

Day 2: Online experiments

Amy Perfors

DAY 2: EXPERIMENTS

Tentative plan

1. Experiment logic, motivation, and design
2. R basics for coding: branching, functions, lists
3. Creating a template experiment with jaysire and putting it online
4. Making a more complex experiment

DAY 2: EXPERIMENTS

Tentative plan

1. Experiment logic, motivation, and design
2. R basics for coding: branching, functions, lists
3. Creating a template experiment with jaysire and putting it online
4. Making a more complex experiment

REMINDER FROM YESTERDAY



Server side

In this case, this is you! You (or your server, which will be Google App Engine) are serving up an experiment to your participant.



Client side

The client's web browser serves up webpages.

As programmer, you are writing code so that the browser on the client side knows what to do

Usually **HTML**: a markup language for displaying all your content

Javascript is a client-side language that lets you do more complex things. It is embedded in html. I'll be talking about a particular library called **jsPsych** used for making online experiments

Jaysire is a **R package** consisting of wrapper functions for javascript. It means you can write the code in R and it will translate to javascript for you

REMINDER: JSPSYCH

Remember from yesterday how we learned a little about how experiments work, and the structure of jsPsych

The way jsPsych works is by creating a bunch of plugins that are Javascript code you can call to do some of the complicated stuff in your experiment

jsPsych

[Introduction](#)

[Tutorials](#) ▾

[Overview](#) ▾

[Core Library API](#) ▾

[Plugins](#) ▾

[About](#) ▾



jspsych

REMINDER: JSPSYCH

Remember from yesterday how we learned a little about how experiments work, and the structure of jsPsych

**You don't
need to
know how
to do this!**

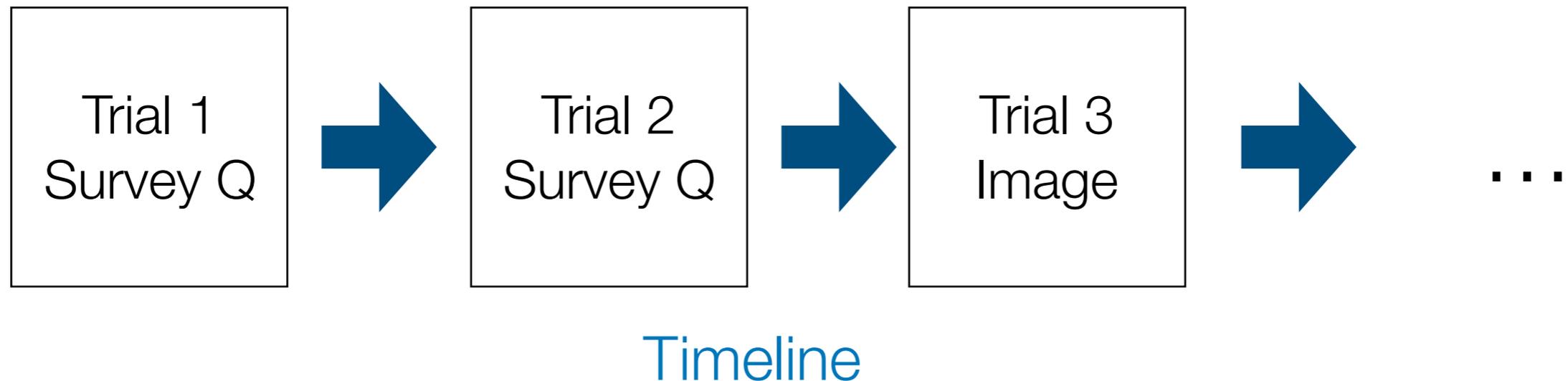
Jaysire

jsPsych
Introduction
Tutorials ▾
Overview ▾
Core Library API ▾
Plugins ▾
About ▾

jspsych

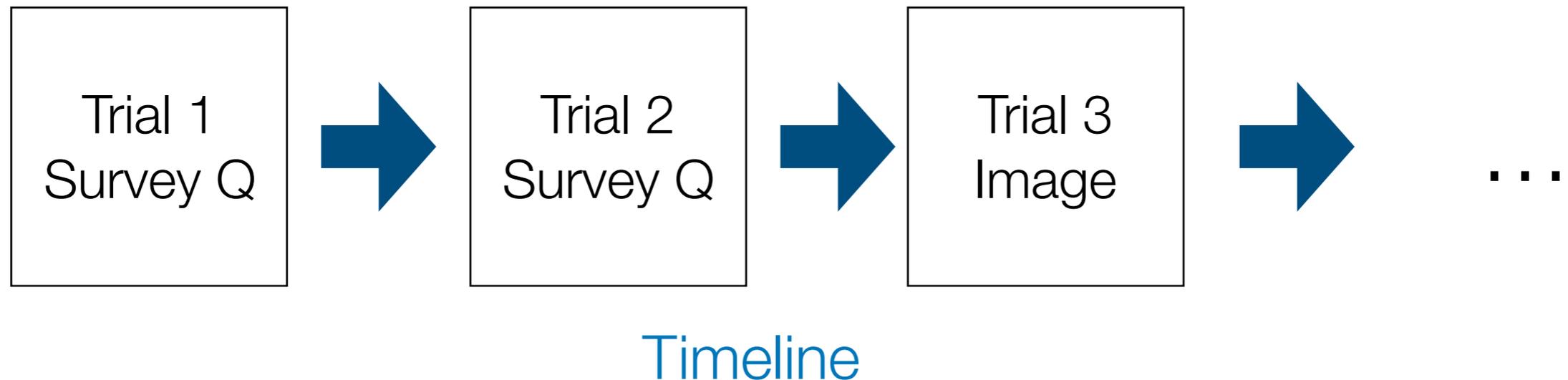
TIMELINES

The way jsPsych and jaysire both work is by building a description of an experiment known as a **timeline**, which is basically a series of variables defining each step (trial).



TIMELINES

The way jsPsych and jaysire both work is by building a description of an experiment known as a **timeline**, which is basically a series of variables defining each step (trial).



You can create and randomise variables, nest timelines within one another, present audio / visual / text / images, create lots of different kinds of questions, etc.

JAYSIRE: STRUCTURE

Collections of functions that do different things:
<https://djenavarro.github.io/jaysire/reference/>

build_ Builds the experiment in javascript

trial_ Functions for creating lots of different kinds of trials

tl_ Functions for putting trials together into a timeline

run_ Functions for running the experiment

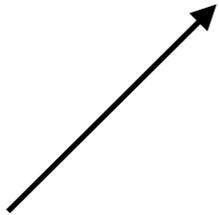
fn_ Manipulates javascript functions (advanced)

JAYSIRE: INSTALLATION

```
install.packages("remotes")
```

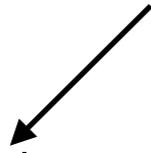
```
remotes::install_github("djnavarro/jaysire")
```

loads it onto
your machine



```
library(jaysire)
```

puts it in working
memory (need this
in every experiment
you make)



```
> remotes::install_github("djnavarro/jaysire")
Downloading GitHub repo djnavarro/jaysire@master
These packages have more recent versions available.
Which would you like to update?

1: All
2: CRAN packages only
3: None
4: rlang (0.4.1 -> 0.4.2 ) [CRAN]
5: digest (0.6.22 -> 0.6.23) [CRAN]

Enter one or more numbers, or an empty line to skip updates:
1
rlang (0.4.1 -> 0.4.2 ) [CRAN]
digest (0.6.22 -> 0.6.23) [CRAN]
Installing 2 packages: rlang, digest

There are binary versions available but the source versions are later:
      binary source needs_compilation
rlang  0.3.1  0.4.2                    TRUE
digest 0.6.18 0.6.23                    TRUE

Do you want to install from sources the packages which need compilation?
y/n: y
```

JAYSIRE: GETTING STARTED

First, let's make sure we're all up-to-date on the day2 content. Go to your CHDSS/chdss2019_content folder in the terminal.

Type `git status` at the prompt to see if you're up to date. If you are, great!

If not, pull the newer content: `git pull`

And then check that you're up-to-date with `git status`

JAYSIRE: GETTING STARTED

Navigate to the `day2` folder (remember the `cd` command).

You should see folders that look like the following:

`slides`
`experiments`

We want you to copy all of `day2` into your `summerschool` folder, so that we don't have 70+ people all trying to change the actual `chdss2019_content` repository

You can do this manually or in the command line:

```
cp -rf ../day2/ ../../summerschool/
```

Now move to the `day2` folder there:

