

Programming Behavioral Experiments in jsPsych

Student handbook and assignment pack

Bartosz Maćkiewicz

Teaching edition 2026.10

Contents

Start here: your course guide	3
A compact programming bridge	5
Assessment and submission guide	7
Meeting 1 reference: A first experiment	8
Meeting 1: live coding	9
Meeting 1: workshop	10
Meeting 1: predict, run, explain	11
Meeting 2 reference: Repeated structure and trustworthy data	13
Meeting 2: live coding	14
Meeting 2: workshop	15
Meeting 2: predict, run, explain	16
Meeting 3 reference: Control flow and bounded practice	18
Meeting 3: live coding	19
Meeting 3: workshop	20
Meeting 3: predict, run, explain	21
Meeting 4 reference: Reliability and participant experience	23
Meeting 4: live coding	24
Meeting 4: workshop	25
Meeting 4: predict, run, explain	26
Meeting 5 reference: Pointer measurements and deployment	28
Meeting 5: live coding	29
Meeting 5: workshop	30
Meeting 5: predict, run, explain	31
Assignment 1: First task: from a question to saved rows	33
Assignment 2: Conditions and data: a randomized interference pilot	34

Assignment 3: Practice and control flow: safe routes through a task	35
Project proposal and pilot	36
Final project: a documented jsPsych prototype	37
Debugging and data checks	39
A small, synthetic data audit	40
Experiment ideas that fit this course	41
Meeting 3: a local JATOS study, export, and import	42
Running the course on Cognition.run or JATOS	44
Paper to prototype worksheet	46
Project title	47
Pilot log	48
Sources and assistance	49
Keep this beside your editor	50

Start here: your course guide

You will build small browser experiments and learn to check whether they measure what you intended. The five workshops move from one trial to a documented pilot with usable data. You do not need prior JavaScript, HTML, or CSS. Basic programming concepts help; the bridge below supports students who need a refresher.

Before Meeting 1 (30-60 minutes)

Meetings 1-2 run locally, so you can focus on files, trials, and data without setting up a platform account. In Meeting 3, we introduce local JATOS and study export/import using an already-working task. Cognition.run is an alternative; Meeting 5 revisits deployment with your project.

1. Download and unzip the **whole student course folder**. Keep its directory structure; copying only `app.js` loses dependencies.
2. Install or use an existing plain-text code editor. A desktop browser and keyboard are sufficient for the core course. Do not use a word processor to edit code.
3. Start a local server from the course folder. With Python installed: Windows `py -m http.server 8000 --bind 127.0.0.1`; macOS/Linux `python3 -m http.server 8000 --bind 127.0.0.1`. Keep the terminal open. If Python is unavailable, an editor's local preview/server feature works; ask for setup support rather than installing tools during the lesson.
4. Visit `http://127.0.0.1:8000/` and follow the Meeting 1 starter link. `127.0.0.1` means your own computer. Closing the server terminal stops the server.
5. Open developer tools (usually F12 or the browser menu), select Console, and run `2 + 2`. It should show 4.
6. Read the programming bridge. Change the welcome text in the Meeting 1 live starter, save, refresh, and check that the browser shows your edit.

The pinned libraries are included under `.shared/vendor/`; ordinary class activities work without an internet connection after download. Use the server even if double-clicking an HTML file appears to work. Your local web server does **not** store experiment results: download them at the end. No camera, paid hosting, npm, or build system is needed for student work.

Your working rhythm

Start each topic with the browser slides and its **Code lab**. Predict what will happen, edit a few lines, run, and inspect the saved rows. **Practice** gives a second change to try independently and an answer to check afterwards. These short exercises are ungraded. Download edits and data before changing examples or leaving a slide; open the lab in its own tab for longer work. The practice sheet links a complete source ZIP if you want to keep working in your editor.

Use **Read** for scrolling slides, the reference handout for explanations, and the PDF/handbook for printing. The experiment field guide connects design decisions, code, and data fields.

Keep a personal copy of each starter folder in its original location, or duplicate the entire course folder. Save a working version before a risky change. Read the matching exercise sheet, make one change, run, and inspect one data row. Work in pairs during class, switching the keyboard every 10-15 minutes. Both people should be able to explain the result. Reference solutions are for checking your reasoning after an attempt; do not submit them unchanged.

Meeting	Prepare	What you will leave with	Next submission
1	Setup and bridge	Eight-item lexical decision pilot, exported data	Assignment 1
2	Arrays/objects; timeline handout	Randomized condition task and guarded summaries	Assignment 2
3	Sketch practice/test flow; local JATOS preparation	Bounded practice and first study import/run/export	Assignment 3 + proposal
4	Bring a project prototype	Preloaded images and a documented failure test	Pilot
5	Bring pilot files and questions	Pointer data, save workflow, project review	Final project in 14 days

The assessment guide contains the weights and links to briefs. Required readings are short, specific sections in each meeting's reference handout; additional links are optional.

Read a trial before editing it

Identify its plugin (type), what appears (stimulus), allowed responses (choices), deadline (trial_duration), and the data labels. Ask: what would be saved if I responded correctly, incorrectly, or not at all? null means a missing response; it is not zero milliseconds. Instruction and fixation rows are not task observations.

The supplied Course helper creates the end-of-pilot download buttons and safe summaries. You may use it without rewriting it. The concepts assessed are your trial structure, labels, filters, and explanation. Its readable source is in `.shared/course.js`.

When something fails

Use the debugging guide. Copy the first error message, record which page you opened, and describe what you expected versus what happened. Avoid sending a screenshot of code when the actual text can be shared. Do not paste private participant data or credentials into help requests or AI tools.

Access and participation

Ask privately if you need an alternative input method, more time, an accessible stimulus set, or equipment. No diagnosis is needed in class. The Stroop activity has a shape/arrow alternative for students who cannot distinguish its colors. Advanced camera demonstrations are optional. The anonymous pre-course survey cannot identify who needs individual assistance; use the instructor's email for a personal response.

What a classroom pilot means

The examples use very few, deliberately simple items. English word familiarity, color discrimination, screen size, and input devices can affect performance. A working demo is evidence of a working implementation, not a replication or a validated psychological test. Your final documentation must distinguish these claims.

A compact programming bridge

Allow 30-45 minutes. Read code aloud, predict the result, then try it in the browser Console. These exercises are ungraded. If most concepts are new, complete the foundation route with the instructor.

Files and the browser

index.html loads libraries and your script. app.js contains experiment logic. stimuli.js contains item information. A path is relative to the HTML page for these examples; capital letters matter on many servers. Saving a file changes the disk; refreshing reruns it. Refreshing also clears in-memory experiment data.

HTML describes content: `<p>Press <strong>F</strong>.</p>`. CSS controls appearance: `.stimulus { font-size: 3rem; }`. JavaScript controls behavior. The string `"<p>Hello</p>"` becomes a paragraph only when a plugin inserts it into the page.

