2(id = "play_instructions",
         "Click the play button that appears
         below to listen to the recording."),

      uiOutput("playInterface")
  )
),
```

The playBttn() Function

- Actually a ui function but…
  - Common problem: The ui object only deals with static elements. The audio file will change on each trial, so can't be hard-coded in ui.
- Solution: add a placeholder with uiOutput()
  - Adds the string "playInterface" to the **output** argument of the server function



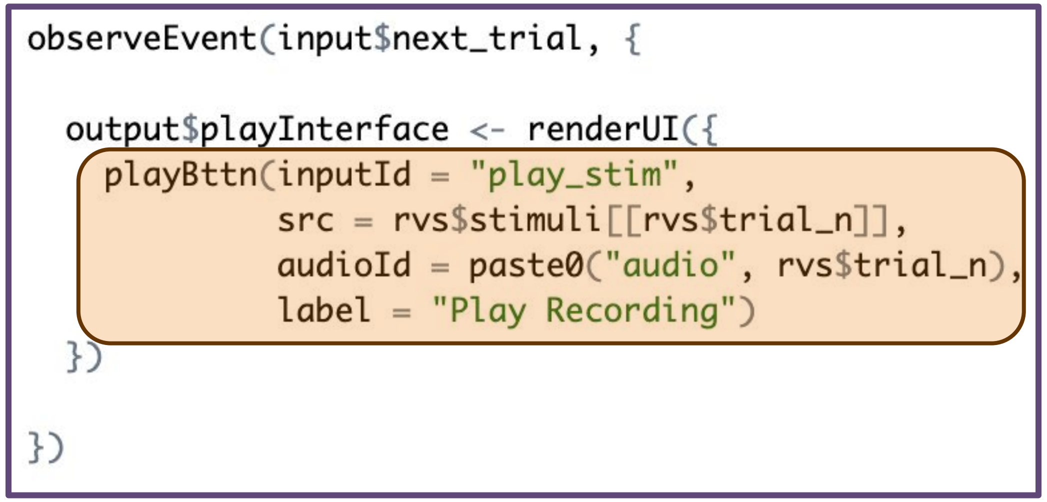
```
hidden(
  div(id = "listenDiv",
      h2(id = "play_instructions",
         "Click the play button that appears
         below to listen to the recording."),

      uiOutput("playInterface")
  )
),
```



```
observeEvent(input$next_trial, {

  output$playInterface <- renderUI({
    playBttn(inputId = "play_stim",
             src = rvs$stimuli[[rvs$trial_n]],
             audioId = paste0("audio", rvs$trial_n),
             label = "Play Recording")
  })

})
```

The playBttn() Function

- has same arguments as actionButton()
  - inputId
  - label
- additional arguments:
  - src (file path for the audio file to be played, defined dynamically using the trial number)
  - audioId (an inputId to attach to the audio element when it is loaded)



```
observeEvent(input$next_trial, {

  output$playInterface <- renderUI({
    playBttn(inputId = "play_stim",
             src = rvs$stimuli[[rvs$trial_n]],
             audioId = paste0("audio", rvs$trial_n),
             label = "Play Recording")
  })