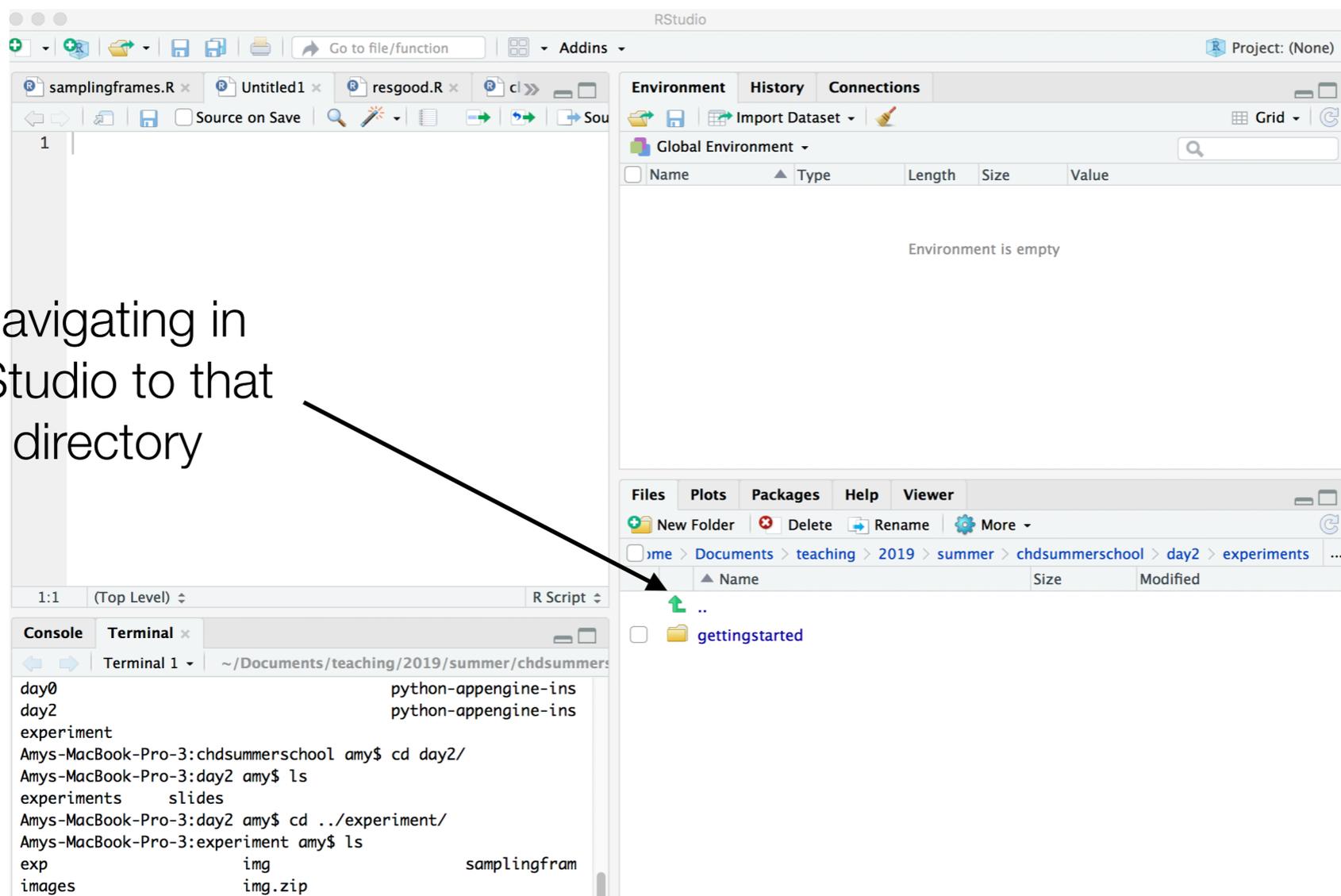
```
cd ../../summerschool/day2
```

 where you should see the same folders

JAYSIRE: GETTING STARTED

Navigate into the `experiments/gettingstarted` folder which should be empty. Set this as your working directory so that RStudio knows you're there too.

Navigating in RStudio to that directory



JAYSIRE: GETTING STARTED

Navigate into the `experiments/gettingstarted` folder which should be empty. Set this as your working directory so that RStudio knows you're there too.

The image shows a screenshot of the RStudio interface. The top-left pane shows a file browser with the path `~/Documents/teaching/2019/summer/chdsun` and a folder named `gettingstart`. The bottom-left pane shows a terminal window with the following commands and output:

```
day0 python-appengine-ins
day2 python-appengine-ins
experiment
Amys-MacBook-Pro-3:chdsunsummerschool amy$ cd day2/
Amys-MacBook-Pro-3:day2 amy$ ls
experiments slides
Amys-MacBook-Pro-3:day2 amy$ cd ../experiment/
Amys-MacBook-Pro-3:experiment amy$ ls
exp img samplingfram
images img.zip
```

The top-right pane shows the Environment tab with the message "Environment is empty". The bottom-right pane shows a context menu for the `gettingstart` folder with the following options:

- Copy...
- Copy To...
- Move...
- Set As Working Directory
- Go To Working Directory
- Show Folder in New Window

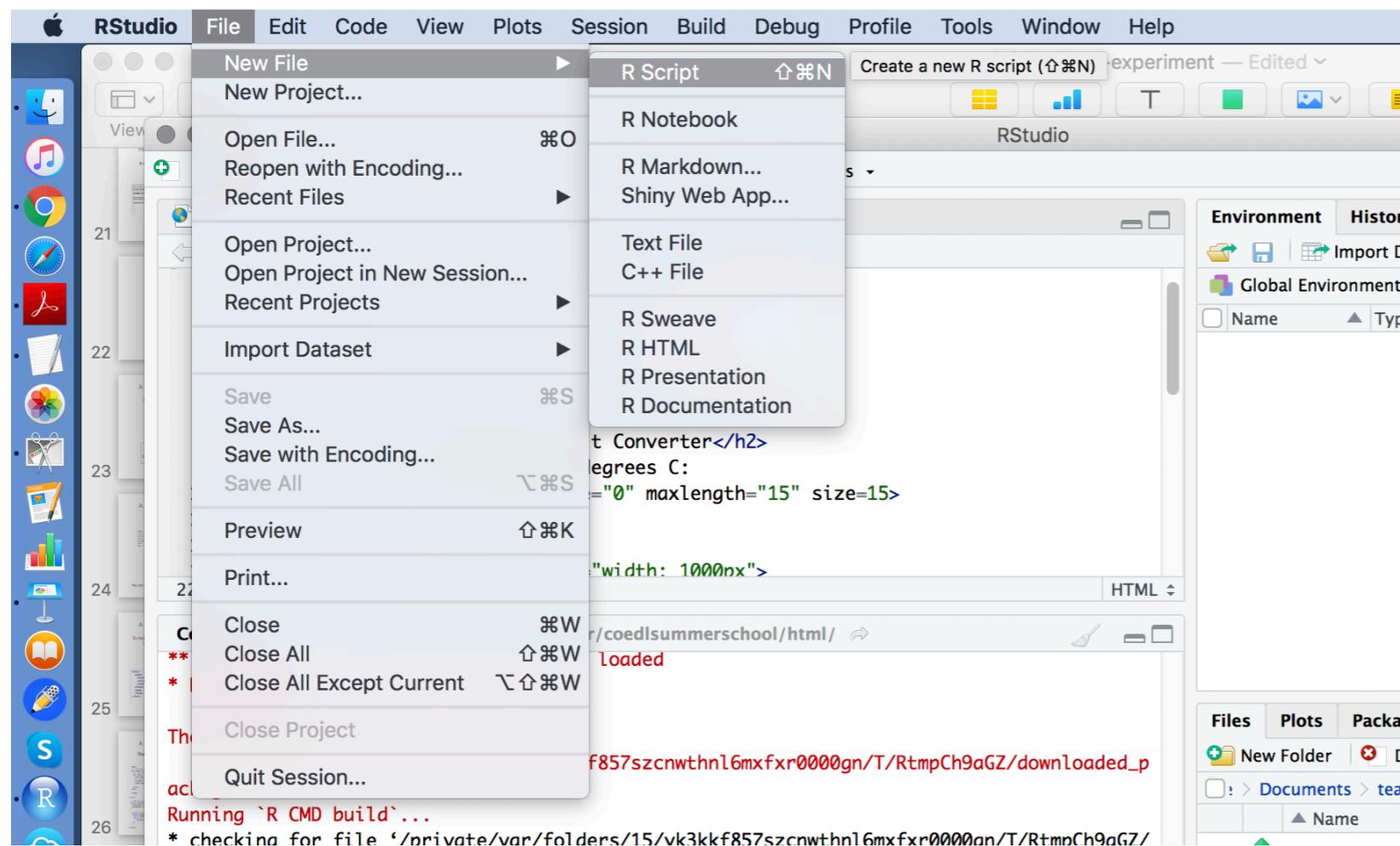
Two annotations with arrows point to the `gettingstart` folder in the file browser and the "Set As Working Directory" option in the context menu.

