Creating an online experiment using jsPsych

Vijay Marupudi

<vijaymarupudi@gatech.edu>

Georgia Tech

2024-10-07

Why create an online (jsPsych) experiment?

- Enables data collection from a larger sample
- Increases sample diversity[1]
- Collect reaction time data
- Allows for shorter experiments
- Facilitates online demos and citizen science
- Reproducibility and replicability, websites from 1995 mostly look and function the same

[1] Henrich, Heine, and Norenzayan 2010; Syed 2021.

Limitations of creating online (jsPsych) experiments

- Timing precision is limited to 2 milliseconds for security reasons[2]
- Requires a little knowledge of multiple languages
  - Javascript (JS)
  - HTML
  - CSS
- Uploading data to a server is more effort than saving locally
  - Not too hard though!
- Data quality can vary based on recruitment platform and attention checks[3]
- Integrating with physical devices (eye-trackers, physical slides, etc.) is harder

[2] *High Resolution Time API | Can I Use... Support Tables for HTML5, CSS3, Etc* 2024.

[3] Chmielewski and Kucker 2020; Douglas, Ewell, and Brauer 2023.

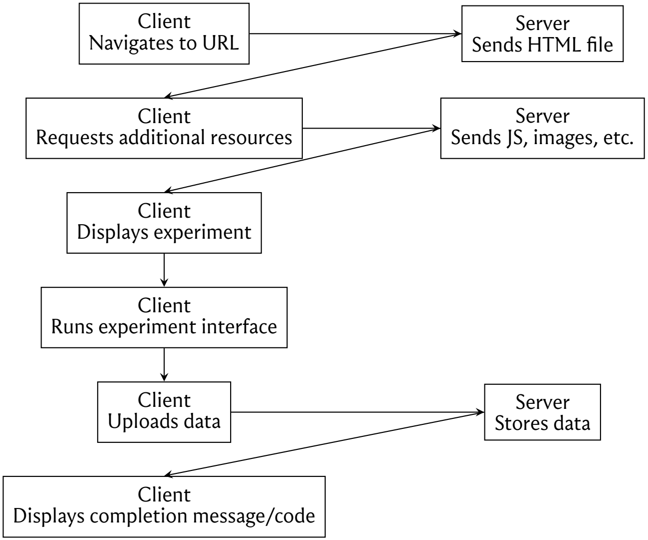
Terminology

- **Client**
  - the participant — what is run and displayed in the web browser
  - Involves HTML, CSS, and Javascript (JS)
- **HTML**: Text that describes the structure of a page
- **CSS**: Text that declares how the page should look
- **Javascript**: Code that outlines how HTML and/or CSS should change in response to events (keypress, clicks, timers, etc.)
- **Server**: Determines what is provided to the client when it requests a URL

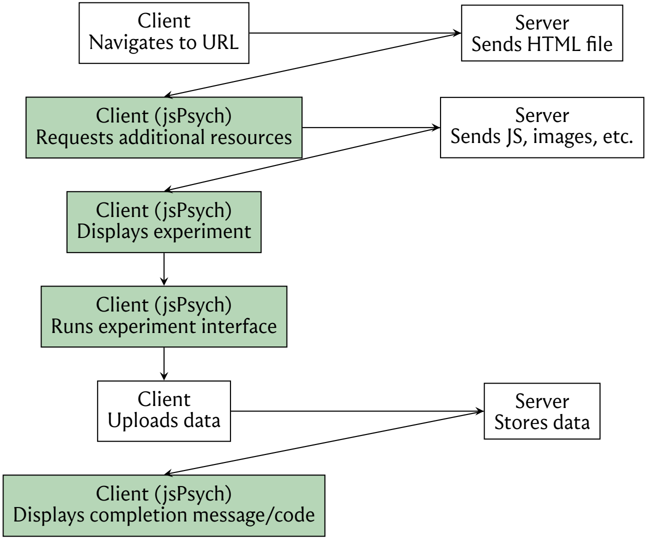
What is jsPsych?