Values, arrays, and objects

```
const duration = 3000;           // Number, in milliseconds in our trials.
const word = "garden";           // String.
const keys = ["f", "j"];         // Array; first element is keys[0].
const item = { text: word, correct_response: "f" }; // Object.
console.log(item.text);          // garden
```

const prevents reassignment of the variable; it does not freeze an object's contents. Use let when a counter changes. JavaScript arrays start at 0 (like Python, unlike R's usual indexing). An object resembles a Python dictionary or a named R list, but property access commonly uses a dot.

Conditions and missing values

```
const response = null;           // No response, not the string "null".
const correct = response !== null && response === item.correct_response;
console.log(correct);            // false
```

Use === for strict equality. = assigns. && means AND; ! means NOT. JavaScript uses true, false, and null; these are not Python's True, False, and None, or R's TRUE, FALSE, and NA. For keyboard scoring, jsPsych's compareKeys also handles case consistently; guard a missing response first.

Functions run when called

```
function label(text) {
  return `

${text}</p>`;
}
console.log(label("garden"));
const double = (x) => x * 2;


```

Backticks allow `${...}` interpolation. Ordinary quotes do not. A function definition describes a computation; calling it performs it. A jsPsych dynamic parameter such as stimulus: `() => label(...)` runs later, when the trial needs its value. on_finish runs after a trial response has been recorded. This distinction matters in Meeting 2.

A loop builds several trials

```
const items = [{ text: "cat" }, { text: "mep" }];
const labels = [];
for (const item of items) {
  labels.push(label(item.text));
}
console.log(labels.length); // 2
```

Use for ... of to visit array values. for ... in visits property names and is not the pattern we want here. A JavaScript loop builds the timeline before the experiment starts; a jsPsych procedure schedules trials during execution.

Filtering and averaging (Meeting 2)

```
const rows = [{ rt: 500, correct: true }, { rt: null, correct: false }];
const usable = rows.filter((row) => row.correct && Number.isFinite(row.rt));
const mean = usable.length
  ? usable.reduce((sum, row) => sum + row.rt, 0) / usable.length
  : null;
```

filter keeps matching values. map transforms every value. reduce combines values. A missing mean stays null; report N/A instead of inventing a zero. You may use the supplied helper while learning these methods.

Check yourself

1. What is keys[1]? What does item.correct_response return?
2. Change the loop so it processes three items. How many labels should appear?
3. Predict whether "0" === 0 is true. Check it.
4. Explain why "\${word}" is different from a template literal using backticks.
5. A line says stimulus: greeting(); another says stimulus: () => greeting(). Which calls the function immediately?

Answers: 1. "j", "f". 2. Three. 3. False: types differ. 4. Double quotes leave the placeholder as text. 5. The first. If you cannot yet explain these, that tells us where to start; it does not prevent you from participating.

Further reference: [MDN JavaScript first steps](#).

Assessment and submission guide

All graded work is individual. In-class pair work and discussion are encouraged. Acknowledge borrowed code, collaboration, and substantial AI assistance. You should be able to explain and modify your submission. No marks are awarded for obtaining a significant effect.

Component	Course points	Default deadline	Brief
Coding assignment 1	10	Before Meeting 2	First task
Coding assignment 2	10	Before Meeting 3	Conditions and data
Coding assignment 3	10	Before Meeting 4	Practice and control flow
Project proposal	5	Before Meeting 4	Proposal submission
Project pilot	5	Before Meeting 5	Pilot submission
Final project	60	14 days after Meeting 5, 23:59 Europe/Warsaw	Final project
Total	100		

The instructor translates these into calendar dates and posts upload links before the course begins. The three exercises collectively retain the original 30% weight; proposal/pilot retain 10%; the project retains 60%.

Grade bands apply to the unrounded total: below 50 = 2 (fail); 50 to below 60 = 3; 60 to below 70 = 3+; 70 to below 80 = 4; 80 to below 90 = 4+; 90 to 100 = 5. Attendance, reassessment, and integrity rules are in the syllabus. Do not interpret these task sheets as replacing binding programme rules.

Every coding submission

Submit an archive with runnable source, all required assets/dependencies or a complete course-relative folder structure, a short README with launch instructions, and a small synthetic/self-pilot data export. Use a student identifier only in the learning platform's submission record or archive name, never in participant-facing experiment code. Do not submit .git, passwords, API keys, browser profiles, or other people's raw research data.

Include a 100-200 word note: what you changed from the starter, one test you performed, one limitation, and attribution/AI assistance. "No external assistance" is acceptable. Templates are in templates.

How marks are awarded

Each rubric names observable evidence and its maximum points. Award full credit for complete correct evidence, about half for a recognizably implemented but incomplete criterion, and zero for absent or nonfunctional evidence; use intermediate points with a specific reason. Assess the submitted version rather than a verbal promise of a fix. Give one actionable next step per assignment. A single bug should not be deducted repeatedly unless it independently undermines multiple criteria (for example, wrong scoring and no pilot evidence).

Students can use the published rubric as a self-check.

Meeting 1 reference: A first experiment

What you should be able to do

Eight lexical-decision task rows, with fixation and a saved export. Explain the logic and verify at least one normal and one edge case, using the actual saved data.

Concepts in context

The HTML page loads the core library, plugins, a small course helper, the item list, then app.js. Keep that dependency order. The local server solves file-serving issues; it does not become a result database. The bundled vendor files remove the ordinary CDN dependency.

A plugin defines an interaction, such as displaying HTML and collecting a key. A trial supplies its parameters. The timeline sequences those trials. The welcome, fixation, and response are all trials and each produces a row. Use task labels to distinguish their meanings.

The workshop uses a familiar JavaScript `for . . . of` loop to construct eight response trials from the item array. The loop runs before the experiment; jsPsych then presents the timeline. This avoids requiring timeline variables before they are taught. The loop must create a new trial using the current item's values and push fixation before it.

Keyboard choices are strings. A button response is a zero-based index. For scored tasks later, the correct response is a design field and the observed response is a participant field. In this meeting you inspect them manually. A missing response has null RT, never a fabricated 0 ms. The `Course.finish` helper adds a download/inspection panel after the timeline; it does not create an extra experimental row.

The lexical items are English teaching examples, not matched stimuli. Word length, familiarity, and nonword plausibility can confound a research comparison. Describe these limits rather than interpreting eight observations as evidence of a lexicality effect.

What to check in your implementation

17 total rows in the workshop: 1 instruction + 8 fixation + 8 response. The finish panel adds no row. Correctness is not scored automatically in Meeting 1. IDs and expected keys are recorded for later checks.

Use the live exercises and workshop sheet as the practical sequence. Compare with the released reference solution after an attempt. The debugging guide provides a symptom-to-check table. Try one editable example and its practice check before the full task.

Questions to answer without copying the code

1. Why does NO_KEYS fixation need a duration?
2. What happens if app.js loads before the plugin?
3. Does no response mean zero RT?

Check your reasoning by running a small example; the instructor guide contains the expected explanations.

Reading (about 15 minutes)

Read the [Hello World tutorial](#) and the single/multiple-trial sections of [the timeline guide](#). Focus on script order and trial objects; stop before advanced sampling.

Preparation for the next checkpoint

Explain a plugin/trial/timeline; identify one task row and one missing-response row. See assessment and deadlines.