Navigating in RStudio to that directory

Setting as your working directory

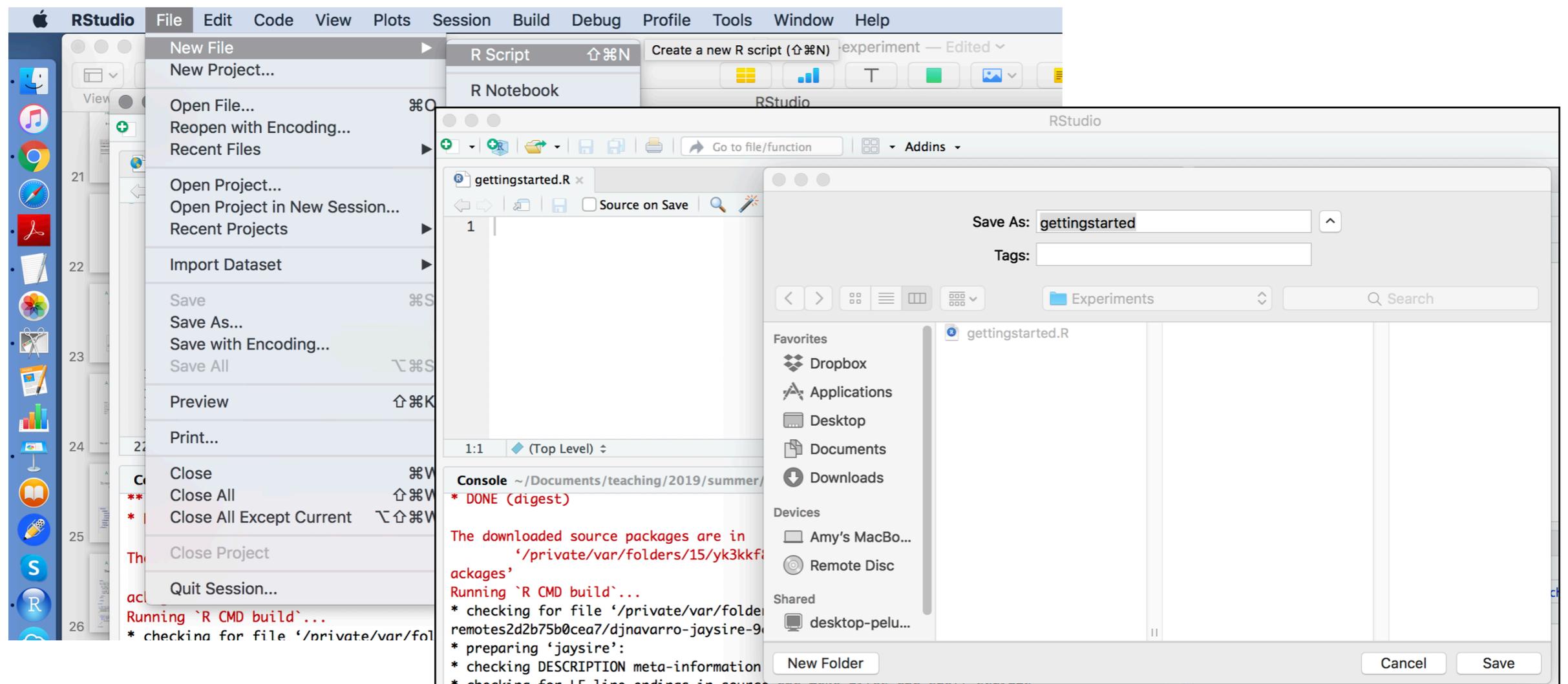
JAYSIRE: GETTING STARTED

Now create a script called `gettingstarted.R` and save it in this location (`summerschool/day2/experiments/gettingstarted`)



JAYSIRE: GETTING STARTED

Now create a script called `gettingstarted.R` and save it in this location (`summerschool/day2/experiments/gettingstarted`)



JAYSIRE: GETTING STARTED

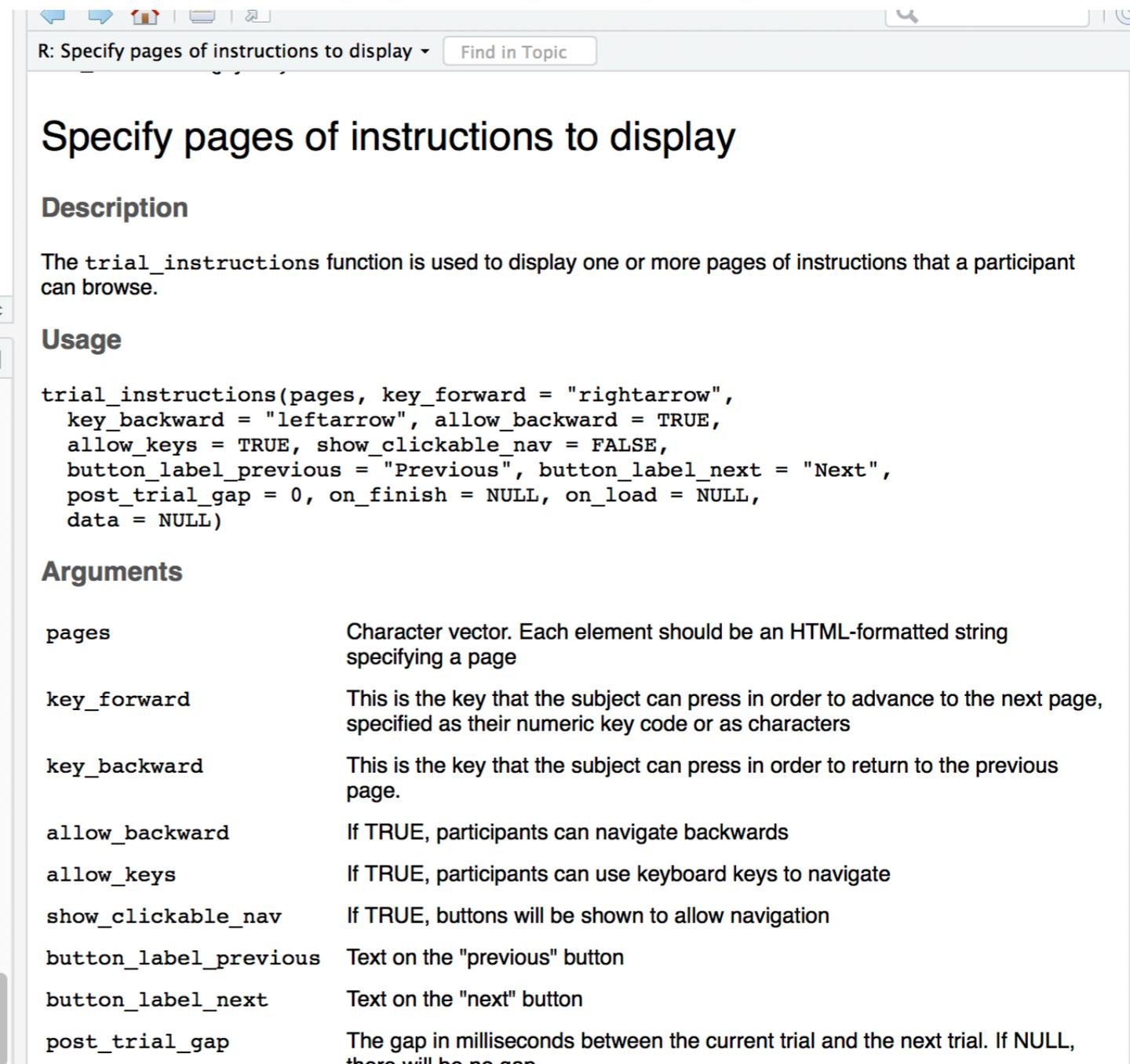
Let's start by creating instructions using the function called `trial_instructions()`

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

```
help(trial_instructions)
```

Takes a bunch of arguments which describe the instructions people will see



R: Specify pages of instructions to display

Specify pages of instructions to display

Description

The `trial_instructions` function is used to display one or more pages of instructions that a participant can browse.

Usage

```
trial_instructions(pages, key_forward = "rightarrow",  
  key_backward = "leftarrow", allow_backward = TRUE,  
  allow_keys = TRUE, show_clickable_nav = FALSE,  
  button_label_previous = "Previous", button_label_next = "Next",  
  post_trial_gap = 0, on_finish = NULL, on_load = NULL,  
  data = NULL)
```

Arguments

<code>pages</code>	Character vector. Each element should be an HTML-formatted string specifying a page
<code>key_forward</code>	This is the key that the subject can press in order to advance to the next page, specified as their numeric key code or as characters
<code>key_backward</code>	This is the key that the subject can press in order to return to the previous page.
<code>allow_backward</code>	If TRUE, participants can navigate backwards
<code>allow_keys</code>	If TRUE, participants can use keyboard keys to navigate
<code>show_clickable_nav</code>	If TRUE, buttons will be shown to allow navigation
<code>button_label_previous</code>	Text on the "previous" button
<code>button_label_next</code>	Text on the "next" button
<code>post_trial_gap</code>	The gap in milliseconds between the current trial and the next trial. If NULL, there will be no gap

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

Putting the instructions into your own variable called instructions

The function name (needs parentheses around arguments)



```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

Putting the instructions into your own variable called instructions

The function name (needs parentheses around arguments)

These are the three arguments to the function:

```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

Putting the instructions into your own variable called instructions

The function name (needs parentheses around arguments)

These are the three arguments to the function:

```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

The two pages are in a vector (the command `c()` does this) separated by commas

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

Putting the instructions into your own variable called instructions

The function name (needs parentheses around arguments)

These are the three arguments to the function:

```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

The two pages are in a vector (the command `c()` does this) separated by commas

Let participants click back and forth

JAYSIRE: GETTING STARTED

Let's start by creating instructions using the function called `trial_instructions()`

Putting the instructions into your own variable called instructions

The function name (needs parentheses around arguments)

These are the three arguments to the function:

```
instructions <- trial_instructions(  
  pages = c(  
    "Welcome! Use the arrow buttons to browse these instructions",  
    "Press the 'Next' button to begin!"  
  ),  
  show_clickable_nav = TRUE,  
  post_trial_gap = 1000  
)
```

The two pages are in a vector (the command `c()` does this) separated by commas

How long to wait after clicking before going to the next page

Let participants click back and forth

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

```
build_experiment(  
    timeline = build_timeline(instructions),  
    path = file.path("starting_exp"),  
    on_finish = fn_save_locally()  
)
```

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

The function name (needs parentheses around arguments)

Builds a timeline consisting only of the instructions we created



```
build_experiment(  
    timeline = build_timeline(instructions),  
    path = file.path("starting_exp"),  
    on_finish = fn_save_locally()  
)
```

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

The function name (needs parentheses around arguments)

Builds a timeline consisting only of the instructions we created

These are the three arguments to the function:

```
build_experiment(  
    timeline = build_timeline(instructions),  
    path = file.path("starting_exp"),  
    on_finish = fn_save_locally()  
)
```

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

The function name (needs parentheses around arguments)

Builds a timeline consisting only of the instructions we created

These are the three arguments to the function:

```
build_experiment(  
  timeline = build_timeline(instructions),  
  path = file.path("starting_exp"),  
  on_finish = fn_save_locally()  
)
```

Tells R to put the experiment in a new[*] folder called **starting_exp** in your current directory

JAYSIRE: GETTING STARTED

Now let's build a minimal experiment with just these instructions

The function name (needs parentheses around arguments)

Builds a timeline consisting only of the instructions we created

These are the three arguments to the function:

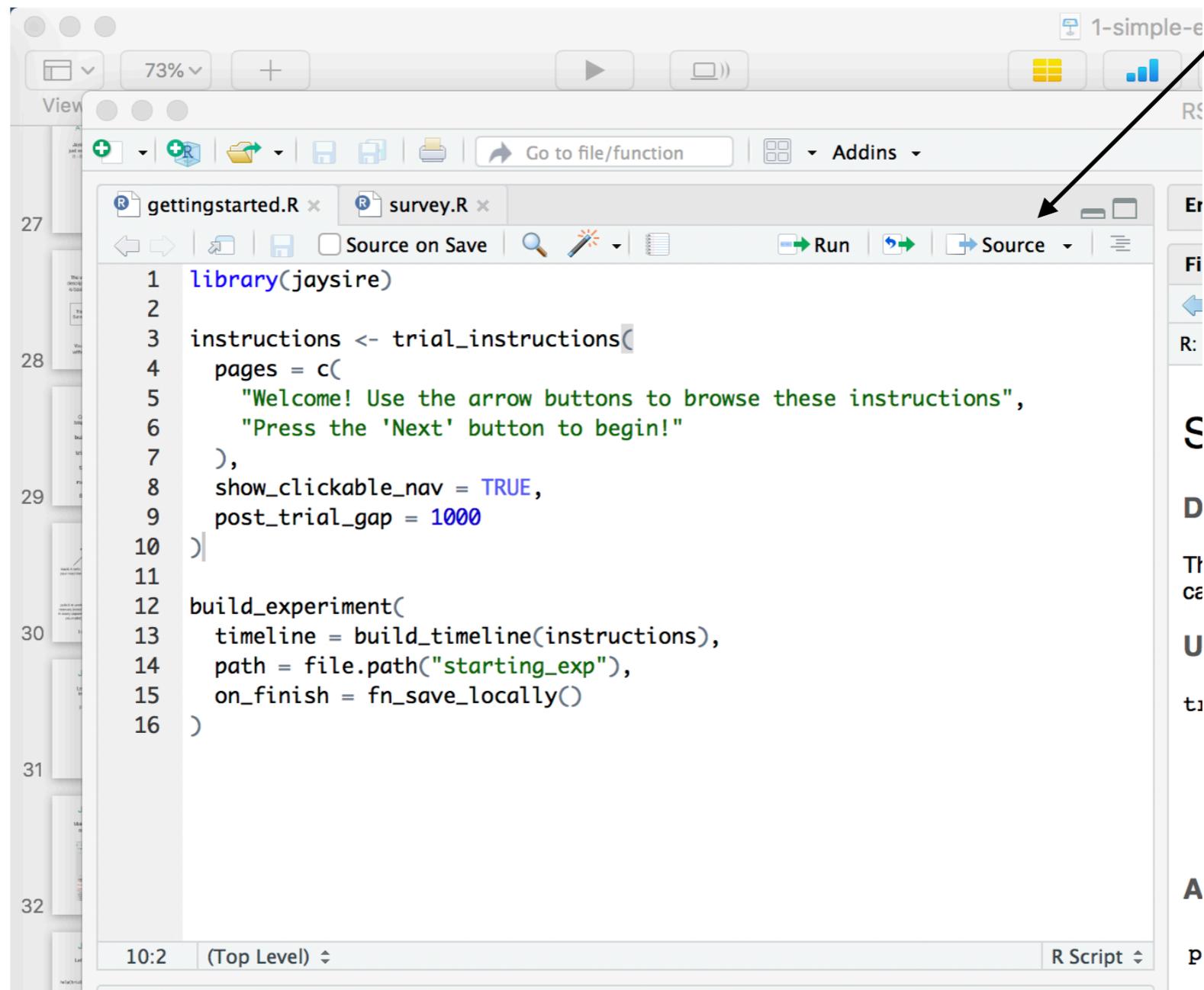
```
build_experiment(  
  timeline = build_timeline(instructions),  
  path = file.path("starting_exp"),  
  on_finish = fn_save_locally()  
)
```

Tells R to save the data locally on your computer

Tells R to put the experiment in a new[*] folder called **starting_exp** in your current directory

JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run



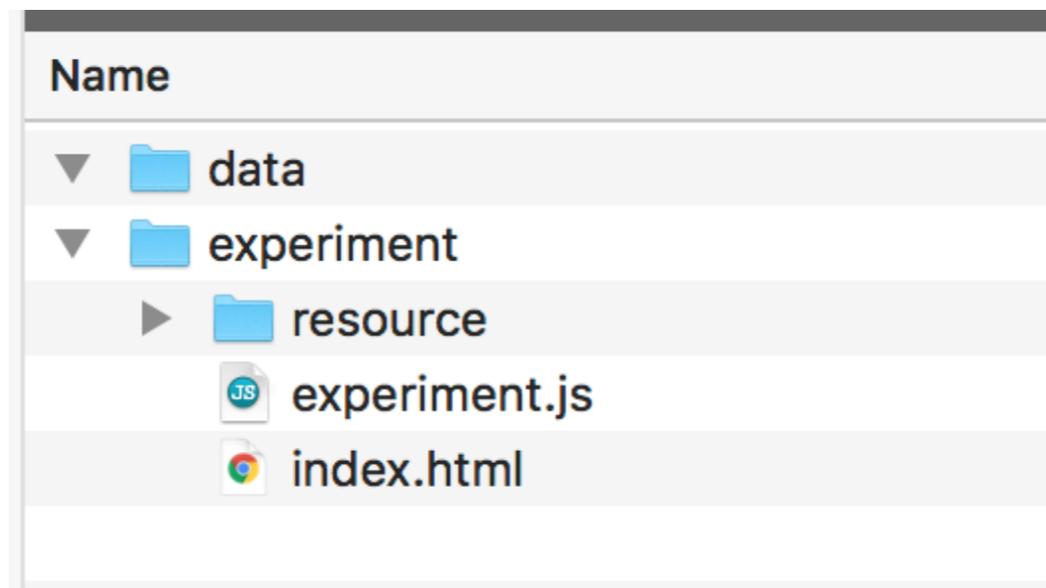
```
1 library(jaysire)
2
3 instructions <- trial_instructions(
4   pages = c(
5     "Welcome! Use the arrow buttons to browse these instructions",
6     "Press the 'Next' button to begin!"
7   ),
8   show_clickable_nav = TRUE,
9   post_trial_gap = 1000
10 )
11
12 build_experiment(
13   timeline = build_timeline(instructions),
14   path = file.path("starting_exp"),
15   on_finish = fn_save_locally()
16 )
```

The screenshot shows the RStudio interface with two tabs: 'gettingstarted.R' and 'survey.R'. The 'gettingstarted.R' tab is active, displaying the R code above. The toolbar at the top of the editor includes buttons for 'Run' and 'Source'. A black arrow points from the text above to the 'Source' button. The status bar at the bottom indicates the cursor is at line 10, column 2, and the file is an R Script.

JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run

You’ll see that this builds the experiment - creates two folders inside `starting_exp`



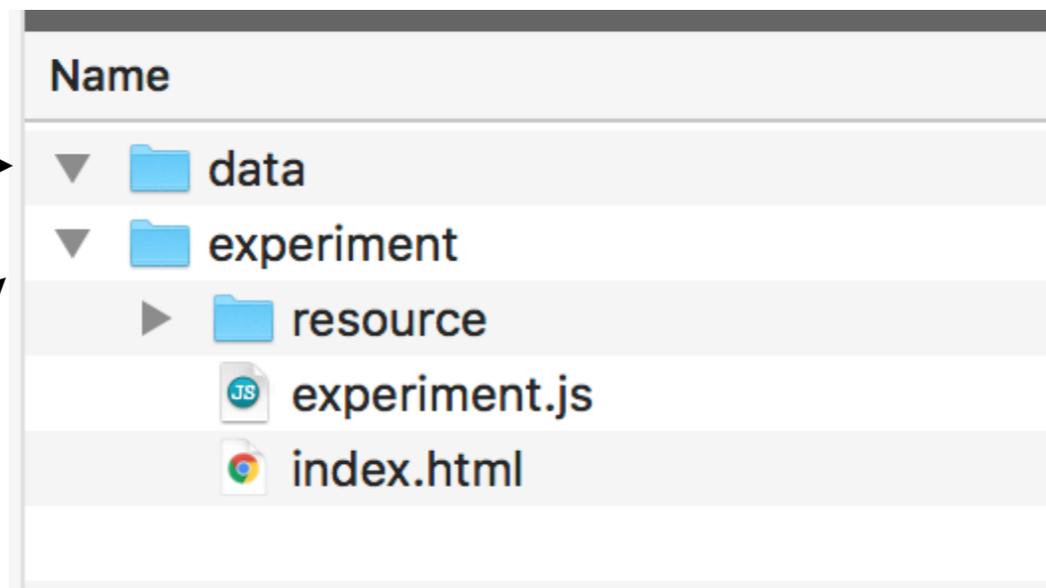
JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run

You’ll see that this builds the experiment - creates two folders inside `starting_exp`

Empty (where data will go)