- A *Javascript* library that makes it easy to display things to participants and collect their responses
- Takes care of most of the Javascript, HTML, and CSS, you just need to provide the custom parts necessary for your experiment
- Give jsPsych a list of trials and the data you want to collect, and it will take care of the plumbing

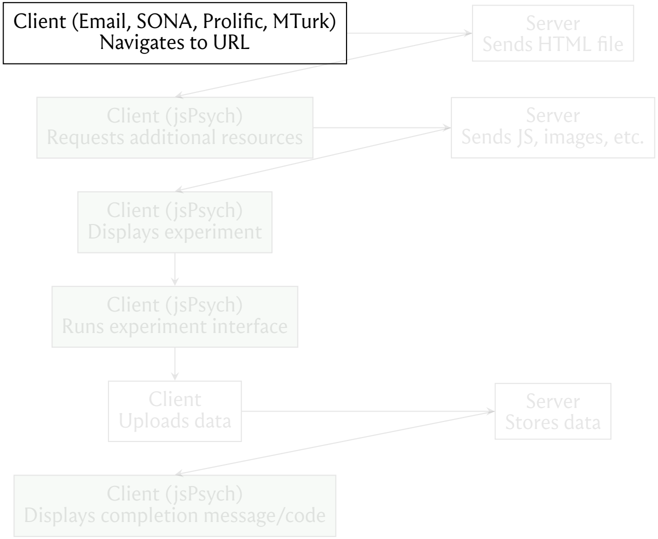
Sequence of an online experiment



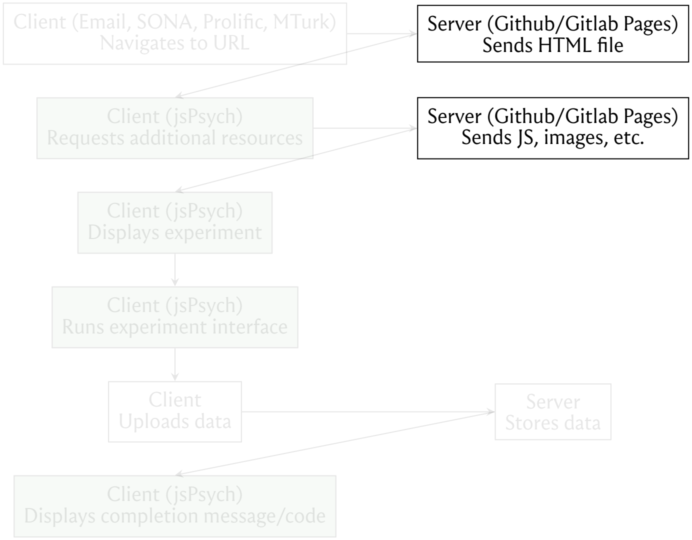
What jsPsych can help with



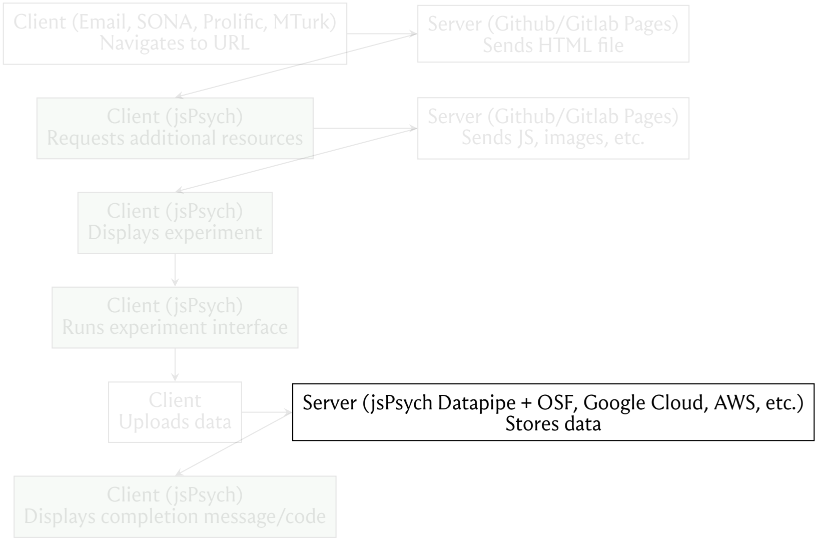
Other services: Recruitment platforms



Other services: Resource hosting server


Client (Email, SONA, Prolific, MTurk)
Navigates to URL
Server (Github/Gitlab Pages)
Sends HTML file
Client (jsPsych)
Requests additional resources
Server (Github/Gitlab Pages)
Sends JS, images, etc.
Client (jsPsych)
Displays experiment
Client (jsPsych)
Runs experiment interface
Client
Uploads data
Server
Stores data
Client (jsPsych)
Displays completion message/code