})
```

More server-side tasks: speechcollectr contributions

- Advancing the trial number
- Generating a unique ID number for each participant
- Randomizing stimuli
- Playing/showing the correct stimulus for the trial
- Saving participant responses (including recorded audio files)
- Recording reaction times

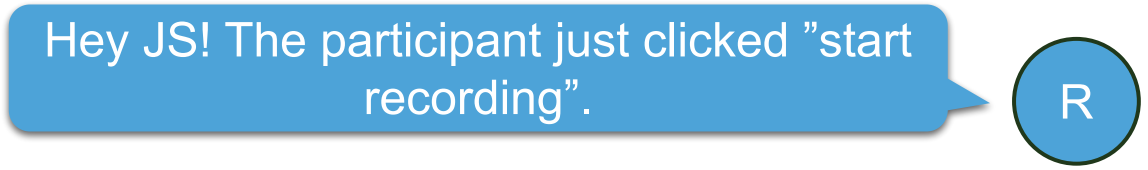
Audio Recording with speechcollectr

- Method 1: A more extensible method using two simple event functions in the server code.
- Method 2: A more comprehensive interface with built-in settings for displaying written stimuli using shiny modules.

Audio Recording with speechcollectr

- Both methods rely on JavaScript backend code, minimally modified from recorder.js (Diamond, 2013; Negrota, 2022)

Audio Recording with speechcollectr



Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

JS

Hey R. Ok, cool. Lemme ask to use the microphone.

Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

Hey R. Ok, cool. Lemme ask to use the microphone.

JS ■ ■ ■

Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

Hey R. Ok, cool. Lemme ask to use the microphone.

JS

We’re good. Recording started.

Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

Hey R. Ok, cool. Lemme ask to use the microphone.

We’re good. Recording started.

R

Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

Hey R. Ok, cool. Lemme ask to use the microphone.

We’re good. Recording started.

The participant just clicked stop.

R

Audio Recording with speechcollectr

Hey JS! The participant just clicked ”start recording”.

Hey R. Ok, cool. Lemme ask to use the microphone.

We’re good. Recording started.

The participant just clicked stop.

Great. Here’s a large binary object containing the audio we recorded.

JS

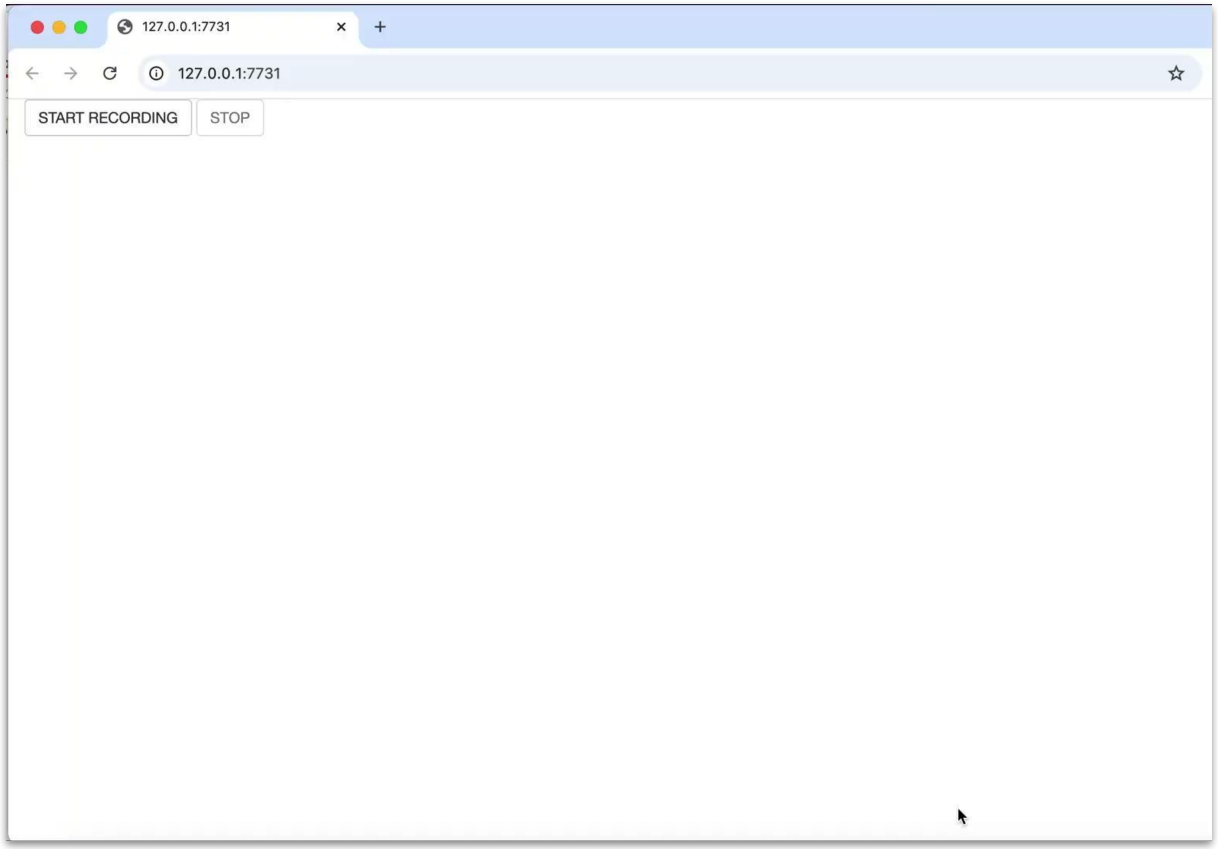
Audio Recording with speechcollectr

- speechcollectr handles writing the binary data to a wav file
- 16-bit WAV files at the *browser's default sampling rate*
  - Usually 44.1 or 48 kHz

Audio Recording with speechcollectr

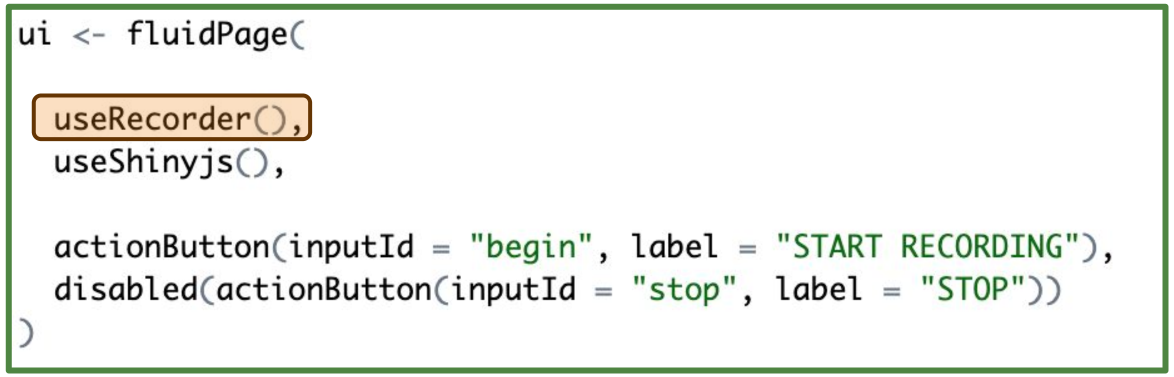
- Method 1: A more extensible method using two simple event functions in the server code.
- Method 2: A more comprehensive interface with built-in settings for displaying written stimuli using shiny modules.

Audio Recording with speechcollectr

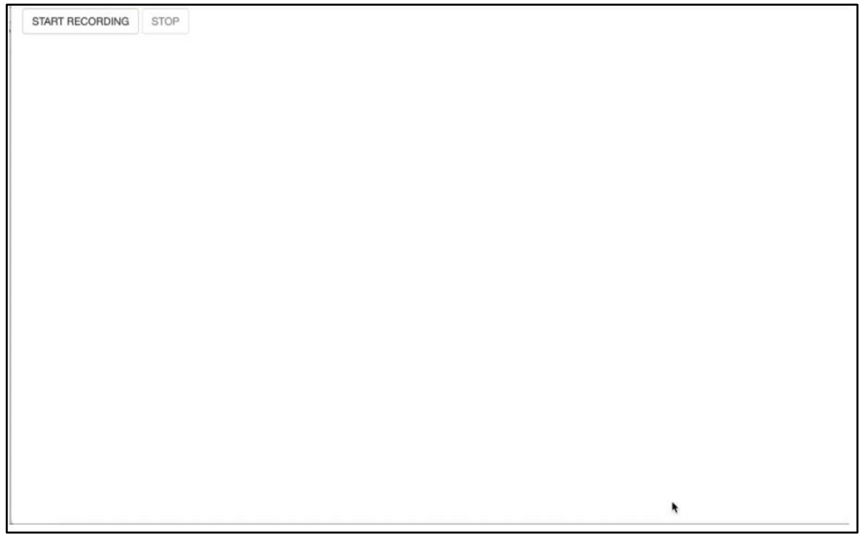


Code: https://github.com/abbey-thomas/speechcollectr-demo2024/blob/main/app5.R

Audio Recording with speechcollectr

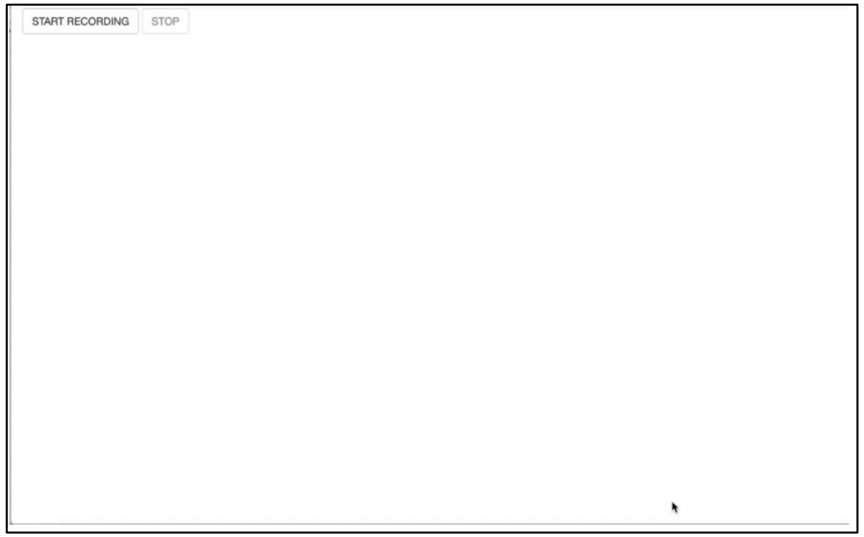


```r
ui <- fluidPage(

  useRecorder(),
  useShinyjs(),

  actionButton(inputId = "begin", label = "START RECORDING"),
  disabled(actionButton(inputId = "stop", label = "STOP"))
)
```



Audio Recording with speechcollectr

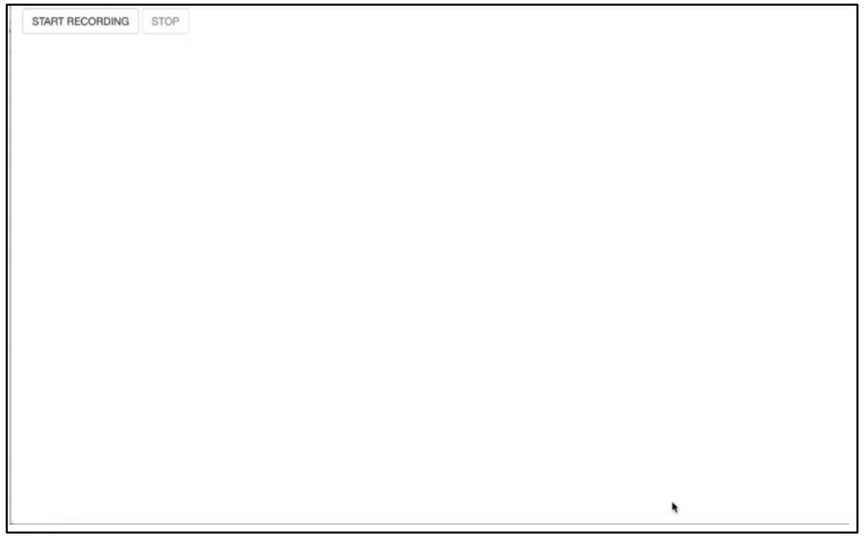
```r
ui <- fluidPage(

  useRecorder(),
  useShinyjs(),

  actionButton(inputId = "begin", label = "START RECORDING"),
  disabled(actionButton(inputId = "stop", label = "STOP"))
)
```