Javascript and html code for the experiment



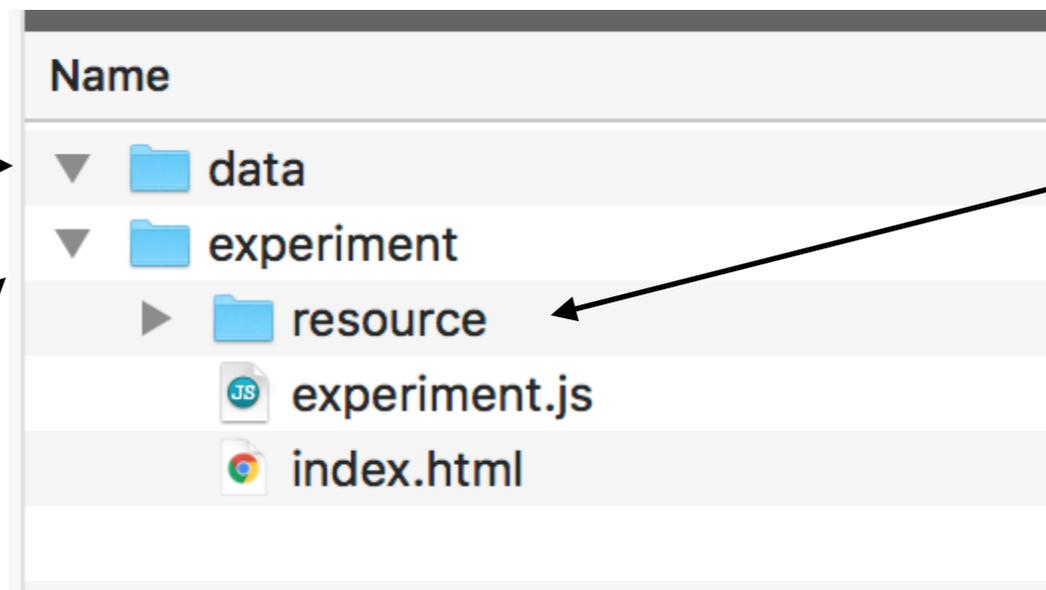
JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run

You’ll see that this builds the experiment - creates two folders inside `starting_exp`

Empty (where data will go)

Javascript and html code for the experiment



Resource files, if any (we don't have any in this)

JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run

You’ll see that this builds the experiment - creates two folders inside `starting_exp`

Name
▼ data
▼ experiment
▶ resource
experiment.js
index.html

Empty (where data will go)

Javascript and html code for the experiment

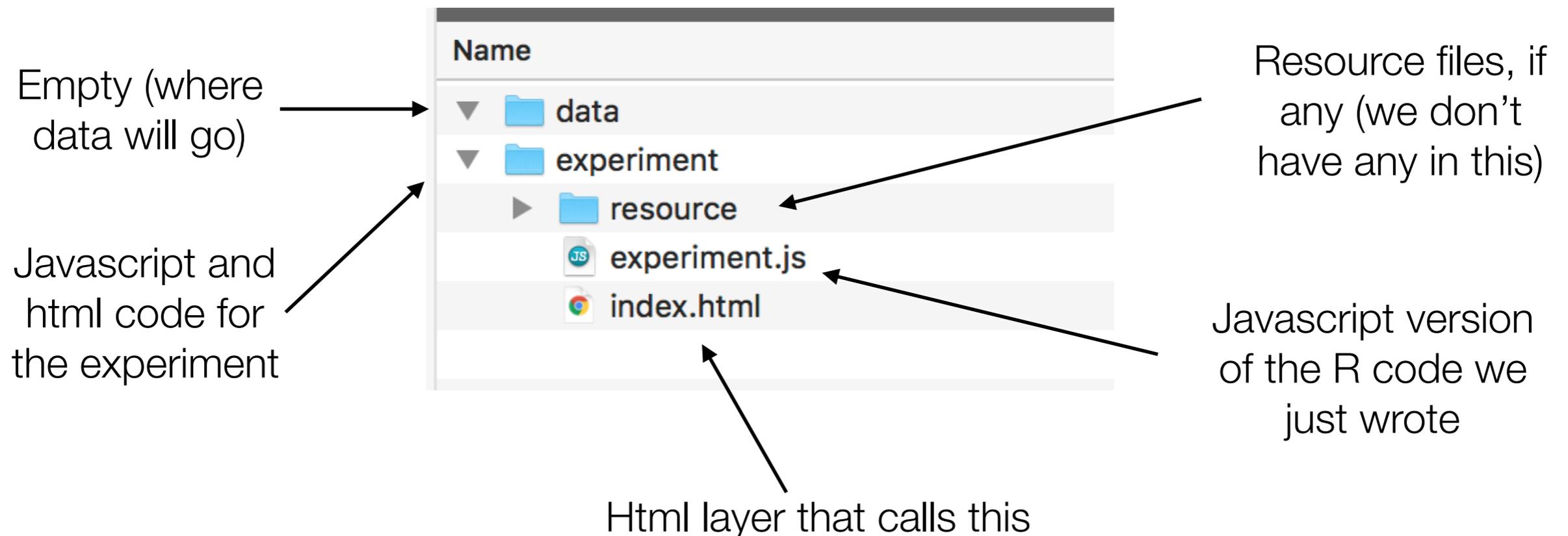
Resource files, if any (we don't have any in this)

Javascript version of the R code we just wrote

JAYSIRE: GETTING STARTED

“Source” the experiment in R to get it ready to run

You’ll see that this builds the experiment - creates two folders inside `starting_exp`



JAYSIRE: GETTING STARTED

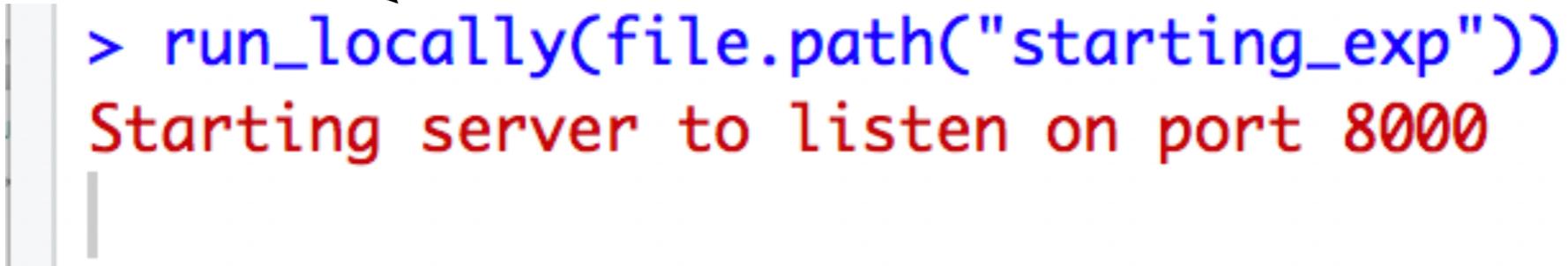
Run it by using the function `run_locally()`.
Type it at the console with your path in as an argument

```
> run_locally(file.path("starting_exp"))  
Starting server to listen on port 8000
```

JAYSIRE: GETTING STARTED

Run it by using the function `run_locally()`.
Type it at the console with your path in as an argument

The function name (needs
parentheses around
arguments)



```
> run_locally(file.path("starting_exp"))  
Starting server to listen on port 8000
```

The image shows a terminal window with a blue prompt character followed by the command `run_locally(file.path("starting_exp"))`. Below the command, the output `Starting server to listen on port 8000` is displayed in red. A vertical grey bar on the left side of the terminal window represents the scrollbar. An arrow from the text above points to the `run_locally` part of the command.

JAYSIRE: GETTING STARTED

Run it by using the function `run_locally()`.
Type it at the console with your path in as an argument

The function name (needs
parentheses around
arguments)

Tells R this is a
path

```
> run_locally(file.path("starting_exp"))  
Starting server to listen on port 8000
```

JAYSIRE: GETTING STARTED

Run it by using the function `run_locally()`.
Type it at the console with your path in as an argument

The function name (needs
parentheses around
arguments)

Tells R this is a
path

Name of the folder with
the experiment

```
> run_locally(file.path("starting_exp"))  
Starting server to listen on port 8000
```

EXERCISE

Try sourcing it again: what happens? Why?

EXERCISE

Try sourcing it again: what happens? Why?

It assumes that `starting_exp` folder doesn't contain any subfolders. Since it does (you've already built it) it throws an error. Can avoid this by deleting the `data` and `experiment` folders each time you source again (tedious) or add in this bit to your code:

```
# set directory (deletes any existing old experiment builds in it)
my_directory <- file.path("starting_exp")
# create the empty folder if necessary
if(dir.exists(my_directory)) {
  unlink(my_directory, recursive = TRUE)
}
```

EXERCISE

Try sourcing it again: what happens? Why?

It assumes that `starting_exp` folder doesn't contain any subfolders. Since it does (you've already built it) it throws an error. Can avoid this by deleting the `data` and `experiment` folders each time you source again (tedious) or add in this bit to your code:

```
# set directory (deletes any existing old experiment builds in it)
my_directory <- file.path("starting_exp")
# create the empty folder if necessary
if(dir.exists(my_directory)) {
  unlink(my_directory, recursive = TRUE)
}
```

And then replace the argument in `build_experiment`:

```
path = my_directory,
```

EXERCISE

Try sourcing it again: what happens? Why?

It assumes that `starting_exp` folder doesn't contain any subfolders. Since it does (you've already built it) it throws an error. Can avoid this by deleting the `data` and `experiment` folders each time you source again (tedious) or add in this bit to your code:

```
# set directory (deletes any existing old experiment builds in it)
my_directory <- file.path("classsurvey_exp")
# create the empty folder if necessary
if(dir.exists(my_directory)) {
  unlink(my_directory, recursive = TRUE)
}
```

And then replace the argument in `build_experiment`:

```
path = my_directory,
```

How do you run it this time?

EXERCISE

Change your instructions so they look like this:

Page 1

Welcome! Use the arrow buttons to browse these instructions

< Backward

Forward >

Page 2

In this experiment you will solve some equations.

It is *very important* that you do your best.

< Backward

Forward >

Page 3

Press the 'Forward' button to begin!

< Backward

Forward >

EXERCISE

Change your instructions so they look like this:

Page 1

Welcome! Use the arrow buttons to browse these instructions

< Backward

Forward >

Page 2

In this experiment you will solve some equations.

It is *very important* that you do your best.

< Backward

Forward >

Page 3

Press the 'Forward' button to begin!