Other services: Data saving server


Client (Email, SONA, Prolific, MTurk)
Navigates to URL
Server (Github/Gitlab Pages)
Sends HTML file
Client (jsPsych)
Requests additional resources
Server (Github/Gitlab Pages)
Sends JS, images, etc.
Client (jsPsych)
Displays experiment
Client (jsPsych)
Runs experiment interface
Client
Uploads data
Server (jsPsych Datapipe + OSF, Google Cloud, AWS, etc.)
Stores data
Client (jsPsych)
Displays completion message/code

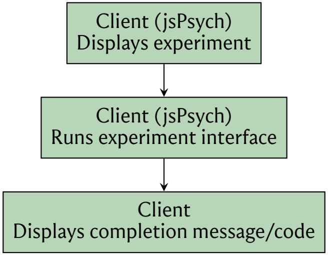
External resources

- Recruitment platforms: Prolific[4]
- Resource hosting server: Github Pages[5]
- Data saving server: jsPsych DataPipe[6] + OSF[7]

Designing the jsPsych experiment is the most complex part!
The rest are relatively straightforward.

[4] https://www.prolific.com/
[5] https://pages.github.com/
[6] https://pipe.jspsych.org/getting-started
[7] https://osf.io/

Today: Getting the jsPsych interface up



HTML: Hello world! [e1]

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
  </head>
  <body>
  <p>Hello world!</p>
  <script>
  </script>
  </body>
</html>
```

- A HTML document has a head and a body
- The head can have a title (shown in the tab)
- The body here has a paragraph that says “Hello world!”
- The body can have a script tag to include Javascript

Using jsPsych in an HTML file

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="https://unpkg.com/jspsych@8.0.2"></script>
    <link href="https://unpkg.com/jspsych@8.0.2/css/jspsych.css"
          rel="stylesheet" type="text/css" />

    <script src="https://unpkg.com/@jspsych/plugin-html-keyboard-response@2.0.0">
    </script>

  </head>
  <body>
    <script></script>
  </body>
</html>
```

- Instructs the browser to use the jsPsych library from the URL
- Instructs it to use the CSS for the jsPsych library from the URL
- The last addition loads a “plugin” from the url
- You can find these urls in the jsPsych website

jsPsych: Hello world [e2]

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="https://unpkg.com/jspsych@8.0.2"></script>
    <script src="https://unpkg.com/@jspsych/plugin-html-keyboard-response@2.0.0">
    </script>
    <link href="https://unpkg.com/jspsych@8.0.2/css/jspsych.css"
          rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script>
      const jsPsych = initJsPsych();

      const hello_trial = {
        type: jsPsychHtmlKeyboardResponse,
        stimulus: 'Hello world!'
      }

      jsPsych.run([hello_trial]);
    </script>
  </body>
</html>
```

- Initialize jsPsych and store it in a variable
- Create a trial object, give it a type and relevant inputs
- Tell jsPsych to run the trial object we made, by providing it a list with the object in it

Where are these functions coming from? jsPsych object

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="https://unpkg.com/jspsych@8.0.2"></script>
    <script src="https://unpkg.com/@jspsych/plugin-html-keyboard-response@2.0.0">
    </script>
    <link href="https://unpkg.com/jspsych@8.0.2/css/jspsych.css"
          rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script>
      const jsPsych = initJsPsych();

      const hello_trial = {
        type: jsPsychHtmlKeyboardResponse,
        stimulus: 'Hello world!'
      }

      jsPsych.run([hello_trial]);
    </script>
  </body>
</html>
```

Where are these functions coming from? The trial type

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="https://unpkg.com/jspsych@8.0.2"></script>
    <script src="https://unpkg.com/@jspsych/plugin-html-keyboard-response@2.0.0">
    </script>
    <link href="https://unpkg.com/jspsych@8.0.2/css/jspsych.css"
          rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script>
      const jsPsych = initJsPsych();

      const hello_trial = {
        type: jsPsychHtmlKeyboardResponse,
        stimulus: 'Hello world!'
      }

      jsPsych.run([hello_trial]);
    </script>
  </body>
</html>
```

What’s the CSS doing?