```r
server <- function(input, output, session) {
  observeEvent(input$begin, {
    disable("begin")
    startRec(readyId = "recording")
  })

  observeEvent(input$recording, {
    enable("stop")
  })

  observeEvent(input$stop, {
    disable("stop")
    stopRec(filename = "test.wav",
            finishedId = "file_saved")
  })

  observeEvent(input$file_saved, {
    enable("begin")
  })
}
```

Audio Recording with speechcollectr

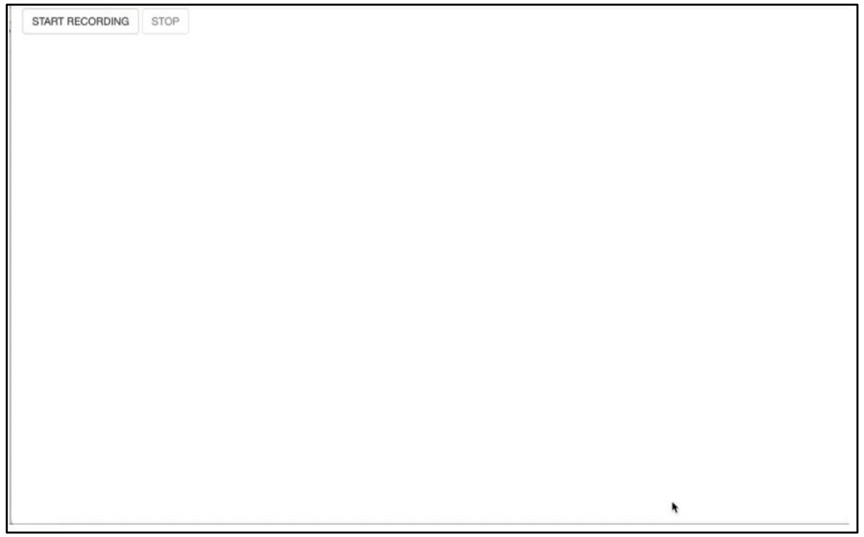
```
ui <- fluidPage(

  useRecorder(),
  useShinyjs(),

  actionButton(inputId = "begin", label = "START RECORDING"),
  disabled(actionButton(inputId = "stop", label = "STOP"))
)
```



```
server <- function(input, output, session) {
  observeEvent(input$begin, {
    disable("begin")
    startRec(readyId = "recording")
  })

  observeEvent(input$recording, {
    enable("stop")
  })

  observeEvent(input$stop, {
    disable("stop")
    stopRec(filename = "test.wav",
            finishedId = "file_saved")
  })

  observeEvent(input$file_saved, {
    enable("begin")
  })
}
```

Audio Recording with speechcollectr

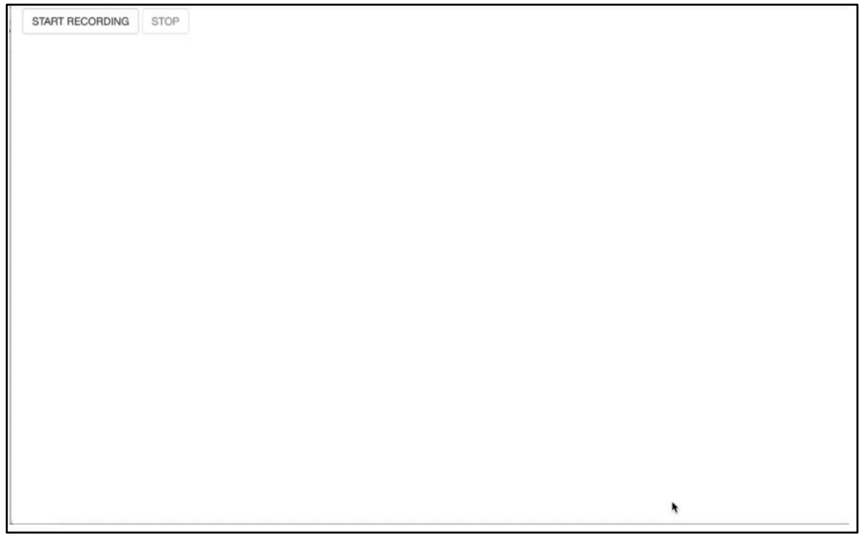
```r
ui <- fluidPage(

  useRecorder(),
  useShinyjs(),

  actionButton(inputId = "begin", label = "START RECORDING"),
  disabled(actionButton(inputId = "stop", label = "STOP"))
)
```



```r
server <- function(input, output, session) {
  observeEvent(input$begin, {
    disable("begin")
    startRec(readyId = "recording")
  })

  observeEvent(input$recording, {
    enable("stop")
  })

  observeEvent(input$stop, {
    disable("stop")
    stopRec(filename = "test.wav",
            finishedId = "file_saved")
  })

  observeEvent(input$file_saved, {
    enable("begin")
  })
}
```

Audio Recording with speechcollectr

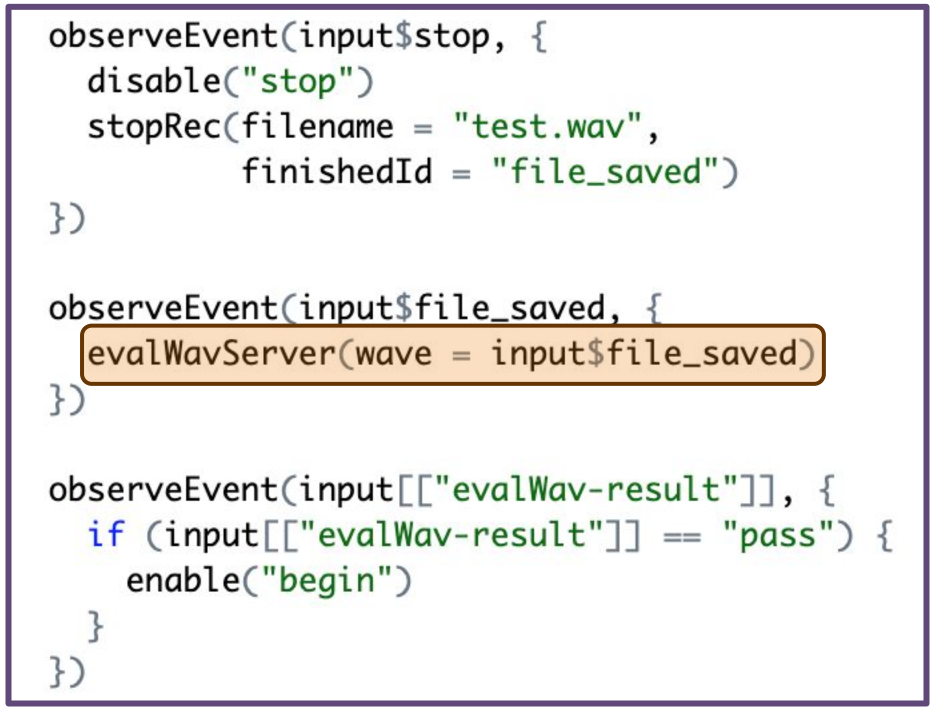
```r
ui <- fluidPage(

  useRecorder(),
  useShinyjs(),

  actionButton(inputId = "begin", label = "START RECORDING"),
  disabled(actionButton(inputId = "stop", label = "STOP"))
)
```