< Backward

Forward >

Hints: remember to use `help(trial_instructions)` to figure out how to change the button labels. And the text in the instruction pages can be formatted using html tags.

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

```
trial1 <- trial_html_button_response(  
  stimulus = "13 + 23 = 36",  
  choices = c("true", "false"),  
  post_trial_gap = 500  
)
```

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

Putting this trial into your own variable called trial1

The function name (needs parentheses around arguments)



```
trial1 <- trial_html_button_response(  
  stimulus = "13 + 23 = 36",  
  choices = c("true", "false"),  
  post_trial_gap = 500  
)
```

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

Putting this trial into your own variable called trial1

The function name (needs parentheses around arguments)

The stimulus shown to the participant

```
trial1 <- trial_html_button_response(  
  stimulus = "13 + 23 = 36",  
  choices = c("true", "false"),  
  post_trial_gap = 500  
)
```

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

Putting this trial into your own variable called trial1

The function name (needs parentheses around arguments)

The stimulus shown to the participant

```
trial1 <- trial_html_button_response(  
  stimulus = "13 + 23 = 36",  
  choices = c("true", "false"),  
  post_trial_gap = 500  
)
```

The button choices and labels

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

Putting this trial into your own variable called trial1

The function name (needs parentheses around arguments)

```
trial1 <- trial_html_button_response(  
  stimulus = "13 + 23 = 36",  
  choices = c("true", "false"),  
  post_trial_gap = 500
```

The stimulus shown to the participant

The button choices and labels

How long to wait after clicking before going to the next page

ADDING A TRIAL

Let's make a trial that presents the stimulus as html and collects a response using buttons:

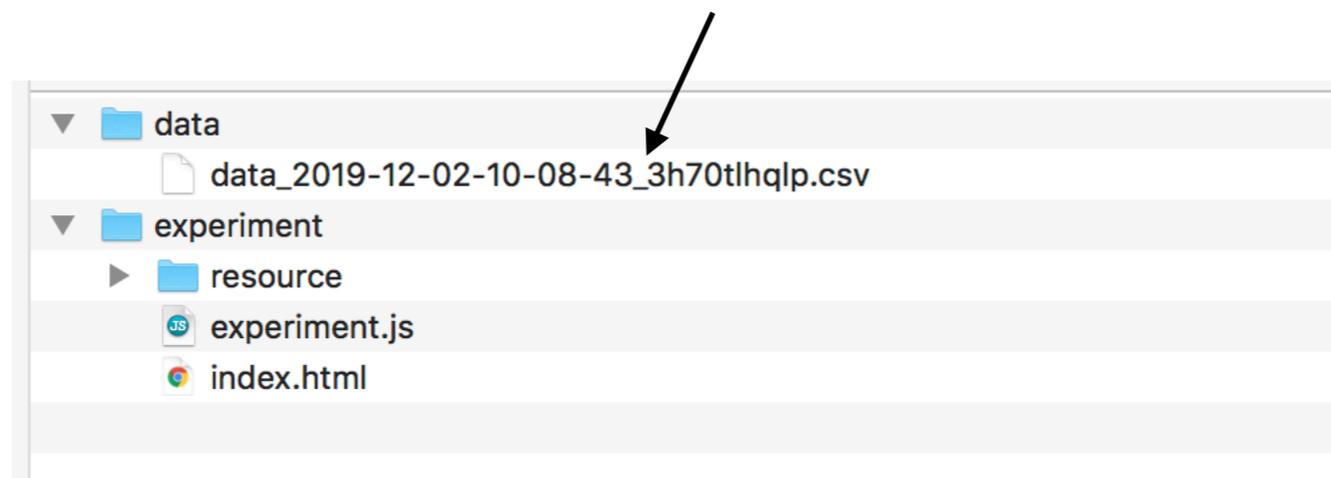
Need to also remember to add it to the timeline!

```
build_experiment(  
    timeline = build_timeline(instructions, trial1),  
    path = my_directory,  
    on_finish = fn_save_locally()  
)
```



ADDING A TRIAL

This time when we run it there is data to save, which goes in the `data` folder



It's saved as a csv (with some things saved in a format called JSON format, which tomorrow we'll learn how to turn into manageable form). Still even now you can see what is there

EXERCISE

Add another trial which shows the equation $2+2=5$ for only 500ms before it disappears.

The participant should have three response options: true, false, and I don't know

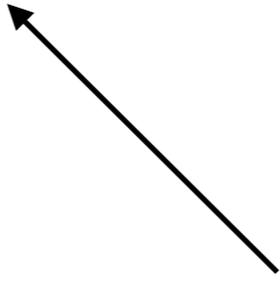
NOW LET'S PUT IT
ONLINE!

FIRST LET'S MODIFY OUR R CODE

Now we need to make it save on the app engine instead of locally

```
build_experiment(  
  timeline = build_timeline(instructions,trial1),  
  path = file.path(my_directory),  
  on_finish = fn_save_datastore()  
)
```

This bit is new



FIRST LET'S MODIFY OUR R CODE

When you source it now, two different files appear in your experiment folder. You don't need to do anything with them; they are for interfacing with the backend of the Google App Engine (GAE)

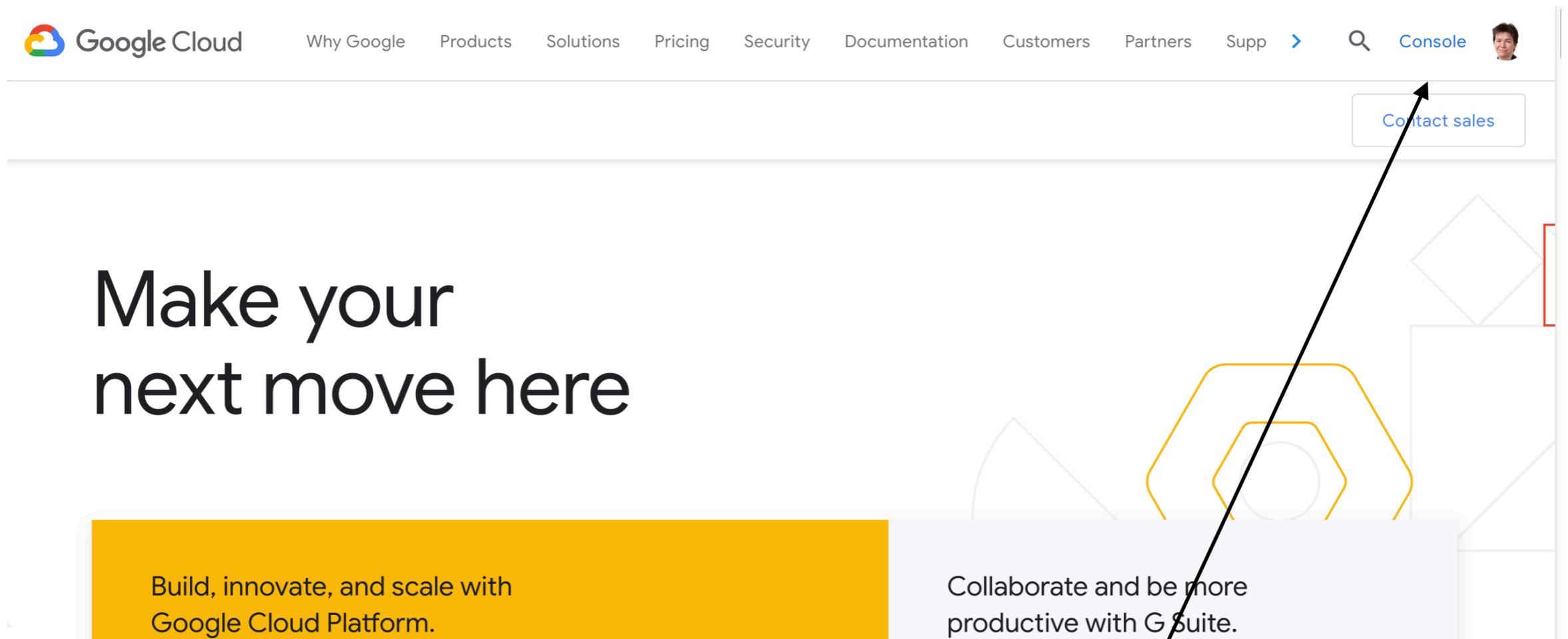


OUTLINE OF STEPS

1. Go to google cloud
2. Make a new project (or go into an existing one if you're just modifying an old one)
3. Open a terminal and go to the command line in the directory with the experiment
4. Initialise google cloud in that directory
5. Deploy the experiment so it shows up online.
6. Go to your project url.

1. GO TO GOOGLE CLOUD

cloud.google.com



Go to console

2. CREATE PROJECT IN GOOGLE CLOUD

The screenshot shows the Google Cloud Platform dashboard for a project named 'qualification-test'. The top navigation bar includes the Google Cloud Platform logo, the project name, a search bar, and various utility icons. Below the navigation bar, there are tabs for 'DASHBOARD' and 'ACTIVITY', and a 'CUSTOMISE' button. The main content area is divided into several sections:

- Project info:** Displays project details such as Project name (qualification-test), Project ID (qualification-test), and Project number (219908920333). A link to 'Go to project settings' is provided.
- Resources:** Lists resources associated with the project, including App Engine (10 versions) and Cloud Storage (2 buckets).
- Trace:** A section for monitoring and debugging.
- App Engine:** A summary of App Engine metrics, including a line chart showing 'http/server/response_count' over time. The chart shows several peaks, with the highest peak reaching approximately 0.150. A link to 'Go to the App Engine dashboard' is provided.
- API APIs:** A section for managing APIs.
- Google Cloud Platform status:** Indicates that all services are normal and provides a link to 'Go to Cloud status dashboard'.
- Billing:** Shows estimated charges for the billing period 1-8 Dec 2018, which are USD \$0.00. A link to 'View detailed charges' is provided.
- Error Reporting:** Indicates that there are no signs of any errors and provides a link to 'Learn how to set up Error Reporting'.