```html
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="https://unpkg.com/jspsych@8.0.2"></script>
    <script src="https://unpkg.com/@jspsych/plugin-html-keyboard-response@2.0.0">
    </script>
    <link href="https://unpkg.com/jspsych@8.0.2/css/jspsych.css"
          rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script>
      const jsPsych = initJsPsych();

      const hello_trial = {
        type: jsPsychHtmlKeyboardResponse,
        stimulus: 'Hello world!'
      }

      jsPsych.run([hello_trial]);
    </script>
  </body>
</html>
```

- Things are centered in the screen
- The font that’s being used is different
- Other plugins use things in the CSS url to make things look the right way

Mostly showing just the Javascript now

- Most of the edits to the HTML will mostly be adding plugins
- The Javascript part is where most of the logic of the experiment lies

What does the data look like? [e3]

```
const jsPsych = initJsPsych({
  on_finish: function() {
    jsPsych.data.displayData('json')
  }
})

const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: 'Hello world!'
}

jsPsych.run([hello_trial]);
```

- Do not use the CSV format, you will lose data if the return format is complex

What does the data look like?

```
[
  {
    "rt": 1438,
    "stimulus": "Hello world!",
    "response": "k",
    "trial_type": "html-keyboard-response",
    "trial_index": 0,
    "plugin_version": "2.0.0",
    "time_elapsed": 1441
  }
]
```

- Key value pairs, like an R `list` or a Python `dict`
- jsPsych data is a list of objects, each of which represents the data from a trial

What else can jsPsych plugins do?

- Displaying stuff: Images, Audio, HTML, Multiple Choice, Likert, Forms, etc.
- Types of responses: Text, Sliders, Audio, Keyboard keys, Buttons, etc.
- Many, many more: `https://www.jspsych.org/latest/plugins/list-of-plugins/`

What to do when jsPsych plugins aren’t enough

- html-response plugins are very flexible, you’ll just need to provide it the HTML
- Otherwise, you need to make or edit your own plugin, which requires some Javascript knowledge
    - You usually only need to do this if you need a new response method or custom graphics

Let’s make a more complicated experiment

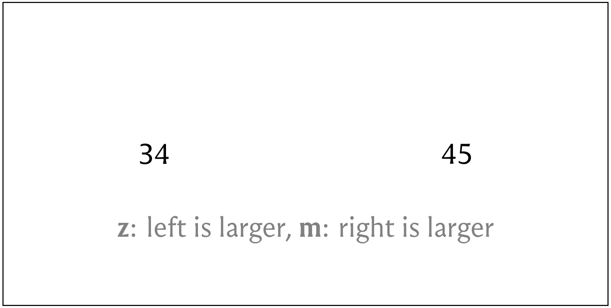
- Magnitude comparison: you see two numbers and you’re asked to pick which one is larger.
- Example: 34   45
- Press ’z’ or ’m’ for the left or right answer
- Allow no other keys
- 10 trials
- Let’s generate the two numbers randomly
  - You can import these from other formats by converting them to JSON using R

How to accomplish this?

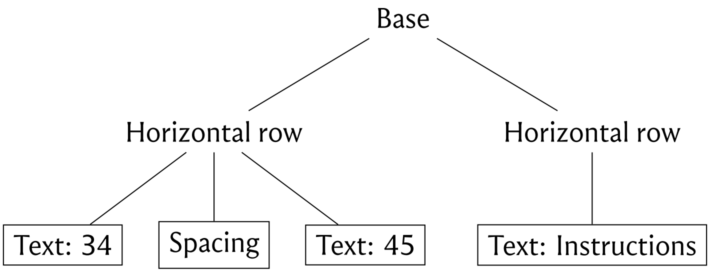
- Is there a plugin for this specific task?
  - Nope :(
- Can we use a HTML plugin for this?
  - Can we display the stimulus using HTML? Yes!
  - Is there a html-...-response plugin that collects the data we want?
    - Yes, `jspsych-html-keyboard-response`!

Designing the visuals in HTML

What do we want a trial to look like?

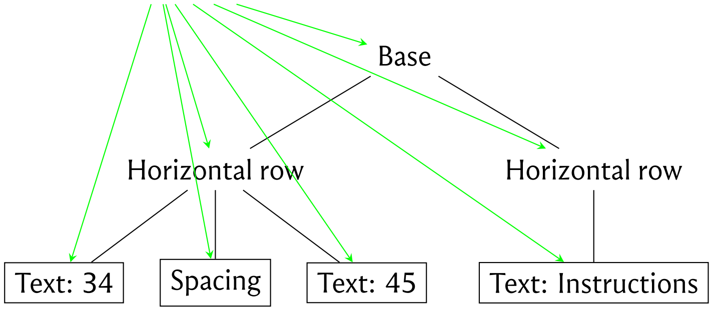


Structure of the HTML



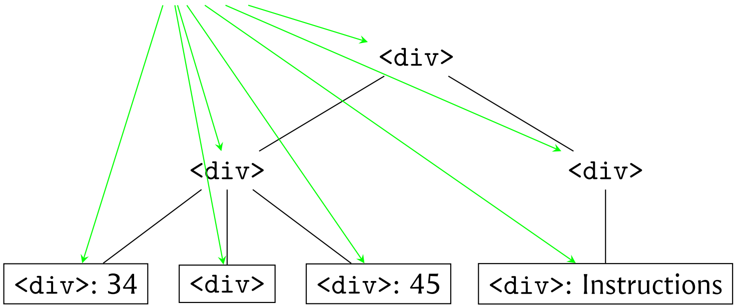
Structure of the HTML

Tip: `use <div>` for almost everything!



Structure of the HTML

Tip: use `<div>` for almost everything!



Structure of the HTML