Meeting 1: live coding

Work in materials/live-starter/ using the root launch page. The starter runs at its first checkpoint. Keep one browser tab and the Console open; save and refresh after each change. Your instructor will release the reference solution after an attempt.

L1. Trace loading and change the welcome (10 min)

Find core, plugins, helper, and app.js in index.html. Explain the order. Change the welcome sentence, save, refresh, and show the new text.

Success: The browser shows your edit and no Console errors.

Hint (after an attempt): Script tags execute in order; the app uses globals supplied by earlier scripts.

L2. Run a three-trial timeline (15 min)

Add fixation and responseTrial after welcome. Press a valid key, try an invalid key, then repeat without responding.

Success: Welcome → fixation → response. The response trial ends on F/J or after 2000 ms.

Hint (after an attempt): The trial objects already exist; edit the array passed to jsPsych.run.

L3. Metadata and timeout (10 min)

Change response text, deadline, and task label. Inspect the response row. Download JSON.

Success: Changed fields match the new trial; a timeout has null response and null RT.

Hint (after an attempt): The finish panel is supplied infrastructure. It is not a jsPsych trial.

Before moving on

Explain one changed line to a partner and inspect the relevant saved row. Record the first unresolved error rather than copying an entire replacement script.

Download this meeting's starter files, or use the matching folder in the whole student course download.

A small example before the full task

Use the predict-run-explain practice sheet and the editable examples in the browser slides. Each isolates one idea, shows the resulting data, and includes an independent transfer question.

Meeting 1: workshop

Deliverable: Eight lexical-decision task rows, with fixation and a saved export. Work from materials/workshop-starter/; attempt the core tasks before consulting the released reference solution. Pair work is encouraged; switch driver/navigator roles halfway.

W1. One item (15 min)

Run the starter with one item. Identify text, category, expected response, and observed response. Check F = word and J = nonword.

W2. Eight items and fixation (25 min)

Replace the one-item slice with all TEST_STIMULI. Insert fixation before each trial. Keep the loop; timeline variables belong to Meeting 2.

W3. Inspect and explain (20 min)

Run one valid word/nonword response and one timeout. Download CSV and JSON. Verify eight task rows and eight fixation rows, plus instructions. Explain one item limitation.

Acceptance checks

17 total rows in the workshop: 1 instruction + 8 fixation + 8 response. The finish panel adds no row. Correctness is not scored automatically in Meeting 1. IDs and expected keys are recorded for later checks.

Save the source and one clearly labeled self/synthetic pilot export. This in-class work is formative; graded homework has a separate brief and may require more items.

If you finish early

Ask students to add a second instruction screen or justify alternative language stimuli. Require an explanation of familiarity and length confounds.

Exit ticket

Explain a plugin/trial/timeline; identify one task row and one missing-response row.

Download this meeting's starter files, or use the matching folder in the whole student course download.

Meeting 1: predict, run, explain

These short activities connect the slides to the full workshop. They are **ungraded**. Use one as a five-minute class demonstration; use the other for a warm-up or independent practice. They replace part of the existing demonstration time.

Before every run, write one prediction. Afterward, record one row or screen that supports or contradicts it. Then make one change and explain the result without reading the code aloud.

1. One trial. Three possible outcomes.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: What will be saved for F, an ignored key, and no response?

Change and test: Change the deadline from 2000 to 4000 ms. Run once with F, once with X then J, and once without responding.

Inspect: Look at response and rt. An ignored key produces no extra row. A timeout has two null values.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

2. Build a timeline. Count its rows.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Two items, each preceded by a fixation: how many rows?

Change and test: Add “mirror” to the item list. Predict the new row count before running.

Inspect: The task column distinguishes fixation rows from decision rows. Your JavaScript loop builds the sequence before jsPsych runs it.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

Transfer to the workshop

In the lexical-decision starter, add a fixation before every item. Predict the total row count for eight decisions plus one instruction. Explain which eight rows answer the task question.

Diagnose one fault

A fixation uses NO_KEYS and has no duration.

Explain the cause and the smallest useful test. Do this before reading the check below.

Meeting 1: check your reasoning after an attempt

Example 1: F or J ends the trial. X is ignored, so the trial waits for F/J or the deadline. A timeout has `response = null` and `rt = null`; it is not a 2000 ms response.

Example 2: Two items produce four rows: fixation, decision, fixation, decision. Three items produce six. Only two or three rows, respectively, are word decisions.

Fault: There is no response route or timer to end it. Add a finite `trial_duration` and confirm that the next trial appears.

Keep a useful record

Download edited JS if you want to keep your change; a reset or refresh replaces it. For experiment examples, download JSON after the run. Source packs include HTML, the initializer, libraries, and assets: small-example source pack. These files are for learning; the workshop remains the core task.

Meeting 2 reference: Repeated structure and trustworthy data

What you should be able to do

Eight-item Stroop pilot with correct condition labels, scoring, and safe summaries. Explain the logic and verify at least one normal and one edge case, using the actual saved data.

Concepts in context

A repeated procedure has a timeline, a list of timeline_variables, and an order rule. One item supplies values for one pass through the procedure. `randomize_order: true` changes order without repairing the design. Sampling a subset and repeating the item set are separate operations; use them only when the design requires them.

In v8, `jsPsych.timelineVariable("name")` creates a placeholder for a direct parameter or data field. Inside your own function, call `jsPsych.evaluateTimelineVariable("name")` to get the current value immediately. Interpolating the placeholder into HTML can yield `[object Object]`. The item procedure must be active when you evaluate a variable.

Score in `on_finish`, after response data exists. Keyboard scoring guards null then calls `compareKeys`; button tasks compare numerical indices instead. The correct response comes from the item/design. Check it against the actual stimulus independently; a faulty answer key can make every downstream calculation look consistent but wrong.

Our accuracy denominator includes all task rows in the target block, including omissions. The RT mean includes correct responses with finite RT. Other research designs may specify different summaries, but must state them explicitly. A summary's stimulus function runs when the end screen is reached, so it uses completed data. The supplied helper returns null for an undefined estimate and formats it as N/A.

Do not average instruction/fixation times with task RT. Do not silently drop errors from accuracy's denominator. Use JSON to preserve types and parse CSV deliberately. A pilot contrast is descriptive; it does not justify population claims or selecting exclusions to obtain an expected effect.

What to check in your implementation

Workshop: eight Stroop rows, four per condition, each ink once in each condition. Synthetic key: 50% accuracy overall/by condition, two omissions, mean correct RT 800 ms overall, 600 congruent, 1000 incongruent. Assignment 2 deliberately expands to 16 balanced items.

Use the live exercises and workshop sheet as the practical sequence. Compare with the released reference solution after an attempt. The debugging guide provides a symptom-to-check table. Try one editable example and its practice check before the full task.

Questions to answer without copying the code

1. Why does `[object Object]` appear in a stimulus?
2. Does `randomize_order` guarantee equal conditions?
3. What is the RT mean with zero correct answers?

Check your reasoning by running a small example; the instructor guide contains the expected explanations.

Reading (about 15 minutes)

Read [timeline variables and random orders](#) and the filtering/aggregation portions of [data handling](#). Use the [v8 migration notes](#) when comparing older tutorials.