```r
server <- function(input, output, session) {
  observeEvent(input$begin, {
    disable("begin")
    startRec(readyId = "recording")
  })

  observeEvent(input$recording, {
    enable("stop")
  })

  observeEvent(input$stop, {
    disable("stop")
    stopRec(filename = "test.wav",
            finishedId = "file_saved")
  })

  observeEvent(input$file_saved, {
    enable("begin")
  })
}
```

evalWavServer(): Optimizing for remote audio recording

- As the name suggests, this function is added to the server code.

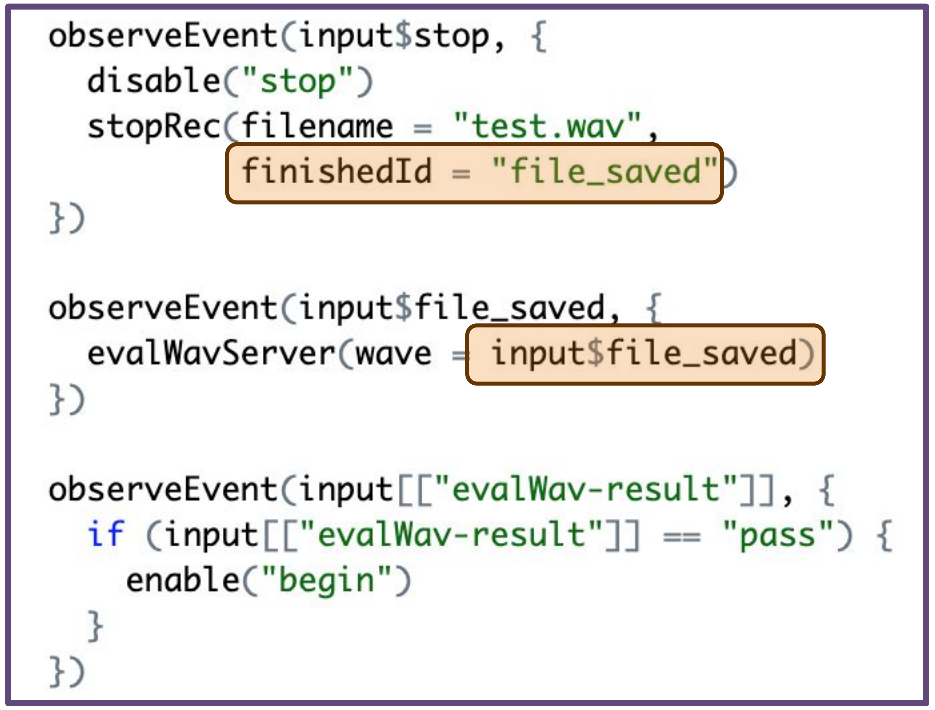


```r
observeEvent(input$stop, {
  disable("stop")
  stopRec(filename = "test.wav",
          finishedId = "file_saved")
})

observeEvent(input$file_saved, {
  evalWavServer(wave = input$file_saved)
})

observeEvent(input[["evalWav-result"]], {
  if (input[["evalWav-result"]] == "pass") {
    enable("begin")
  }
})
```

evalWavServer(): Optimizing for remote audio recording

- As the name suggests, this function is added to the server code.

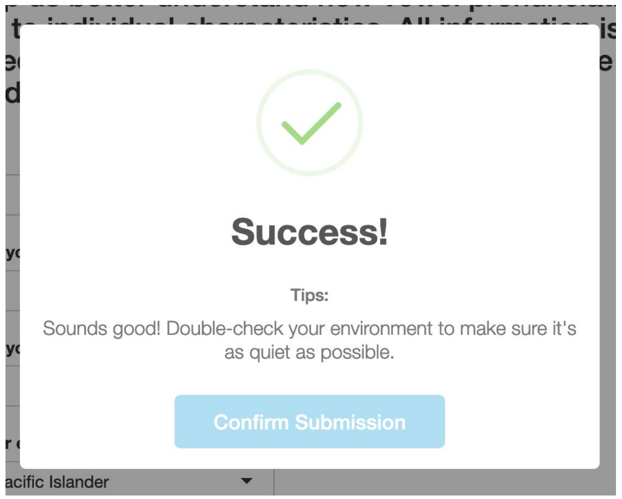
- Immediate acoustic analysis of wav from stopRec()
  - check sampling rate
  - find speech token
  - get signal-to-noise ratio of speech to background/environment noise
  - check for amplitude clipping



```
observeEvent(input$stop, {
  disable("stop")
  stopRec(filename = "test.wav",
          finishedId = "file_saved")
})

observeEvent(input$file_saved, {
  evalWavServer(wave = input$file_saved)
})

observeEvent(input[["evalWav-result"]], {
  if (input[["evalWav-result"]] == "pass") {
    enable("begin")
  }
})
```

evalWavServer(): Optimizing for remote audio recording

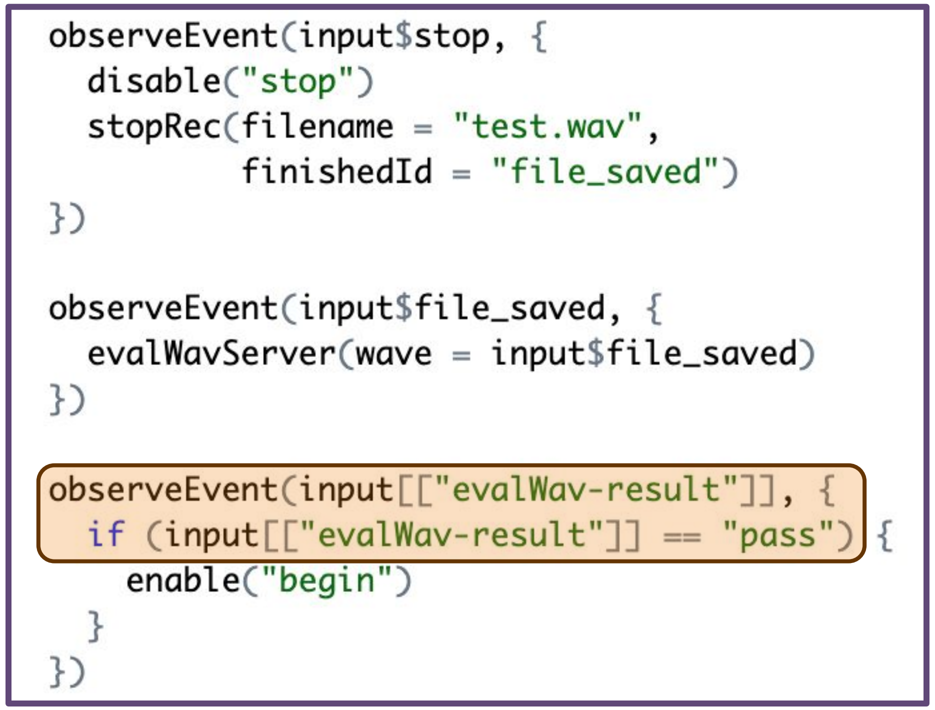
- As the name suggests, this function is added to the server code.



Success!

Tips:

Sounds good! Double-check your environment to make sure it's as quiet as possible.