```
<div>
  <div><div>34</div><div></div><div>45</div></div>
  <div>
    <div>z: left is larger, m: right is larger</div>
  </div>
</div>
```

Structure of the HTML

```
<div>
  <div>
    <div>34</div>
    <div></div>
    <div>45</div>
  </div>
  <div>
    <div>z: left is larger, m: right is larger</div>
  </div>
</div>
```

Using CSS to change how things look

- Just HTML is not enough, the layout is not right
- CSS is needed to make it look the way we want
- You can add inline CSS to HTML elements using `style="..."`

Use `https://flems.io` to test things out!

Using CSS to change how things look

- Use `display: flex;` (CSS Flexbox) to make an element a row
- Use `justify-content: center;` to visually center row elements

```
<div style="display: flex; justify-content:
center;">
```

- Use `width: 50vw;` to add width to the spacing element
- Flexbox ignores width, unless we say `flex-shrink: 0;`

```
<div style=" width: 50vw; flex-shrink: 0;">
```

Structure of the HTML

```html
<div>
  <div style="display: flex; justify-content: center;">
    <div>34</div>
    <div style="width: 50vw; flex-shrink: 0;"></div>
    <div>45</div>
  </div>
  <div>
    <div>
      z: left is larger, m: right is larger
    </div>
  </div>
</div>
```

Make instructions look better

- Align text: `text-align: center;`
- Add space to the top: `margin-top: 200px;`
- Change the text color: `color: grey;`

Structure of the HTML

```
<div>
  <div style="display: flex; justify-content: center;">
    <div>34</div>
    <div style="width: 50vw; flex-shrink: 0;"></div>
    <div>45</div>
  </div>
  <div>
    <div style="text-align: center; margin-top: 200px;
                color: grey;">
      z: left is larger, m: right is larger
    </div>
  </div>
</div>
```

Make text of the first row much bigger

- Add `font-size: 80px;` to the row
- The children of HTML elements inherit CSS properties

Structure of the HTML, done! [e4]