Preparation for the next checkpoint

Explain the two timeline-variable methods and the accuracy/RT denominators. See assessment and deadlines.

Meeting 2: live coding

Work in materials/live-starter/ using the root launch page. The starter runs at its first checkpoint. Keep one browser tab and the Console open; save and refresh after each change. Your instructor will release the reference solution after an attempt.

L1. Four arithmetic items (15 min)

Use all ARITHMETIC_ITEMS in the procedure and set randomize_order to true. Explain how the stimulus function gets the current item.

Success: Four distinct IDs and correct visible arithmetic.

Hint (after an attempt): Use evaluateTimelineVariable inside the stimulus function; direct data properties can use timelineVariable.

L2. Score responses (10 min)

Replace the null correctness placeholder with a guarded comparison of response and expected key. Test correct, wrong, and no response.

Success: Booleans are correct and the omission has correct=false.

Hint (after an attempt): Guard response===null before compareKeys.

L3. End summary (10 min)

Add the supplied summary to the timeline; trace its task/block and RT filters.

Success: Summary is computed after responses and N/A appears if no eligible RT exists.

Hint (after an attempt): The summary stimulus is a function. Course.summarize uses correct finite RTs.

Before moving on

Explain one changed line to a partner and inspect the relevant saved row. Record the first unresolved error rather than copying an entire replacement script.

Download this meeting's starter files, or use the matching folder in the whole student course download.

A small example before the full task

Use the predict-run-explain practice sheet and the editable examples in the browser slides. Each isolates one idea, shows the resulting data, and includes an independent transfer question.

Meeting 2: workshop

Deliverable: Eight-item Stroop pilot with correct condition labels, scoring, and safe summaries. Work from materials/workshop-starter/; attempt the core tasks before consulting the released reference solution. Pair work is encouraged; switch driver/navigator roles halfway.

W1. Stimulus and conditions (15 min)

Read ink_color dynamically rather than always showing black. Inspect congruent and incongruent items; confirm every target-color key.

W2. Randomize and score (20 min)

Use all eight items in random order. Complete correctness/omission logic. Verify four rows per condition and balanced ink colors.

W3. Summaries and audit (25 min)

Add the provided summary and explain every filter. Run an all-timeout pilot. Use the synthetic CSV to hand-check the calculation, then download your own pilot.

Acceptance checks

Workshop: eight Stroop rows, four per condition, each ink once in each condition. Synthetic key: 50% accuracy overall/by condition, two omissions, mean correct RT 800 ms overall, 600 congruent, 1000 incongruent. Assignment 2 deliberately expands to 16 balanced items.

Save the source and one clearly labeled self/synthetic pilot export. This in-class work is formative; graded homework has a separate brief and may require more items.

If you finish early

Add balanced items, distinguish sampling from repetition, or demonstrate a deterministic order for debugging. Do not introduce post-hoc RT exclusions.

Exit ticket

Explain the two timeline-variable methods and the accuracy/RT denominators.

Download this meeting's starter files, or use the matching folder in the whole student course download.

Meeting 2: predict, run, explain

These short activities connect the slides to the full workshop. They are **ungraded**. Use one as a five-minute class demonstration; use the other for a warm-up or independent practice. They replace part of the existing demonstration time.

Before every run, write one prediction. Afterward, record one row or screen that supports or contradicts it. Then make one change and explain the result without reading the code aloud.

1. One procedure, different items.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Will randomizing the order change the number of items?

Change and test: Switch `randomize_order` to false. Then add a third item with its own `item_id` and expected key.

Inspect: The displayed target, saved item ID, expected key, and observed response should agree.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

2. Which rows belong in the mean?

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Calculate accuracy and mean correct RT before running this code.

Change and test: Change every correct value to false. What should the mean become? Restore the example, then compute separate condition means.

Inspect: Incorrect responses and omissions stay in the accuracy denominator. Neither enters the correct-response RT mean.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

Transfer to the workshop

In the Stroop starter, explain how to add two balanced items without copying the response trial. Choose one row and reconstruct its expected response from its word, ink, and mapping. State both summary denominators.

Diagnose one fault

A summary averages `rt` over every row in the export.

Explain the cause and the smallest useful test. Do this before reading the check below.

Meeting 2: check your reasoning after an attempt

Example 1: Randomization changes order, not the supplied item counts. Each item supplies values to one pass of the procedure. Scoring runs after the response exists.

Example 2: Accuracy is $4/8 = 50\%$. Mean correct RT is $(500 + 700 + 900 + 1100)/4 = 800$ ms. Condition means are 600 and 1000 ms. With no correct rows, meanRT is null, not zero.

Fault: Instruction, fixation, wrong-response, and omission rows do not belong in the mean correct-response RT. Filter task/block, then correct finite RTs; report N/A for an empty eligible subset.

Keep a useful record

Download edited JS if you want to keep your change; a reset or refresh replaces it. For experiment examples, download JSON after the run. Source packs include HTML, the initializer, libraries, and assets: small-example source pack. These files are for learning; the workshop remains the core task.

Meeting 3 reference: Control flow and bounded practice

What you should be able to do

Flanker task that passes, repeats, or stops appropriately in both mapping groups. Explain the logic and verify at least one normal and one edge case, using the actual saved data.

Concepts in context

A block is a nested timeline with a distinct role. Practice and test may share a response-trial template but differ in feedback, item list, and eligibility. Explicit block labels prevent later analyses from relying on fragile row positions. Feedback rows have their own task label and must not count as practice responses.

The target direction and response key are separate concepts. The center character defines left/right; the assigned mapping defines F/J. The item list records direction and the trial computes the expected key. Show the active mapping. The teaching example uses B only when explicitly requested and A otherwise. The course default is A for local convenience; that is not a research allocation scheme.

Practice passes at 75% on the current four-item attempt and permits at most three attempts. The counter starts at 1 and increments in `loop_function` only when another attempt will run. A loop timeline's `on_timeline_start` is not a per-repetition callback. Each practice row stores that attempt. The practice summary computes the criterion from those rows. The loop repeats only when the criterion is unmet and attempts remain. Test is separately guarded by `conditional_function`; otherwise a stopped loop could send a failed participant into test.

The failed-practice branch shows a helpful message and still allows export for diagnosis. It creates zero test rows. Missing responses are incorrect; feedback explicitly names no response instead of pretending a wrong key was pressed.

The item list uses the literal arrow characters $\leftarrow$ and $\rightarrow$, so the stimulus can be a simple HTML template string. These are fixed course stimuli, not text entered by a participant.

A practice exclusion is a design choice. Record and report its rule and stopped cases. Counterbalancing requires an assignment procedure and an audit of actual groups, not merely a URL parameter. The small supplied flanker list is a coding demonstration; stimulus repetition and limited trials constrain interpretation.

What to check in your implementation

Pass-first: 4 practice + 8 test. Fail-then-pass: 8 + 8. Fail-all: 12 + 0. In $\leftarrow\leftrightarrow\leftarrow\leftarrow$ the correct direction is right; in $\rightarrow\leftrightarrow\rightarrow$ it is left. Group A left=F/right=J; B reverses these. Practice attempt IDs must be 1,2,3 only where applicable.