```
observeEvent(input$stop, {
  disable("stop")
  stopRec(filename = "test.wav",
          finishedId = "file_saved")
})

observeEvent(input$file_saved, {
  evalWavServer(wave = input$file_saved)
})

observeEvent(input[["evalWav-result"]], {
  if (input[["evalWav-result"]] == "pass") {
    enable("begin")
  }
})
```

- Provides feedback
  - to participant in a dialog box/popup that appears on screen.
  - to the server in input[["evalWav-result"]] (value of *pass* or *fail*)

Audio Recording with speechcollectr

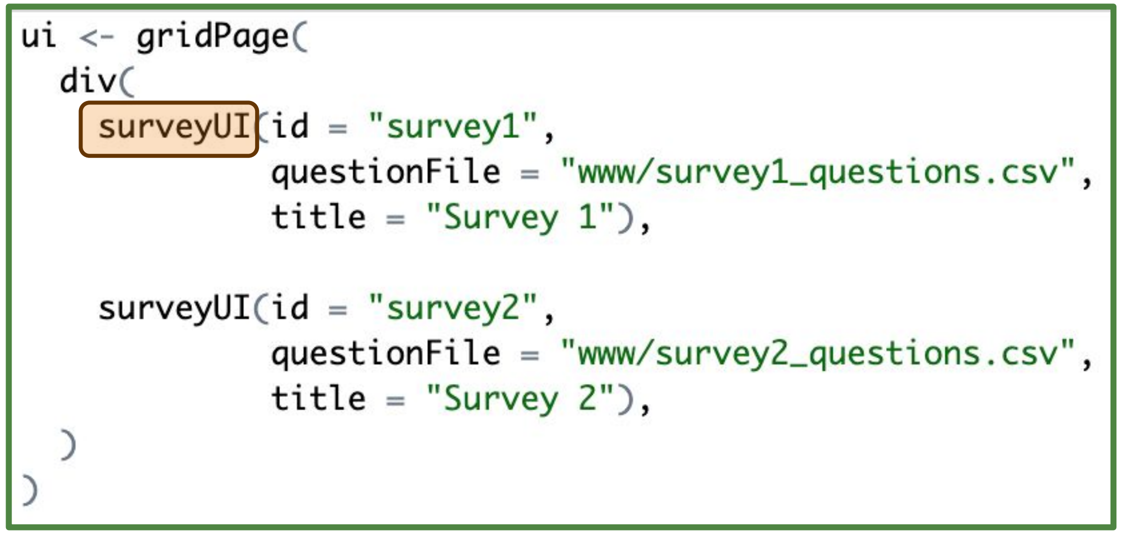
- Method 1: A more extensible method using two simple event functions in the server code.
- Method 2: A more comprehensive interface with built-in settings for displaying written stimuli using shiny **modules**.

Topics Covered in Today’s Talk

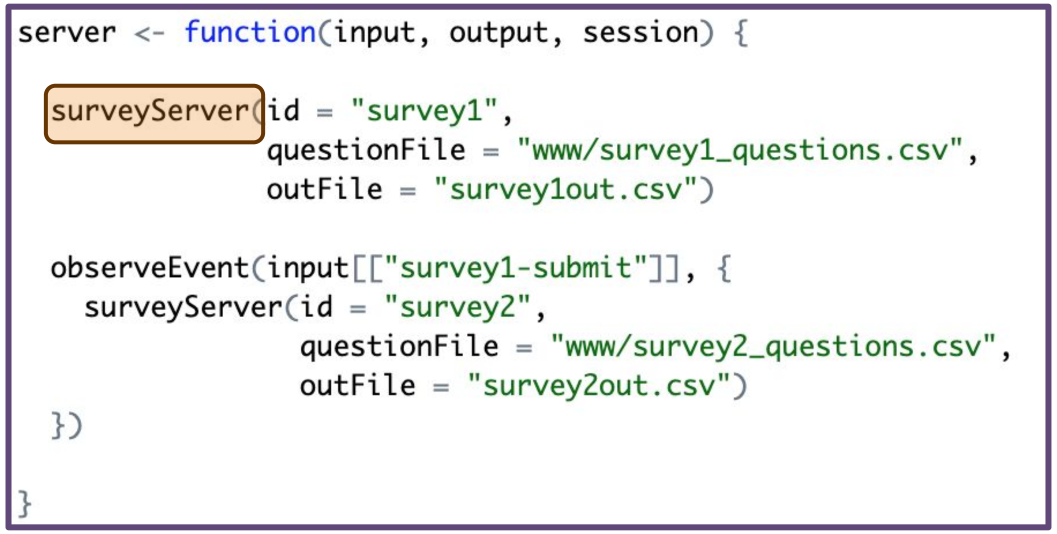
1. What is speechcollectr?
2. Fundamentals of experiment code
3. Possible interfaces
4. Server/ reactive coding in R
5. Other common server task methods
6. speechcollectr modules for common experimental tasks

A Brief Introduction to Modules

- Include both a ui function and server function (placed in an app’s ui and server code, respectively)



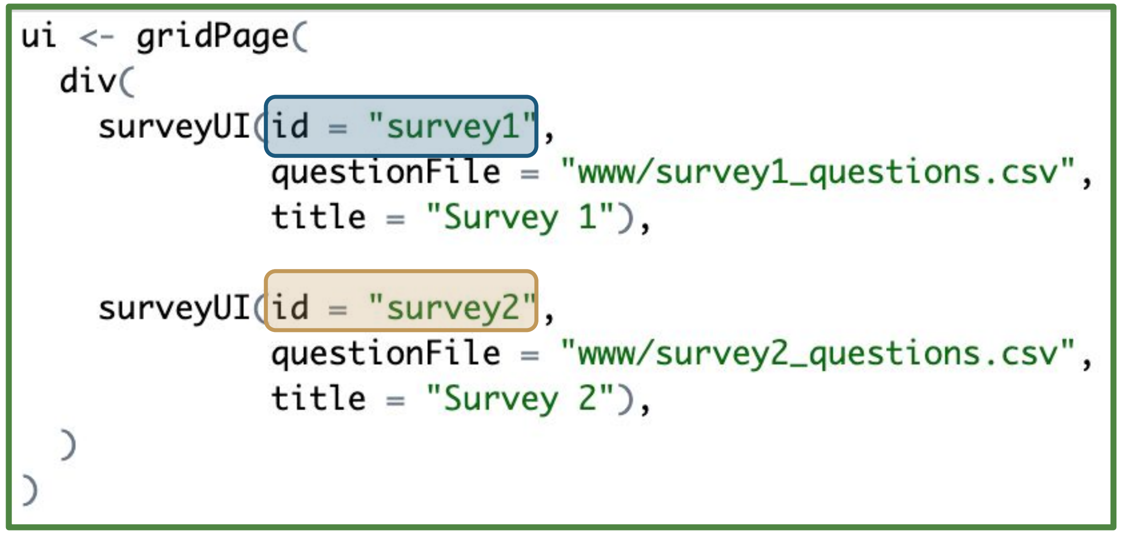
```r
ui <- gridPage(
  div(
    surveyUI(id = "survey1",
             questionFile = "www/survey1_questions.csv",
             title = "Survey 1"),