```
<div>
  <div style="display: flex; justify-content: center;
              font-size: 80px;">
    <div>34</div>
    <div style="width: 50vw; flex-shrink: 0;"></div>
    <div>45</div>
  </div>
  <div>
    <div style="text-align: center; margin-top: 200px;
                color: grey;">
      z: left is larger, m: right is larger
    </div>
  </div>
</div>
```

Making a Javascript function that plugs in two numbers into the HTML

```
function makeCompStimulus(left, right) {
  return `
<div style="display: flex; justify-content: center;
             font-size: 80px;">
   <div>${left}</div>
   <div style="width: 50vw; flex-shrink: 0;"></div>
   <div>${right}</div>
 </div>
 <div>
   <div style="text-align: center; margin-top: 200px;
               color: grey;">
     z: left is larger, m: right is larger
   </div>
 </div>`
}
```

Use this for a jsPsych trial! [e5]

```
const jsPsych = initJsPsych({
  on_finish: function() {
    jsPsych.data.displayData('json')
  }
});

const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: makeCompStimulus(34, 45)
}

jsPsych.run([hello_trial]);
```

Voila, magnitude comparison! not done yet...

Fixing small issues

- Restrict the keys participants can press to advance
- **Save information about which numbers are being compared**

Good single trial experiment [e6]

```
const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: makeCompStimulus(34, 45),
  choices: ['z', 'm'],
  data: {leftNumber: 34, rightNumber: 45}
}
```

Generalizing to more (10) trials [e7]

```
const jsPsych = initJsPsych({
  on_finish: function() {
    // the data once the experiment completes
    jsPsych.data.get().localSave('json','mydata.json');
  }
});
```

- Download the data at the end instead of displaying it
  - **Always save data as JSON, it makes life much easier during data processing**
  - Saves the data types in addition to the data
  - Handles nested data
- Generalize to 10 trials, add trial objects to a list and run it
- Add pre-trial instructions and a post-trial message
- Give it a shot, experience the distance, ratio, and place value effects

Loading the data in R