Use the live exercises and workshop sheet as the practical sequence. Compare with the released reference solution after an attempt. The debugging guide provides a symptom-to-check table. Try one editable example and its practice check before the full task.

Questions to answer without copying the code

1. Why is a loop cap not a test gate?
2. Why filter the current attempt?
3. What does alternating links accomplish?

Check your reasoning by running a small example; the instructor guide contains the expected explanations.

Reading (about 15 minutes)

Read the [looping and conditional timeline sections](#). Bring the project proposal worksheet before Meeting 4.

Preparation for the next checkpoint

Explain the two failure protections and identify an attempt-2 row. See assessment and deadlines. For Meeting 3, follow the local JATOS guide with the working Meeting 2 task. A `.jzip` transfers configuration and assets; export results separately. A localhost link works only on that computer. Cognition.run is the alternative.

Meeting 3: live coding

Work in materials/live-starter/ using the root launch page. The starter runs at its first checkpoint. Keep one browser tab and the Console open; save and refresh after each change. Your instructor will release the reference solution after an attempt.

L1. Practice feedback (10 min)

Add feedback after each practice response. Test a correct answer, a wrong answer, and a timeout.

Success: Three distinct messages; feedback never enters accuracy counts.

Hint (after an attempt): Read the last direction-task row, not simply the last arbitrary row.

L2. Mapping groups (10 min)

Run ?group=A and ?group=B. Predict both keys from the central arrow.

Success: Instructions, correct key, and group data agree.

Hint (after an attempt): The URL carries assignment; it does not balance your participant groups.

L3. Bounded repetition (10 min)

Replace the loop placeholder with failure AND attempts-below-maximum. Inspect the current attempt filter.

Success: At most three attempts; exactly 3/4 correct passes.

Hint (after an attempt): Start at attempt 1; increment the counter only when the loop decides to repeat. The summary updates practicePassed.

Before moving on

Explain one changed line to a partner and inspect the relevant saved row. Record the first unresolved error rather than copying an entire replacement script.

Download this meeting's starter files, or use the matching folder in the whole student course download.

A small example before the full task

Use the predict-run-explain practice sheet and the editable examples in the browser slides. Each isolates one idea, shows the resulting data, and includes an independent transfer question.

Meeting 3: workshop

Deliverable: Flanker task that passes, repeats, or stops appropriately in both mapping groups. Work from materials/workshop-starter/; attempt the core tasks before consulting the released reference solution. Pair work is encouraged; switch driver/navigator roles halfway.

W1. Check mapping and feedback (15 min)

Read the central character independently. Complete practice-only feedback; leave feedback out of test.

W2. Loop and test gate (15 min)

Complete bounded repetition and condition test on practicePassed. Preserve the helpful failure route.

W3. Branch audit (20 min)

Run pass-first, fail-then-pass, and fail-all. Check both groups and exactly 75%. Download labeled exports and compare task counts with the table.

Acceptance checks

Pass-first: 4 practice + 8 test. Fail-then-pass: 8 + 8. Fail-all: 12 + 0. In $\leftarrow\leftarrow\rightarrow\leftarrow\leftarrow$ the correct direction is right; in $\rightarrow\rightarrow\leftarrow\rightarrow\rightarrow$ it is left. Group A left=F/right=J; B reverses these. Practice attempt IDs must be 1,2,3 only where applicable.

Save the source and one clearly labeled self/synthetic pilot export. This in-class work is formative; graded homework has a separate brief and may require more items.

If you finish early

Add a voluntary repeat after a pass with explicit data recording, or design a balanced block-order assignment. Keep the safety bound.

Exit ticket

Explain the two failure protections and identify an attempt-2 row.

Separate platform checkpoint

Complete the 30-minute local JATOS activity using the working Meeting 2 task. Keep its study archive separate from its results export. This is formative platform practice, not additional defensive programming for Assignment 3.

Download this meeting's starter files, or use the matching folder in the whole student course download.

Meeting 3: predict, run, explain

These short activities connect the slides to the full workshop. They are **ungraded**. Use one as a five-minute class demonstration; use the other for a warm-up or independent practice. They replace part of the existing demonstration time.

Before every run, write one prediction. Afterward, record one row or screen that supports or contradicts it. Then make one change and explain the result without reading the code aloud.

1. Direction stays fixed. The key changes.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: For $\leftarrow\leftarrow\rightarrow\leftarrow\leftarrow$, which key is correct in each group?

Change and test: Change group from “A” to “B” and rerun. Then change one arrow string and its target direction together.

Inspect: Read the center arrow independently. The expected key follows the group mapping; the target direction does not change.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

2. Pass, repeat, or stop.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Will exactly three correct answers out of four pass?

Change and test: Try 2/4 then 3/4. Rerun and fail both attempts. This small demo permits two attempts; the workshop permits three.

Inspect: Filter by task and attempt. Passing creates one test row; failing at the maximum creates no test rows.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

Transfer to the workshop

Draw the full flanker routes with three allowed attempts and eight test trials. Mark where practice feedback belongs. Write expected practice/test counts for pass-first, fail-then-pass, and fail-all routes.

Diagnose one fault

Practice stops after its maximum attempts, then the next timeline node always runs test.

Explain the cause and the smallest useful test. Do this before reading the check below.

Meeting 3: check your reasoning after an attempt

Example 1: The center points right. Group A uses J; group B uses F. A URL parameter can carry this assignment but does not balance how many people receive each group.

Example 2: 3/4 passes because the condition is ≥ 0.75 . Failing once then passing creates eight practice rows and one test row. Failing twice creates eight practice rows and zero test rows.

Fault: The bound only stops repetition. Add a conditional test gate based on passing practice, and verify zero test rows in the fail-all export.

Keep a useful record

Download edited JS if you want to keep your change; a reset or refresh replaces it. For experiment examples, download JSON after the run. Source packs include HTML, the initializer, libraries, and assets: small-example source pack. These files are for learning; the workshop remains the core task.

Meeting 4 reference: Reliability and participant experience

What you should be able to do

Four-image shape pilot plus a documented normal and failure path. Explain the logic and verify at least one normal and one edge case, using the actual saved data.

Concepts in context

Participant instructions should specify the target, response mapping, speed/accuracy emphasis, deadline, practice, and how to stop. Ask another person to explain the task without coaching. A layout can look polished while still communicating the wrong decision.

Use semantic HTML and a restrained CSS style. Control image size and stimulus geometry deliberately; changing them between conditions may introduce a confound. Pixels are not visual angle and identical CSS does not make different displays physically identical. Keep keyboard focus visible and provide alternatives where color or input method is not essential to the question.

Preload an explicit list derived from the stimulus table when image paths are dynamic. The preload plugin should run before image trials, have a finite maximum loading time, and stop helpfully on failure. The example does not continue after an error. Test a broken path and restore it. Relative paths resolve from the page; source packaging must preserve or rewrite them.