This will list your projects

2. CREATE PROJECT IN GOOGLE CLOUD

The screenshot displays the Google Cloud Platform interface. A modal dialog titled "Select a project" is open, showing a search bar and a list of projects. The "NEW PROJECT" button is highlighted with a black arrow. The background shows the "Project info" section with details for "qualification-test" and a list of resources like App Engine and Cloud Storage.

Select a project

NEW PROJECT

Search projects and folders

RECENT ALL

Name	ID
✓ qualification-test ?	qualification-test
black-swan ?	black-swan-melbourne
Category Learning ?	category-learning
black-swan-unsw ?	black-swan-unsw
Science Decisions ?	science-decisions-adelaide
Statistical Learning ?	statistical-learning-adelaide
Monty Hall Problem ?	mhp-adelaide
Mandarin Phonetic Training ?	learning-language-sounds

CANCEL OPEN

Project info

Project name
qualification-test

Project ID
qualification-test

Project number
219908920333

Go to project settings

Resources

App Engine
10 versions

Cloud Storage
2 buckets

Trace

You can select "new project"

2. CREATE PROJECT IN GOOGLE CLOUD

Google Cloud Platform

New Project

You have 6 projects remaining in your quota. Request an increase or delete projects.
[Learn more](#)
[MANAGE QUOTAS](#)

Project Name *
My Project 86837

Project ID: linen-age-224911. It cannot be changed later. [EDIT](#)

Location *
No organisation [BROWSE](#)

Parent organisation or folder

[CREATE](#) [CANCEL](#)

This is what is going to show up in your url so try to name it something descriptive (for you) but that doesn't give away details you don't want to give away to the participants

2. CREATE PROJECT IN GOOGLE CLOUD

Select a project NEW PROJECT

Search projects and folders

RECENT ALL

Name	ID
chdss-expt ?	chdss-expt
amycoedl ?	amycoedl
visual-statistical-learning ?	visual-statistical-learning
Statistical Learning ?	statistical-learning-adelaide
qualification-test ?	qualification-test
Category Learning ?	category-learning
Science Decisions ?	science-decisions-adelaide
black-swan ?	black-swan-melbourne
black-swan-unsw ?	black-swan-unsw
Monty Hall Problem ?	mhp-adelaide

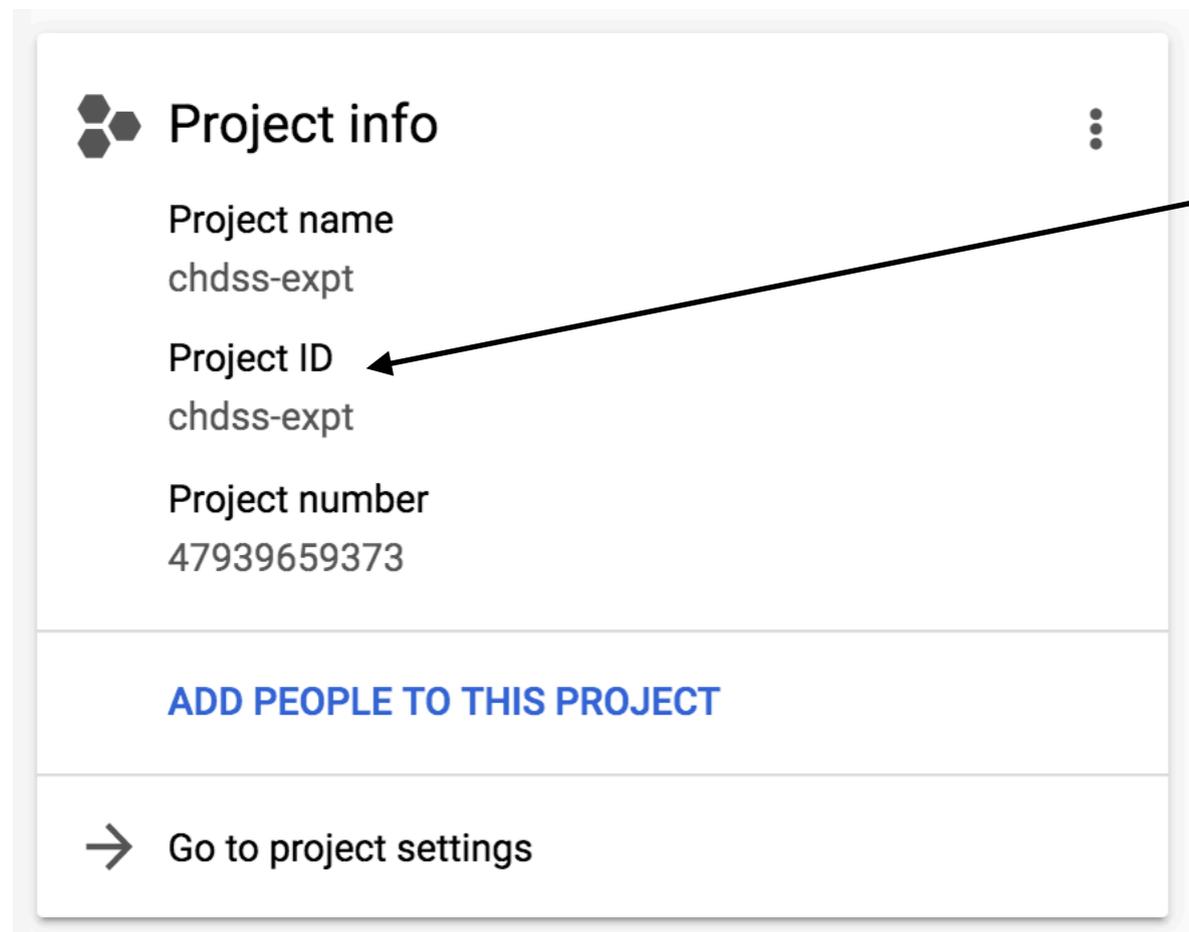
CANCEL OPEN

Now select it from your list

3. DEPLOY THE EXPERIMENT

At the R console type: `run_appengine(path,project_id)`

Your path is my_directory and your project id is your project name:



The screenshot shows a 'Project info' panel with the following details:

- Project name: chdss-expt
- Project ID: chdss-expt
- Project number: 47939659373

Below the info panel is a blue button labeled 'ADD PEOPLE TO THIS PROJECT' and a link with a right-pointing arrow labeled 'Go to project settings'.

`run_appengine(my_directory,"chdss-expt")`

3. DEPLOY THE EXPERIMENT

It will probably give you a message like the following:

To deploy, enter the following command at the console:

```
gcloud app deploy starting_exp/experiment/app.yaml --project=chdss-expt
```

So go to your terminal and type that

```
gcloud app deploy starting_exp/experiment/app.yaml --project=chdss-expt
```

3. DEPLOY THE EXPERIMENT

You are creating an app for project [chdss-expt].

WARNING: Creating an App Engine application for a project is irreversible and the region cannot be changed. More information about regions is at <https://cloud.google.com/appengine/docs/locations>.

Please choose the region where you want your App Engine application located:

- [1] asia-east2 (supports standard and flexible)
- [2] us-west2 (supports standard and flexible)
- [3] asia-northeast2 (supports standard and flexible)
- [4] europe-west6 (supports standard and flexible)
- [5] us-central (supports standard and flexible)
- [6] europe-west3 (supports standard and flexible)
- [7] europe-west2 (supports standard and flexible)
- [8] europe-west (supports standard and flexible)
- [9] us-east1 (supports standard and flexible)
- [10] us-east4 (supports standard and flexible)
- [11] asia-northeast1 (supports standard and flexible)
- [12] asia-south1 (supports standard and flexible)
- [13] australia-southeast1 (supports standard and flexible)
- [14] southamerica-east1 (supports standard and flexible)
- [15] northamerica-northeast1 (supports standard and flexible)
- [16] cancel

Please enter your numeric choice: **13**

3. DEPLOY THE EXPERIMENT

```
Creating App Engine application in project [chdss-expt] and region [australia-southeast1]....done.  
Services to deploy:
```

```
descriptor:      [/Users/amy/Documents/teaching/2019/summer/chdsummerschool/samplingframes/exp/  
experiment/app.yaml]  
source:          [/Users/amy/Documents/teaching/2019/summer/chdsummerschool/samplingframes/exp/  
experiment]  
target project: [chdss-expt]  
target service: [default]  
target version: [20191216t142805]  
target url:      [https://chdss-expt.appspot.com]
```

```
Do you want to continue (Y/n)? Y
```

4. GO TO YOUR EXPERIMENT!

```
Beginning deployment of service [default]...  
Some files were skipped. Pass `--verbosity=info` to see which ones.  
You may also view the gcloud log file, found at  
[/Users/amy/.config/gcloud/logs/2019.12.16/14.26.09.118147.log].
```

```
┌= Uploading 10 files to Google Cloud Storage =┐
```

```
File upload done.  
Updating service [default]...done.  
Setting traffic split for service [default]...done.  
Deployed service [default] to [https://chdss-expt.appspot.com]
```