```r
# install.packages("jsonlite")
df = jsonlite::read_json("mydata.json", simplifyVector = TRUE)
df
```

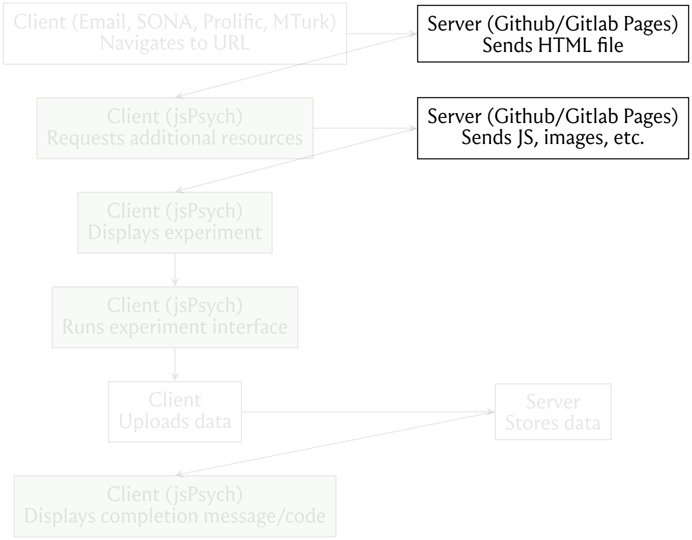
- The `jsonlite` package reads JSON files into nice dataframes
- Now you can process and clean the data in R

More polish required

- Consent forms
- Instructions
- Timeouts
- Surveys
- ...

jsPsych has plugins for all of these!

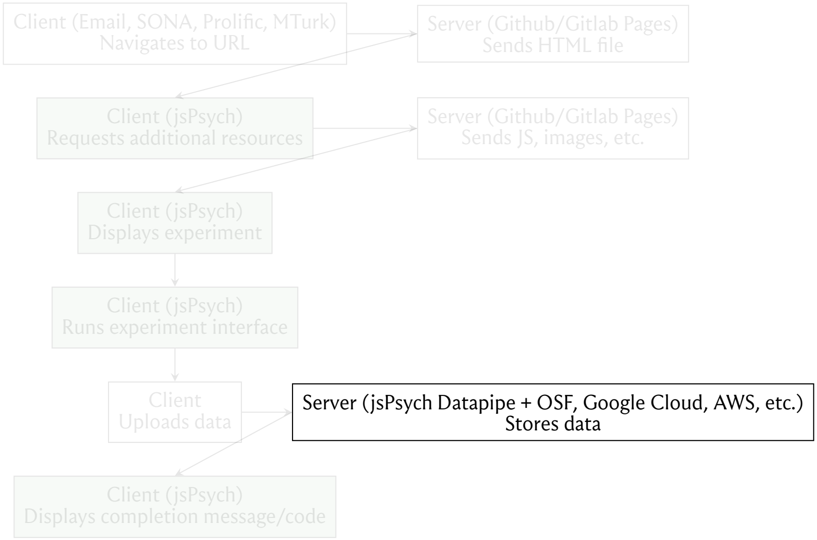
Getting the experiment online [for free]


Client (Email, SONA, Prolific, MTurk)
Navigates to URL
Server (Github/Gitlab Pages)
Sends HTML file
Client (jsPsych)
Requests additional resources
Server (Github/Gitlab Pages)
Sends JS, images, etc.
Client (jsPsych)
Displays experiment
Client (jsPsych)
Runs experiment interface
Client
Uploads data
Server
Stores data
Client (jsPsych)
Displays completion message/code

Getting the experiment online [for free]

- This includes other files, such as images, videos, audio, etc.
- Use either Github Pages or Gitlab Pages
  - `https://pages.github.com/`
  - `https://docs.gitlab.com/ee/user/project/pages/`
- Use a generic name for the project so that participants can't guess the purpose using the URL

Uploading data to a server for free



Client (Email, SONA, Prolific, MTurk)
Navigates to URL
Server (Github/Gitlab Pages)
Sends HTML file
Client (jsPsych)
Requests additional resources
Server (Github/Gitlab Pages)
Sends JS, images, etc.
Client (jsPsych)
Displays experiment
Client (jsPsych)
Runs experiment interface
Client
Uploads data
Server (jsPsych Datapipe + OSF, Google Cloud, AWS, etc.)
Stores data
Client (jsPsych)
Displays completion message/code

Uploading data to a server [for free]

- Use jsPsych DataPipe and an OSF.io project
  - `https://pipe.jspsych.org/getting-started`

1. Sign up for OSF.io
2. Sign up for jsPsych DataPipe
3. Make a private OSF.io project
4. Create an authorization token on OSF.io
5. Add authorization token to DataPipe
6. Create a DataPipe experiment and copy its ID
7. Use the `plugin-pipe` jsPsych plugin and provide the ID
   - `https://github.com/jspsych/jspsych-contrib/tree/main/packages/plugin-pipe`

Conditions for between subjects manipulations

- jsPsych DataPipe can provide a counter that you can access in your code
  - {0, 1, 2, 3, 0, 1, 2, 3, ...}
- Prolific can assign conditions to participants when they sign up
  - The information is passed to you using the URL, which you can access in code

Thank you!

- Questions?

Extra slides!

Quick tutorial to Javascript

- Very similar to Python or R
- Can be run in the browser
- Data types: Numbers, Booleans (true/false), Strings, Arrays (vectors), Objects
- Numbers: `3`
- Booleans: `true`, `false`
- Strings: `"hello world"`
- Array: `[38, 10, 1, 2]`
- Object: `{}`

Quick tutorial to Javascript: Printing stuff

```
console.log(3) // => 3
console.log("Hello world!") // => 3
```

Quick tutorial to Javascript: Declaring variables

```
let name = "Mary"
console.log(name) // => Mary
name = "John"
console.log(name) // => John

const age = 42
console.log(age) // => 42
age = 32 // ERROR!
```

- Use `let` for variables you want the change the value of
- For all other things, use `const`
- You can still change (mutate) arrays and objects when they are in a const variable

Quick tutorial to Javascript: Arrays