The browser check tests a minimum viewport and excludes detected mobile devices. It does not prove that the participant has a keyboard, is attentive, or meets every inclusion criterion. Select thresholds for the actual task, explain exclusion, and test resize behavior. Do not copy a threshold into every project without a reason.

Preloading addresses asset readiness, not every source of timing variability. Hardware, refresh rates, browsers, operating systems, and input devices matter. A requested duration in code is not a direct measurement of physical presentation.

A pilot log records expected behavior before the test, actual observations, evidence, and fixes/retests. A failed test can be valuable evidence if honestly diagnosed. Open exported files and inspect their final rows; a download notification alone is not a complete data check.

What to check in your implementation

Four image-task rows in the workshop; circle/ellipse map to F, square/triangle to J. Two rows in the live demonstration (circle and triangle). Preload failure must not silently continue. All-timeout summaries have 0% accuracy and N/A correct RT.

Use the live exercises and workshop sheet as the practical sequence. Compare with the released reference solution after an attempt. The debugging guide provides a symptom-to-check table. Try one editable example and its practice check before the full task.

Questions to answer without copying the code

1. Does preload guarantee timing accuracy?
2. Does `mobile=false` prove a keyboard?
3. What makes a useful pilot log?

Check your reasoning by running a small example; the instructor guide contains the expected explanations.

Reading (about 15 minutes)

Read [media preloading](#), [browser-check](#), and [timing accuracy](#). Before Meeting 5, read the course deployment guide.

Preparation for the next checkpoint

Show one deliberately tested failure and the evidence that it was handled. See assessment and deadlines.

Meeting 4: live coding

Work in materials/live-starter/ using the root launch page. The starter runs at its first checkpoint. Keep one browser tab and the Console open; save and refresh after each change. Your instructor will release the reference solution after an attempt.

L1. Instruction card (10 min)

Add the card class and improve one instruction sentence. Have a partner explain the mapping unaided.

Success: Legible card, clear mapping, deadline, and stop route.

Hint (after an attempt): Use the existing class in .shared/course.css; small local CSS can go in the HTML style block.

L2. Preload and images (15 min)

Fill the explicit image list. Run both circle and triangle; then temporarily break an asset path and restore it.

Success: Correct image/key rows; failed loading stops with a helpful message.

Hint (after an attempt): Map SHAPE_STIMULI to item.file; dynamic trial paths are not a reliable automatic preload list.

L3. Browser assumptions (10 min)

Complete the mobile exclusion. Resize below 800x600 and inspect the resize/continue route.

Success: Small viewport is detected; supported desktop continues.

Hint (after an attempt): Document what is checked and what is merely assumed, such as a physical keyboard.

Before moving on

Explain one changed line to a partner and inspect the relevant saved row. Record the first unresolved error rather than copying an entire replacement script.

Download this meeting's starter files, or use the matching folder in the whole student course download.

A small example before the full task

Use the predict-run-explain practice sheet and the editable examples in the browser slides. Each isolates one idea, shows the resulting data, and includes an independent transfer question.

Meeting 4: workshop

Deliverable: Four-image shape pilot plus a documented normal and failure path. Work from materials/workshop-starter/; attempt the core tasks before consulting the released reference solution. Pair work is encouraged; switch driver/navigator roles halfway.

W1. Preload and instructions (15 min)

Use the complete four-image list and confirm that the response mapping is clear.

W2. Layout and procedure (20 min)

Use a controlled 220 px stimulus height and randomized order. Inspect rounded/angular scoring and the final summary.

W3. Pilot and project check (25 min)

Record a timeout, missing asset, small viewport, and reopened export. Restore changed paths. Give a partner one concrete project improvement.

Acceptance checks

Four image-task rows in the workshop; circle/ellipse map to F, square/triangle to J. Two rows in the live demonstration (circle and triangle). Preload failure must not silently continue. All-timeout summaries have 0% accuracy and N/A correct RT.

Save the source and one clearly labeled self/synthetic pilot export. This in-class work is formative; graded homework has a separate brief and may require more items.

If you finish early

Compare two reasonable viewport/layout choices or package the task and test extraction in a fresh directory. Explain any accessibility adaptation that changes the measure.

Exit ticket

Show one deliberately tested failure and the evidence that it was handled.

Download this meeting's starter files, or use the matching folder in the whole student course download.

Meeting 4: predict, run, explain

These short activities connect the slides to the full workshop. They are **ungraded**. Use one as a five-minute class demonstration; use the other for a warm-up or independent practice. They replace part of the existing demonstration time.

Before every run, write one prediction. Afterward, record one row or screen that supports or contradicts it. Then make one change and explain the result without reading the code aloud.

1. Load the image before the trial.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: If the image cannot load, should a response trial still run?

Change and test: Change circle.svg to missing.svg in the file variable. Run, inspect the stop message, then restore the correct path.

Inspect: Successful preload produces a row. On a missing file, the plugin stays on the error screen: no completed preload row and no image_decision row.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

2. What does 160 pixels specify?

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Does doubling stimulus_height guarantee twice the visual angle?

Change and test: Change the height from 160 to 240. Inspect the rendered image and the saved height. Then resize the preview window.

Inspect: The code controls a CSS-pixel dimension. Physical size also depends on the display; visual angle depends on viewing distance.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

Transfer to the workshop

Give a partner a fresh extracted shape task. Ask them to run it using only its instructions. Record one path or instruction problem, its fix, and evidence from the retest. Use the full task for the browser-size check.

Diagnose one fault

The task runs in its original folder but the image disappears from a copied folder.

Explain the cause and the smallest useful test. Do this before reading the check below.

Meeting 4: check your reasoning after an attempt

Example 1: With `continue_after_error` false, the failed preload holds on its error screen and does not finish a data row. No response to an absent image is recorded. Restore the path and retest the normal route.

Example 2: The requested height changes from 160 to 240 CSS pixels. This is not a guarantee about centimeters or degrees of visual angle. Layout constraints may also limit a rendered image.

Fault: The copied package may omit an asset or depend on its old relative path. Test a complete fresh extraction; compare the requested path with the actual filename and case.

Keep a useful record

Download edited JS if you want to keep your change; a reset or refresh replaces it. For experiment examples, download JSON after the run. Source packs include HTML, the initializer, libraries, and assets: small-example source pack. These files are for learning; the workshop remains the core task.

Meeting 5 reference: Pointer measurements and deployment

What you should be able to do

Four-decision pointer pilot, a verified local save, and a project completion plan. Explain the logic and verify at least one normal and one edge case, using the actual saved data.

Concepts in context

An extension is registered at jsPsych initialization, then enabled on the trials that need it. Tracking only the decision avoids confusing the return-to-start movement with the decision trajectory. The supplied center Start button standardizes the starting action before every item; it does not guarantee identical physical hand position or device behavior.

The mouse extension stores event coordinates relative to the viewport and event times relative to the tracked trial. Target rectangles make the coordinates interpretable. We explicitly request mousemove, mousedown, and mouseup; the default is only movement. A minimum sample interval limits event density but does not create a regular sampling clock. A stationary pointer can produce no movement samples, and the click that ends a trial may mean not every release event is captured.