    surveyUI(id = "survey2",
             questionFile = "www/survey2_questions.csv",
             title = "Survey 2"),
  )
)
```



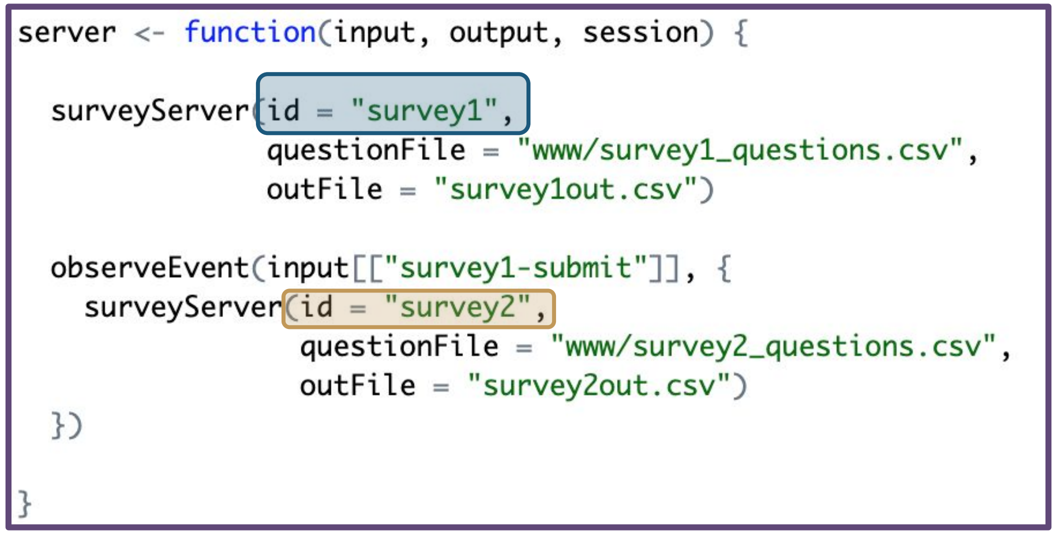
```r
server <- function(input, output, session) {

  surveyServer(id = "survey1",
               questionFile = "www/survey1_questions.csv",
               outFile = "survey1out.csv")

  observeEvent(input[["survey1-submit"]], {
    surveyServer(id = "survey2",
                 questionFile = "www/survey2_questions.csv",
                 outFile = "survey2out.csv")
  })

}
```

A Brief Introduction to Modules

- Include both a ui function and server function (placed in an app's ui and server code, respectively)
- Both the ui and server functions must share an **id**, which will be appended to all inputIds created inside the module.



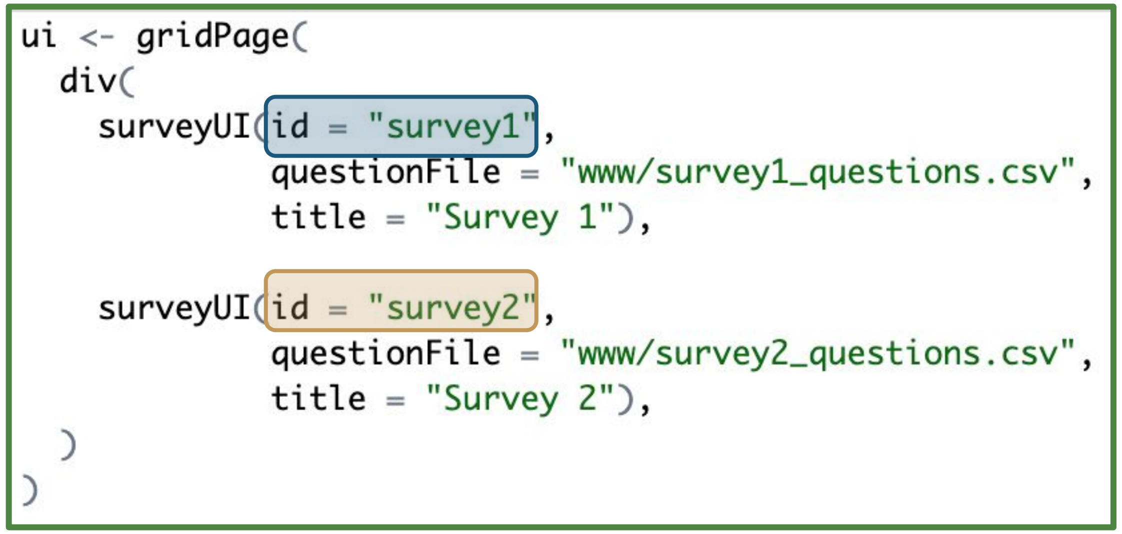
```
ui <- gridPage(
  div(
    surveyUI(id = "survey1",
             questionFile = "www/survey1_questions.csv",
             title = "Survey 1"),

    surveyUI(id = "survey2",
             questionFile = "www/survey2_questions.csv",
             title = "Survey 2"),
  )
)
```



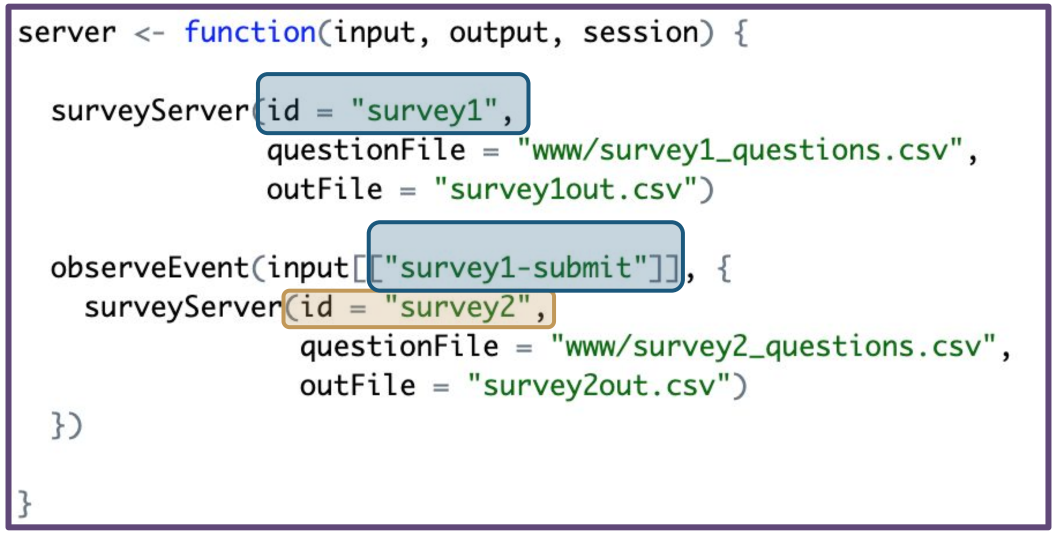
```
server <- function(input, output, session) {

  surveyServer(id = "survey1",
               questionFile = "www/survey1_questions.csv",
               outFile = "survey1out.csv")

  observeEvent(input[["survey1-submit"]], {
    surveyServer(id = "survey2",
                 questionFile = "www/survey2_questions.csv",
                 outFile = "survey2out.csv")
  })