```
const numbers = [1, 2, 3]
const names = []

numbers.push(10)
names.push("Emika")
names.push("Aurora")

console.log(numbers) // => [1, 2, 3, 10]
console.log(names) // => ["Emika", "Aurora"]

console.log(numbers[0]) // => 1
console.log(names[1]) // => Aurora
console.log(names.length) // => 2
```

- Contains a list of items that can be accessed by position

Quick tutorial to Javascript: Objects

```
const person = {name: "Aurora", age: 42}
console.log(person) // => {name: "Aurora", age: 42}

person.hobby = "Bouldering"
console.log(person) // => {name: "Aurora", age: 42, hobby: "Bouldering"}

person["occupation"] = "Graduate student"
console.log(person) // => {name: "Aurora", age: 42, hobby: "Bouldering",
                    //     occupation: "Graduate Student"}

delete person["hobby"]
console.log(person) // => {name: "Aurora", age: 42,
                    //     occupation: "Graduate Student"}

console.log(person.occupation) // => Graduate student

const namesByParticipantId = {"1": "John", "2": "Aurora", "300": "May"}
console.log(namesByParticipantId[300]) // => May
console.log(namesByParticipantId["2"]) // => Aurora
```

- Contains key value pairs that can be indexed by key

Quick tutorial to Javascript: Conditionals

```
const age = 42
if (age < 32) {
  console.log("You not meet the experimental criteria")
} else {
  console.log("Please proceed to the next trial.")
}

// => Please proceed to the next trial.
```

Quick tutorial to Javascript: Iteration

```
for (let i = 0; i < 3; i++) {
  console.log(i)
}
// => 0
// => 1
// => 2

const names = ["Mary", "Emika"]
for (const name of names) {
  console.log(name)
}
// => "Mary"
// => "Emika"

const namesByParticipantId = {"1": "John", "2": "Aurora", "300": "May"}

for (const [key, value] of Object.entries(namesByParticipantId)) {
  console.log(value, key)
}
// => John 1
// => Aurora 2
// => May 300
```

Quick tutorial to Javascript: Functions

```
function greet(name) {
  console.log("Hello " + name + "!")
}

greet("Raj") // => Hello Raj!

const similarGreet = function (name) {
  console.log("Hello " + name + "!")
}

similarGreet("Mary") // => Hello Mary!

const anotherGreet = (name, greeting) => {
  console.log(greeting + " " + name + "!")
}

anotherGreet("Nick", "Salutations") // => Salutations Nick!
```

Quick tutorial to Javascript: Random numbers

```
console.log(Math.random()) // => 0.2342415

console.log(Math.round(Math.random() * 1000)) // => 382
```

References I

Chmielewski, Michael and Sarah C. Kucker (May 1, 2020). “An MTurk Crisis? Shifts in Data Quality and the Impact on Study Results.” In: *Social Psychological and Personality Science* 11.4, pp. 464–473. ISSN: 1948-5506. DOI: 10.1177/1948550619875149. URL: https://doi.org/10.1177/1948550619875149 (visited on 09/24/2024).

Douglas, Benjamin D., Patrick J. Ewell, and Markus Brauer (Mar. 14, 2023). “Data Quality in Online Human-Subjects Research: Comparisons between MTurk, Prolific, CloudResearch, Qualtrics, and SONA.” In: *PLOS ONE* 18.3, e0279720. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0279720. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0279720 (visited on 09/24/2024).

References II

- Henrich, Joseph, Steven J. Heine, and Ara Norenzayan (June 2010). "The Weirdest People in the World?" In: *Behavioral and Brain Sciences* 33.2-3, pp. 61–83. ISSN: 1469-1825, 0140-525X. DOI: 10.1017/S0140525X0999152X. URL: https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/weirdest-people-in-the-world/BF84F7517D56AFF7B7EB58411A554C17 (visited on 09/24/2024).
- *High Resolution Time API | Can I Use... Support Tables for HTML5, CSS3, Etc* (2024). URL: https://caniuse.com/high-resolution-time (visited on 09/24/2024).
- Syed, Moin (June 10, 2021). *WEIRD Times: Three Reasons to Stop Using a Silly Acronym*. Get Syeducated. URL: https://getsyeducated.substack.com/p/weird-times-three-reasons-to-stop-21-06-10 (visited on 09/24/2024).