Buttons record zero-based numeric indices. Compare response to `correct_choice`, not to a key string. Keep word, category, item, viewport, and target geometry with the trajectory. JSON is the primary teaching format for nested event arrays. Coordinates alone do not establish gaze, mental conflict, or a validated psychological measure.

The local completion panel downloads results on the current computer. For JATOS, the exported entry page loads `jatos.js`, starts inside `onLoad`, and calls `startNextComponent` with the result JSON in `on_finish`. This sends the final data and advances or finishes the study. Cognition handles saving through its platform workflow. Students do not implement custom retry or upload interfaces. Retrieve a complete pilot in the actual platform before distributing the task.

Cognition.run uses its own platform workflow. Do not assume JATOS-specific code applies there. Whichever runner is used, retrieve the saved pilot and check the final trial rather than relying on a finished screen.

WebGazer is optional methodological literacy: initialization, calibration, validation, and a justified task. Community extensions need checks of compatibility, permissions, units, and measurement quality. Neither is required for a strong final project.

What to check in your implementation

Four pointer-start rows and four decision rows in the workshop, plus instructions and summary (10 total). The live demo has one of each (4 total). Correct choice is 0 for cat/oak and 1 for hammer/lamp. Empty movement data is possible without motion; target metadata should still exist.

Use the live exercises and workshop sheet as the practical sequence. Compare with the released reference solution after an attempt. The debugging guide provides a symptom-to-check table. Try one editable example and its practice check before the full task.

Questions to answer without copying the code

1. Does `minimum_sample_time=16` guarantee 62.5 Hz?
2. What proves the researcher received results?
3. Does calibration validate webcam gaze?

Check your reasoning by running a small example; the instructor guide contains the expected explanations.

Reading (about 15 minutes)

Read [mouse tracking](#) and the course deployment guide. Optional: [eye-tracking workflow](#) and the two local extension worksheets.

Preparation for the next checkpoint

Explain local download versus submission versus confirmed stored result; state your next project fix. See assessment and deadlines.

Meeting 5: live coding

Work in materials/live-starter/ using the root launch page. The starter runs at its first checkpoint. Keep one browser tab and the Console open; save and refresh after each change. Your instructor will release the reference solution after an attempt.

L1. Enable tracking (15 min)

Register the extension globally and attach it to the decision trial with the supplied target IDs and explicit event types.

Success: The one-word live demo includes a nested trajectory and three target rectangles.

Hint (after an attempt): The `button_html` parameter is a function in v8. Copying a v7 array will fail.

L2. Inspect a trajectory (10 min)

Move the pointer before responding, finish, and inspect the decision row in JSON.

Success: Identify x/y/t/event and target bounds; distinguish start and decision rows.

Hint (after an attempt): 16 ms is a minimum interval, not a guaranteed sample frequency.

L3. Runner-aware completion (10 min)

Open the supplied platform copy. Identify Cognition's automatic saving or the two JATOS integration lines. Complete one pilot and retrieve its results; local JSON download is the fallback.

Success: The final summary row is included and the actual save destination is known.

Hint (after an attempt): JATOS starts inside `onLoad` and sends results with `startNextComponent` in `on_finish`. Cognition uses its own saving workflow.

Before moving on

Explain one changed line to a partner and inspect the relevant saved row. Record the first unresolved error rather than copying an entire replacement script.

Download this meeting's starter files, or use the matching folder in the whole student course download.

A small example before the full task

Use the predict-run-explain practice sheet and the editable examples in the browser slides. Each isolates one idea, shows the resulting data, and includes an independent transfer question.

Meeting 5: workshop

Deliverable: Four-decision pointer pilot, a verified local save, and a project completion plan. Work from materials/workshop-starter/; attempt the core tasks before consulting the released reference solution. Pair work is encouraged; switch driver/navigator roles halfway.

W1. Track four decisions (10 min)

Complete the same extension parameters in the workshop. Preserve the center Start action before every word.

W2. Check data and scoring (10 min)

Verify button indices 0/1, word/category, correctness, viewport, event records, and target rectangles.

W3. Save and document (10 min)

Complete the task on the selected platform and retrieve the result. If working locally, download/re-open JSON and label server saving as untested.

Acceptance checks

Four pointer-start rows and four decision rows in the workshop, plus instructions and summary (10 total). The live demo has one of each (4 total). Correct choice is 0 for cat/oak and 1 for hammer/lamp. Empty movement data is possible without motion; target metadata should still exist.

Save the source and one clearly labeled self/synthetic pilot export. This in-class work is formative; graded homework has a separate brief and may require more items.

If you finish early

Plot a saved trajectory relative to its target rectangles, or explain how changing geometry affects comparisons. Do not infer cognition directly from curve shape.

Exit ticket

Explain local download versus submission versus confirmed stored result; state your next project fix.

Download this meeting's starter files, or use the matching folder in the whole student course download.

Meeting 5: predict, run, explain

These short activities connect the slides to the full workshop. They are **ungraded**. Use one as a five-minute class demonstration; use the other for a warm-up or independent practice. They replace part of the existing demonstration time.

Before every run, write one prediction. Afterward, record one row or screen that supports or contradicts it. Then make one change and explain the result without reading the code aloud.

1. A button returns an index.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: What response value will clicking Living produce?

Change and test: Reverse the choices array. Update the correct response to match, then inspect a correct and an incorrect click.

Inspect: Button responses are numeric indices starting at zero. The label, index, and scoring rule must stay aligned.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

2. A click becomes a trajectory.

Open the editable example. The editor supplies startup and output code so you can focus on the experiment. Expand **Supplied setup and run lines** to see those lines; downloaded JS includes them.

Predict: Will a click without moving produce the same record as a curved movement?

Change and test: Move in a curve before clicking a response. Rerun with a nearly direct movement. Compare raw JSON and the trace.

Inspect: Use the decision row. Inspect x, y, t, event, and target rectangles. Coordinates belong to the experiment frame, not the whole slide.

Write three short notes:

- My prediction: ...
- Evidence from the run (field/value or screen): ...
- What I would change or test next: ...

Transfer to the workshop

For the semantic mouse task, identify the decision row and explain one target rectangle and one movement event. Then retrieve the exported result and check the final row. Distinguish this from merely reaching the last screen.

Diagnose one fault

The program compares a button response with the string "Living".

Explain the cause and the smallest useful test. Do this before reading the check below.

Meeting 5: check your reasoning after an attempt

Example 1: With ["Living", "Nonliving"], Living is index 0. After reversing the labels, Living is index 1. Comparing a button response with the string "Living" would fail.

Example 2: Both runs can end at the same button while containing different movement events. A 16 ms minimum interval is not a guaranteed sample rate. Pointer motion alone does not establish a cognitive conflict effect.

Fault: The button plugin records a numeric index. Compare with 0 or 1 for the active choices array, and check the mapping again after changing label order.

Keep a useful record

Download edited JS if you want to keep your change; a reset or refresh replaces it. For experiment examples, download JSON after the run. Source packs include HTML, the initializer, libraries, and assets: small-example source pack. These files are for learning; the workshop remains the core task.

Assignment 1: First task: from a question to saved rows

Weight: 10 course points. **Due:** Before Meeting 2. **Expected work:** 2-3 hours.