}
```

A Brief Introduction to Modules

- Include both a ui function and server function (placed in an app’s ui and server code, respectively)
- Both the ui and server functions must share an **id**, which will be appended to all inputIds created inside the module.



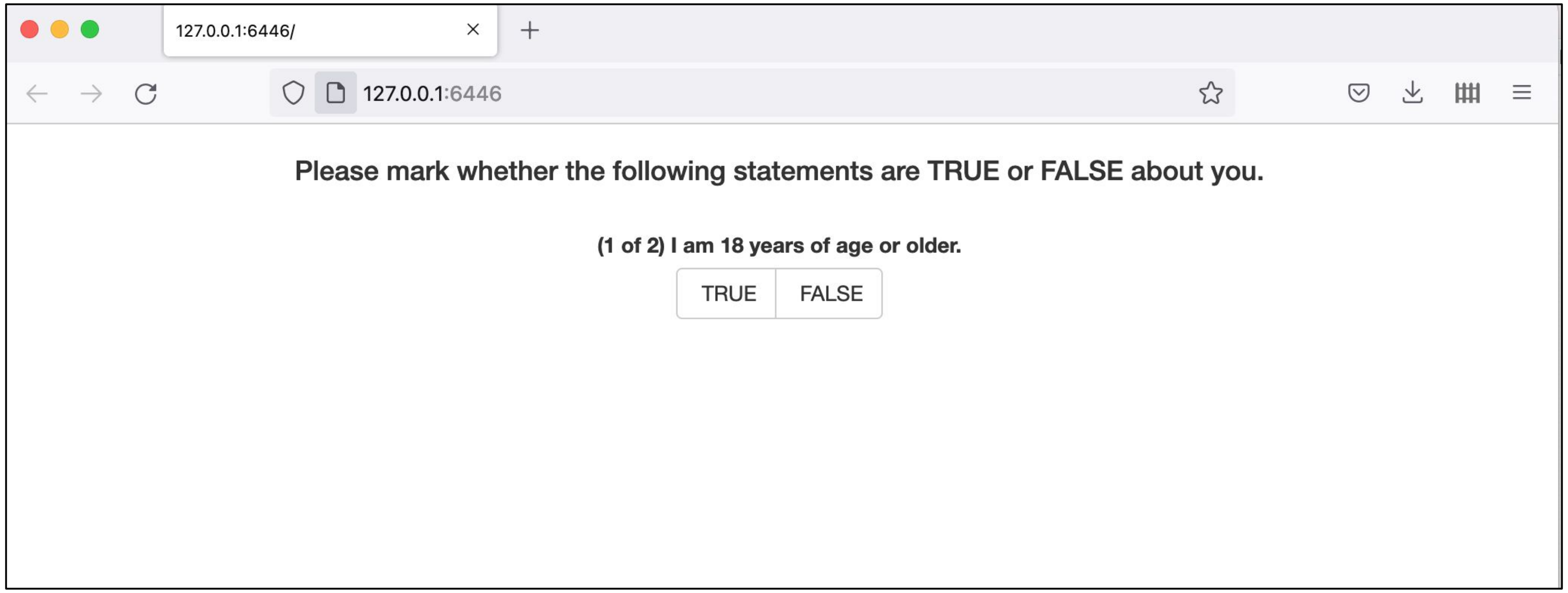
```
ui <- gridPage(
  div(
    surveyUI(id = "survey1",
             questionFile = "www/survey1_questions.csv",
             title = "Survey 1"),

    surveyUI(id = "survey2",
             questionFile = "www/survey2_questions.csv",
             title = "Survey 2"),
  )
)
```



```
server <- function(input, output, session) {

  surveyServer(id = "survey1",
               questionFile = "www/survey1_questions.csv",
               outFile = "survey1out.csv")

  observeEvent(input[["survey1-submit"]], {
    surveyServer(id = "survey2",
                 questionFile = "www/survey2_questions.csv",
                 outFile = "survey2out.csv")
  })