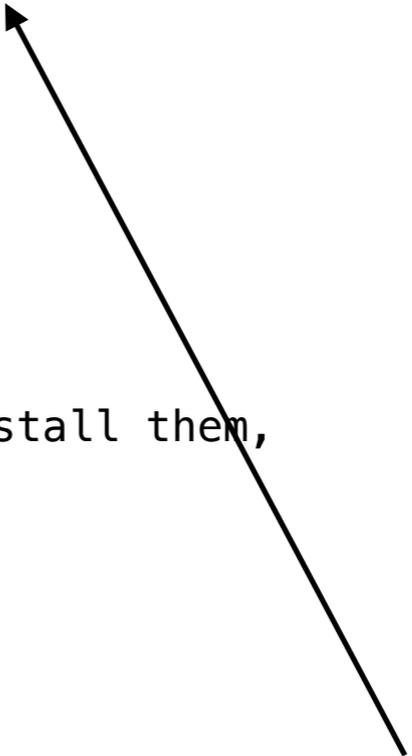
```
You can stream logs from the command line by running:  
$ gcloud app logs tail -s default
```

```
To view your application in the web browser run:  
$ gcloud app browse --project=chdss-expt
```

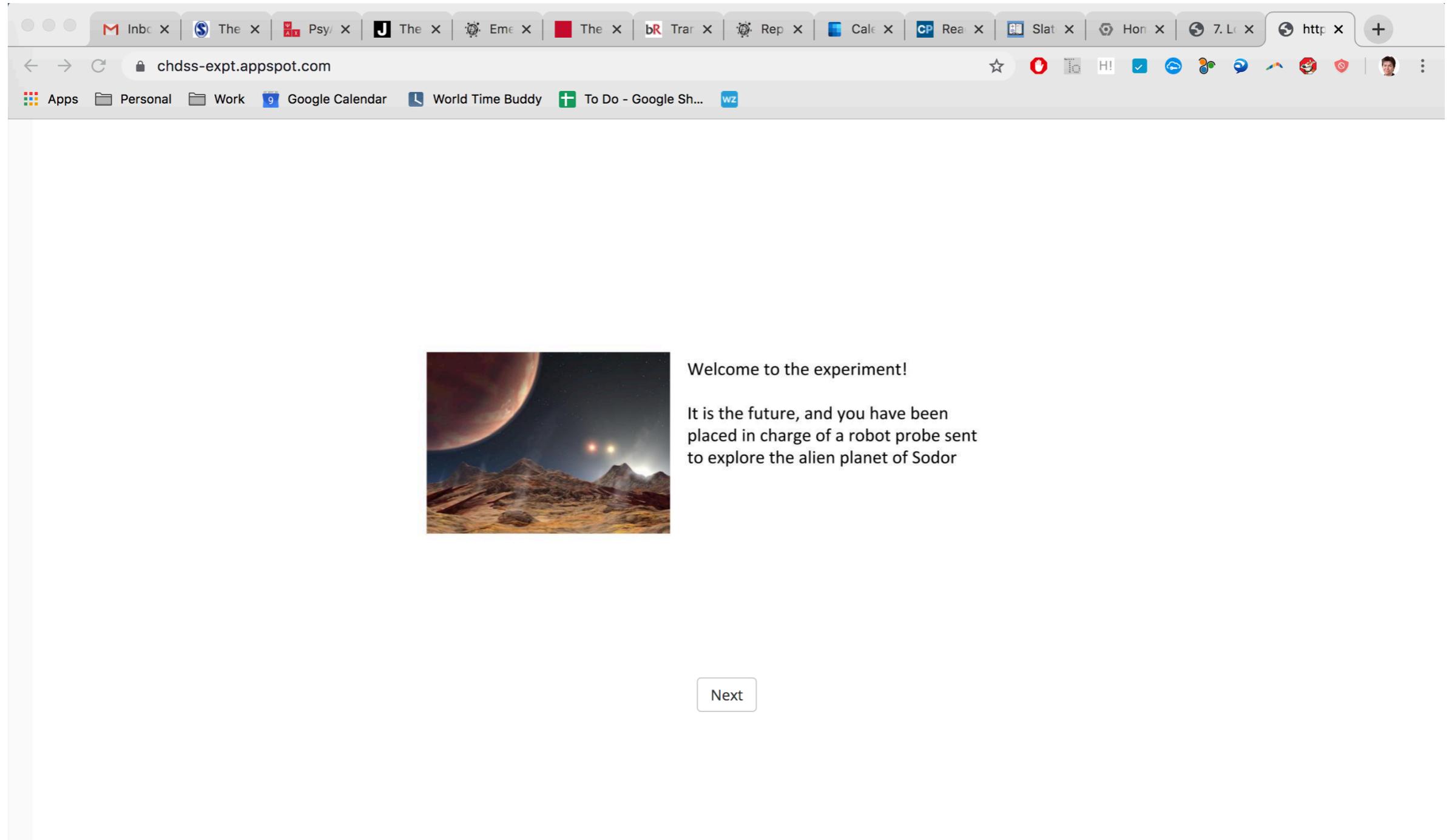
```
Updates are available for some Cloud SDK components. To install them,  
please run:  
$ gcloud components update
```

```
Amys-MacBook-Pro-3:samplingframes amy$
```

There is your url!



4. GO TO YOUR EXPERIMENT!



The screenshot shows a web browser window with the address bar displaying "chdss-expt.appspot.com". The browser's tab bar contains several open tabs, including "Inbc", "The", "Psy", "The", "Eme", "The", "Trar", "Rep", "Calc", "Rea", "Slat", "Hon", "7. L", and "http". The browser's toolbar includes a star icon, a red hand icon, a "H!" icon, a checkmark icon, a blue cloud icon, a multi-colored circle icon, a speech bubble icon, a rainbow icon, a red shield icon, and a user profile icon. Below the toolbar, there are several application icons: "Apps", "Personal", "Work", "Google Calendar", "World Time Buddy", "To Do - Google Sh...", and "WZ".

The main content area of the browser displays a welcome message for an experiment. On the left, there is a square image showing a landscape with mountains and a large, reddish planet in the sky. To the right of the image, the text reads:

Welcome to the experiment!

It is the future, and you have been placed in charge of a robot probe sent to explore the alien planet of Sodor

Below the text, there is a button labeled "Next".

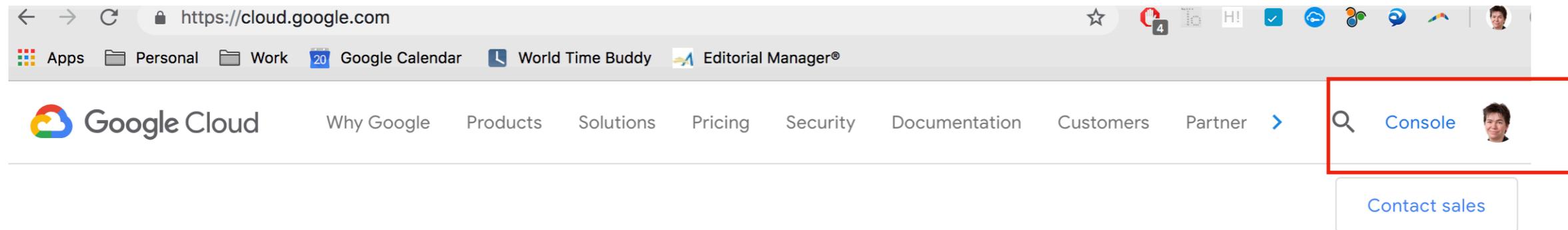
WHAT IF YOU MAKE CHANGES?

Save and source

Use the `run_appengine()` function to redeploy

VIEW YOUR DATA

You can see your data coming in by looking in the google cloud console online



Make your next move here

Build, innovate, and scale with

Collaborate and be more

VIEW YOUR DATA

The screenshot shows the Google Cloud Platform console interface. The top navigation bar includes the Google Cloud Platform logo, the user name 'amycoedl', and a search icon. The left-hand navigation menu is open, displaying various service categories: Home, Security, COMPUTE (App Engine, Compute Engine, Kubernetes Engine, Cloud Functions, Cloud Run), and STORAGE (Bigtable, Datastore, Firestore). The 'Bigtable' and 'Datastore' items are highlighted with a red rectangular box. The main content area is dimmed, showing a monitoring chart with a legend for 'http/server/response_count: 0' and a button to 'Go to the App Engine dashboard'.

Google Cloud Platform amycoedl

Home

Pins appear here ?

Security

COMPUTE

App Engine

Compute Engine

Kubernetes Engine

Cloud Functions

Cloud Run

STORAGE

Bigtable

Datastore

Firestore

The data is kept in your datastore (make sure you're in the right project!)

10:15 10:30 10:45 11:00

0 0.05 0.10 0.15

http/server/response_count: 0

http/server/response_count: 0

Go to the App Engine dashboard

API APIs

Requests (requests/sec)

VIEW YOUR DATA

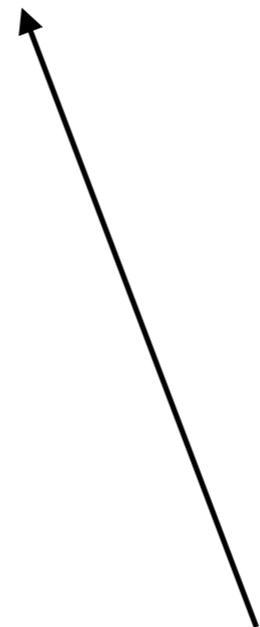
Google Cloud Platform chdss-expt

Entities [+ CREATE ENTITY](#) [↓ IMPORT](#) [↑ EXPORT](#) [🗑 DELETE](#)

[QUERY BY KIND](#) [QUERY BY GQL](#)

Kind: [FILTER ENTITIES](#)

<input type="checkbox"/>	Name/ID ↑	content	date	exp
<input type="checkbox"/>	id=5629499534213120	"rt","stimulus","button_pressed","trial_type","tri...	2019-12-16 (14:37:47.363) AEDT	



There it is!

DOWNLOADING DATA IS EASY

Go to your url and add on `/info`— it will automatically download a csv file called `results`

* Note that this will probably change in future versions of jaysire, because it creates a security issue (anybody can do this). It's still in development and we ran out of time and figured it would be best to just get something working for now. The documentation for jaysire will reflect any changes that are made