

From an idea to a first experiment

Meeting 1 | Trials, timelines, files, and interpretable rows

Bartosz Maćkiewicz

Programming Behavioral Experiments in jsPsych

Build something you can explain

By the end, you can:

- run a local experiment and find its source;
- change a trial's stimulus, keys, and deadline;
- build a short sequence and inspect its saved rows.

Today's artifact: an eight-item lexical-decision pilot.

The route through today

1. Predict the participant's experience.
2. Connect HTML, JavaScript, and the browser.
3. Build a three-trial timeline together.
4. Extend a starter and inspect its data.

Four 45-minute blocks; take the scheduled break halfway.

Question: how do people classify familiar and unfamiliar strings?

- Participant sees a word or invented nonword.
- F = word; J = nonword.
- We save the item, category, key, and response time.

Our tiny English item set is a teaching example, not a validated test.

A fixation appears, then **garden**.

- What should the participant do?
- Which key should count as correct?
- What should be saved if they do nothing?

Write the answers before opening the code.

Garden is a word: F is the expected key. If no allowed key arrives before the deadline, response and rt are null.

Three files, three jobs

`index.html` loads; `app.js` defines the task; `stimuli.js` lists the items.

Change the file responsible for the behavior you want to change.

Open the right page

- Start the supplied local server from the course root.
- Open the starter from the launch page.
- Keep the folder structure intact.
- Save the file, refresh the browser, inspect the Console.

A local server serves files. It does not save response data.

A little JavaScript: arrays and objects

```
const keys = ["f", "j"];
const item = {
  text: "garden",
  lexicality: "word",
  correct_response: "f"
};
console.log(keys[0], item.text);
```

Predict the output. Arrays start at index 0.

Functions and strings

```
const text = "garden";  
const html = `

${text}</p>`;   
const makeHTML = (word) => `

${word}</p>`;


```

Backticks allow interpolation. A function runs when called. Quotes, commas, and braces have jobs: read the first error.

Plugin: kind of interaction. **Trial:** one configuration. **Timeline:** the sequence. **Row:** what a completed trial records.

Fixation and instructions produce rows too.

One configured trial

```
const welcome = {  
  type: jsPsychHtmlButtonResponse,  
  stimulus: "<h1>Welcome</h1>",  
  choices: ["Start"]  
};  
jsPsych.run([welcome]);
```

The core and plugin scripts must load before `app.js`.

Live checkpoint L1: trace the files

Open `materials/live-starter/` through the local server.

1. Locate core, plugins, helpers, and `app.js` in HTML.
2. Change the welcome message.
3. Save, refresh, and show the changed screen.

Success: no Console errors and a visible change.

A fixation has no response

```
const fixation = {  
  type: jsPsychHtmlKeyboardResponse,  
  stimulus: "+",  
  choices: "NO_KEYS",  
  trial_duration: 300  
};
```

Why would omitting the duration be a problem here?

With NO_KEYS and no duration, neither a key nor a timer can end the fixation.

A response trial has a deadline

```
const responseTrial = {  
  type: jsPsychHtmlKeyboardResponse,  
  stimulus: "<p>Press F or J.</p>",  
  choices: ["f", "j"],  
  trial_duration: 2000,  
  data: { task: "response_mapping_demo" }  
};
```

Time is in milliseconds. A timeout is a missing response.

Live checkpoint L2: connect the sequence

```
jsPsych.run([welcome, fixation, responseTrial]);
```

Test once with a valid key and once without responding. Try an invalid key. Describe which event ends the trial.

Read the result, not just the screen

```
jsPsych.data.get().values()
```

Find `trial_type`, `trial_index`, `response`, `rt`, and `task`.

A keyboard response is a string. A button response is an index. A timeout has `response: null` and `rt: null`.

Live checkpoint L3: make one deliberate change

Change the response text, deadline, and task label.

- Predict the effect of each change.
- Run and inspect the response row.
- Download the JSON using the supplied finish panel.

Refreshing before saving loses this in-memory dataset.

Core requirements:

- instructions: F = word, J = nonword;
- fixation before each item;
- eight task trials, each with a 3000 ms deadline;
- item/category labels and downloadable data.

No accuracy summary or timeline variables required today.

From one item to eight

```
const timeline = [instructions];  
for (const item of TEST_STIMULI) {  
    // Build a trial using this item's values.  
    timeline.push(fixation, trial);  
}
```

This is a construction sketch: define `trial` inside the loop. The complete starter shows where it belongs.

1. Run one item and verify its mapping and labels.
2. Add fixation and use all eight items.
3. Check one word, one nonword, and a timeout.
4. Download and count **task** rows.

Pair roles: driver edits; navigator predicts and checks.

What would make a better stimulus set?

Our examples differ in more than lexicality.

Consider length, familiarity, frequency, language, and how word-like the nonwords are. A working program does not remove these design decisions.

Debugging is part of programming

- Read the first Console error.
- Confirm the path and script order.
- Check spelling and case.
- Make one change, then rerun a small example.

Use the debugging guide before replacing the whole script.

Explain plugin, trial, and timeline using your own task. Show one response row and one timeout row.

Before Meeting 2: submit Assignment 1; review arrays, objects, functions, and the timeline handout.