}
```

Modules Available in speechcollectr

- checkUI() / checkServer()
  - can be used to present true/false questions one at a time to participants to check their qualifications or levels of environmental noise (for example)


Please mark whether the following statements are TRUE or FALSE about you.

(1 of 2) I am 18 years of age or older.

Modules Available in speechcollectr

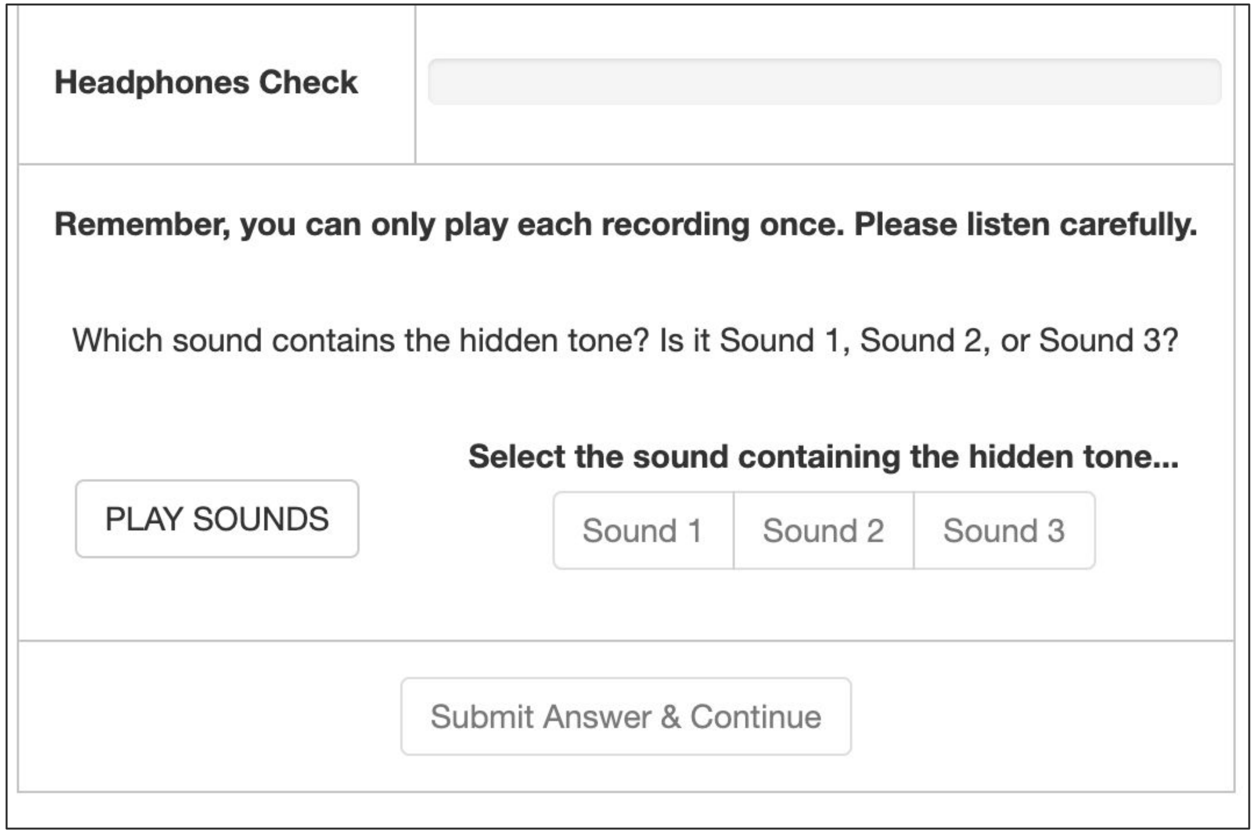
- checkUI() / checkServer()
- surveyUI() / surveyServer()
    - presents multiple questions simultaneously of various input types (e.g., radio buttons, text entry, etc.)
    - Questions built in a CSV table…see also surveyPrep()

Modules Available in speechcollectr

- checkUI() / checkServer()
- surveyUI() / surveyServer()
- headphoneTest*()
  - R implementations of existing screening tools for determining whether (remote) participants are wearing headphones

Milne, A. E., Bianco, R., Poole, K. C., Zhao, S., Oxenham, A. J., Billig, A. J., & Chait, M. (2021). An online headphone screening test based on dichotic pitch. Behavior Research Methods, 53(4), 1551-1562.

Woods, K. J., Siegel, M. H., Traer, J., & McDermott, J. H. (2017). Headphone screening to facilitate web-based auditory experiments. Attention, Perception, & Psychophysics, 79(7), 2064-2072. https://doi.org/10.3758/s13414-017-1361-2


Headphones Check

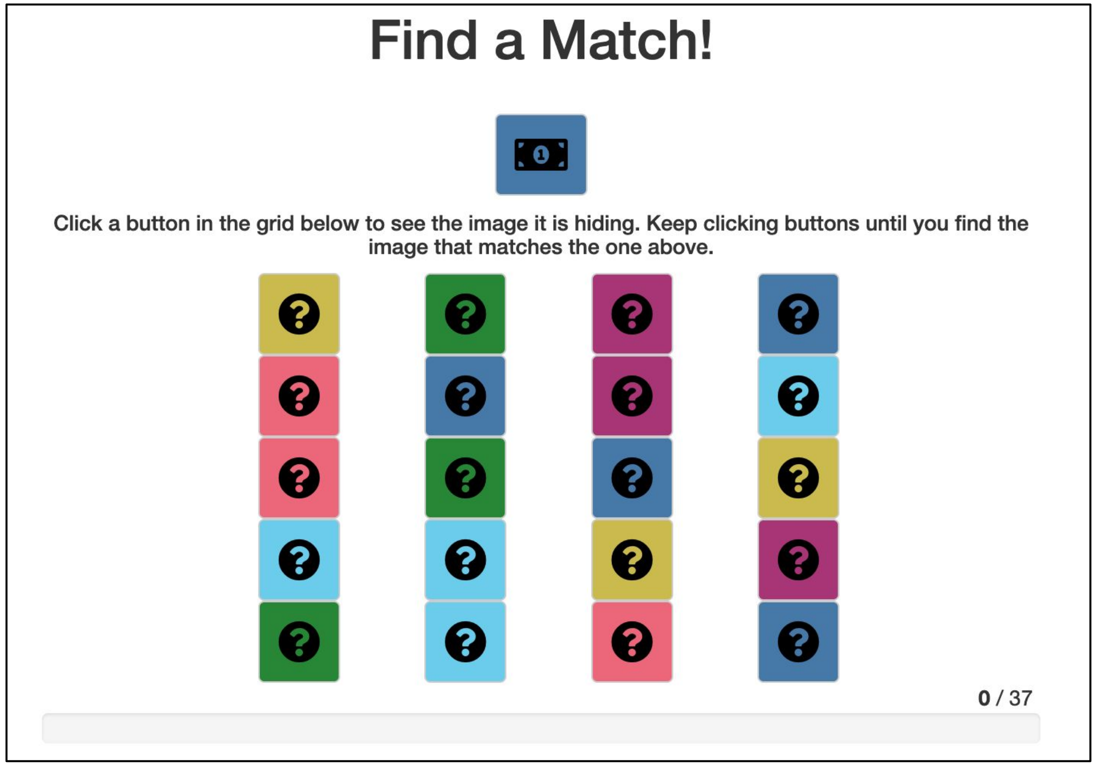
Remember, you can only play each recording once. Please listen carefully.

Which sound contains the hidden tone? Is it Sound 1, Sound 2, or Sound 3?

Select the sound containing the hidden tone...

Modules Available in speechcollectr

- checkUI() / checkServer()
- surveyUI() / surveyServer()
- headphoneTestUI() / headphoneTestServer()
- matchUI() / matchServer()
    - a matching game interface for monitoring/sustaining attention



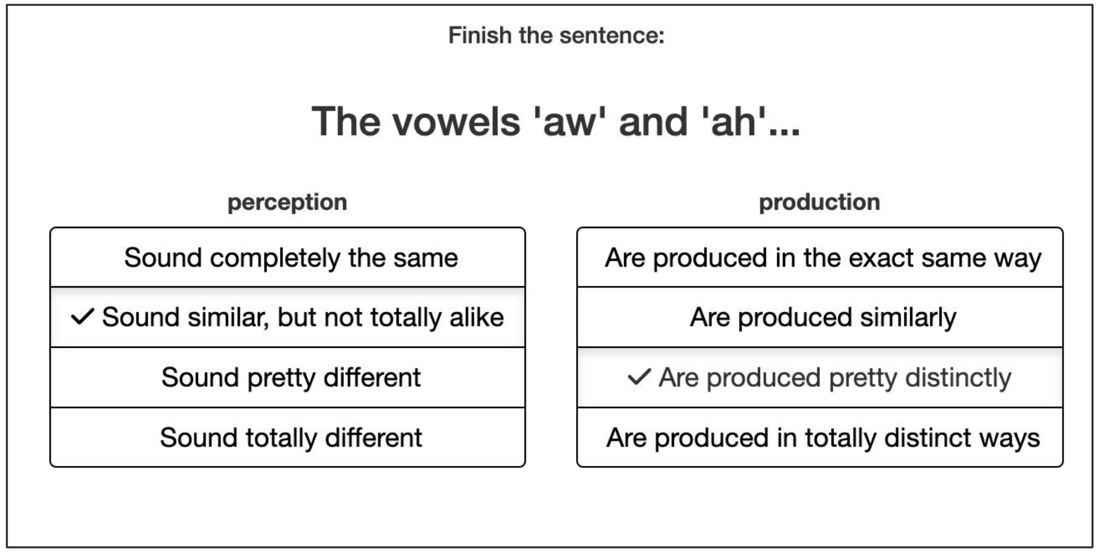
Find a Match!

Click a button in the grid below to see the image it is hiding. Keep clicking buttons until you find the image that matches the one above.

0 / 37

Modules Available in speechcollectr

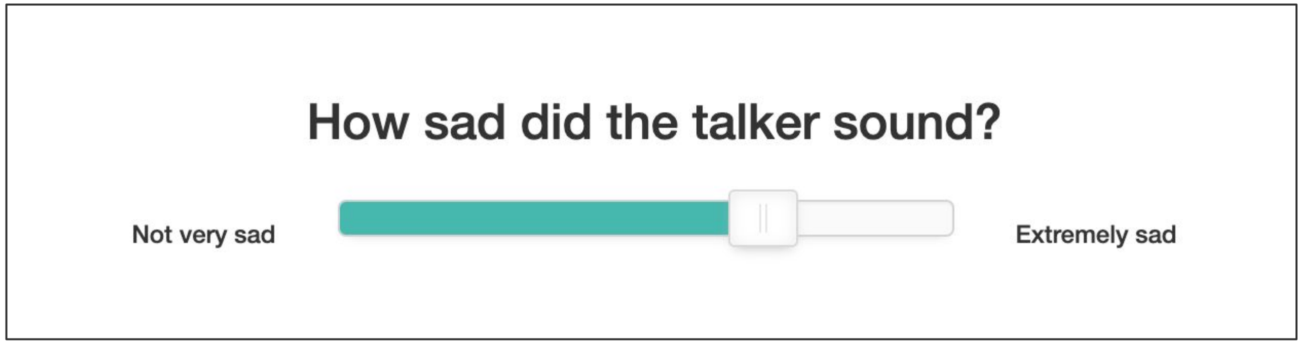
- checkUI() / checkServer()
- surveyUI() / surveyServer()
- headphoneTestUI() / headphoneTestServer()
- matchUI() / matchServer()
- rateUI() / rateServer()



Finish the sentence:

The vowels 'aw' and 'ah'...

perception	production
Sound completely the same	Are produced in the exact same way
✓ Sound similar, but not totally alike	Are produced similarly
Sound pretty different	✓ Are produced pretty distinctly
Sound totally different	Are produced in totally distinct ways



How sad did the talker sound?

Not very sad — Extremely sad

(Some) Modules Available in speechcollectr

Check package documentation to see other ui/server paired tools and examples.

…Or have a go at building the modules you need!

Resources

The speechcollectr package:

https://github.com/abbey-thomas/speechcollectr

Article about doing data collection with R shiny and speechcollectr:
https://doi.org/10.3758/s13428-024-02399-z

Code accompanying article (example experiments):
https://github.com/abbey-thomas/shiny4speech-science

Code and slides from today's demonstrations:
https://github.com/abbey-thomas/speechcollectr-demo2024