Task

Adapt the Meeting 1 lexical-decision starter into an eight-item task. Keep four words and four invented nonwords; use your own item set or explain one principled change to the supplied items. Present instructions, a fixation before each item, F/J responses, a 3000 ms deadline, and a data export. Use a JavaScript loop or explicit trials; timeline variables and summary statistics are not required yet.

Before coding, state in two sentences what the participant decides and which variable distinguishes your two item categories. This is a pilot, not a study of a lexicality effect: the tiny item set is not matched or validated.

Required checks

Run once normally and once leaving one response blank. Count eight task rows, not all timeline rows. Check one word and one nonword against the mapping. In this meeting, correctness need not be computed automatically; annotate the expected result in your README. Explain why eight items do not support a strong scientific conclusion.

Submit

Follow the common submission requirements. Include the runnable experiment, a synthetic/self-pilot export, and a short note with your checks. Keep a failed run when it demonstrates an intended branch; label it clearly. Do not collect research participants for this assignment.

Rubric

Criterion	Evidence	Points
Instructions and running sequence	Instructions, eight response trials, fixation before each, functional download	3
Response specification	F/J mapping and 3000 ms deadline match instructions; wrong keys ignored	2
Data labels	Task/block, unique stimulus IDs, category and expected response saved correctly	2
Verification	Export includes an intentional timeout and a hand-checked ordinary response	2
Explanation and attribution	Clear run instructions, changed items explained, help acknowledged	1
Total		10

Optional extension (no extra points required for full credit)

Add a second instruction page or adapt the task to an explicitly justified language. Explain how familiarity and item matching affect interpretation. Do not add randomization or accuracy code you cannot explain just to make the task look sophisticated.

Assignment 2: Conditions and data: a randomized interference pilot

Weight: 10 course points. **Due:** Before Meeting 3. **Expected work:** 3 hours.

Task

Extend the Meeting 2 Stroop starter to 16 trials: eight congruent and eight incongruent. Each ink color must occur twice in each condition. Give every item a unique ID, randomize the order without changing these counts, and store word, ink, condition, expected response, response, omission, and correctness. If a color task is unsuitable, use left/right central arrows with congruent/incongruent flankers and balanced target directions; document the substitution.

Show total trials, omissions, accuracy over all task trials, and mean RT for correct finite responses, overall and by condition. N/A must appear when an eligible subset is empty. Download CSV and JSON.

Required checks

Hand-calculate accuracy and mean correct RT for a four-row subset, then compare with your code. Run an all-timeout pilot to show that no NaN/Infinity or 0 ms mean is fabricated. Run twice and report the observed order; different order is plausible but not guaranteed by randomization. Counts must remain balanced regardless of order.

Submit

Follow the common submission requirements. Include the runnable experiment, a synthetic/self-pilot export, and a short note with your checks. Keep a failed run when it demonstrates an intended branch; label it clearly. Do not collect research participants for this assignment.

Rubric

Criterion	Evidence	Points
Design and item balance	16 items; eight per condition; target colors/directions balanced within condition	2
Reusable procedure	Correct v8 timeline variables, fixation, randomized order	2
Scoring and labels	Correct mappings, omission handling, independent answer-key check	2
Summaries	Explicit denominators, correct finite RT filter, empty-subset guard	2
Pilot and explanation	Exports, two observed orders, hand-calculated subset, attribution	2
Total		10

Optional extension (no extra points required for full credit)

Describe a possible congruency contrast without treating this pilot as evidence for an effect. Explain why full randomization does not guarantee alternation and why equal condition counts do not remove item confounds.

Assignment 3: Practice and control flow: safe routes through a task

Weight: 10 course points. **Due:** Before Meeting 4. **Expected work:** 3 hours.

Task

Complete the Meeting 3 flanker starter. Use the URL to choose A or B (default A), reverse the left/right key mapping across groups, and show the active mapping. Add practice-only feedback. Repeat the four-item practice until at least 75% are correct, up to three attempts. Compute the criterion from the current attempt. Run eight test trials only after a pass; otherwise show a helpful stop screen. Keep test feedback off. Save group, block, condition, item, attempt, expected response, correctness, and omission fields.

The provided stimulus lists are intentionally small. Independently check the middle character of every arrow string before trusting any answer key.

Required checks

Test A and B. Demonstrate: (a) pass first attempt: 4 practice + 8 test rows; (b) fail then pass: 8 practice + 8 test rows; (c) fail all three: 12 practice + 0 test rows. Also test exactly 3/4 correct to verify the boundary. Feedback and fixation rows do not enter these counts.

Submit

Follow the common submission requirements. Include the runnable experiment, a synthetic/self-pilot export, and a short note with your checks. Keep a failed run when it demonstrates an intended branch; label it clearly. Do not collect research participants for this assignment.

Rubric

Criterion	Evidence	Points
Mapping and instructions	Both A/B work; instructions and stored group match the active mapping	2
Phases and feedback	Practice has feedback, test does not; labels distinguish all rows	2
Bounded practice	Current attempt criterion, 75% threshold, maximum three attempts	3
Branch verification	Pass-first, fail-then-pass, fail-all evidence with expected trial counts	2
Documentation	Readable flow diagram/table, launch URLs, attribution	1
Total		10

Optional extension (no extra points required for full credit)

Offer a voluntary repeat after a successful practice pass, without removing the maximum-attempt rule or hiding that choice in the data. Explain why alternating two URL links is an allocation procedure, while merely reading a URL is not counterbalancing by itself.

Project proposal and pilot

These two checkpoints earn 5 points each and reduce the risk of building an oversized or uninterpretable project.

Proposal: before Meeting 4 (5 points)

Submit 300-500 words using the paper-to-prototype worksheet. A paper is optional. State a research question, a testable descriptive prediction, the manipulation/conditions, response, planned design, minimum item set, and data fields. Include a trial sequence sketch, a realistic classroom-sized scope, and one main limitation. If adapting a paper, identify the experiment/figure and clearly list what your version changes.

Scoring: question and manipulation 2; feasible implementation and timeline 2; source/limitations 1. Approval means “feasible as a course prototype,” not ethical approval or approval to recruit participants. Feedback should identify the smallest complete version to build next.

Pilot: before Meeting 5 (5 points)

Submit a runnable partial or complete experiment and one exported synthetic/self-test dataset, plus a pilot log. Exercise at least a normal response, a wrong response or invalid input, a missing response, and a loading/saving failure where relevant. Ask a classmate to read the instructions and voluntarily try a non-sensitive task; if that is impractical, use a second browser yourself and state that limitation. Do not record names or retain a classmate’s personal data.

Scoring: runnable path and usable export 2; documented tests including one edge case 2; prioritized fix list and next step 1. A discovered bug can strengthen the pilot report if you diagnose it honestly; an unexplained “everything worked” is weak evidence.

The final project should incorporate the feedback. If the server is unavailable, a local pilot remains valid for this checkpoint; record the deployment check still outstanding.

Final project: a documented jsPsych prototype

Weight: 60 course points. **Due:** 14 days after Meeting 5, 23:59 Europe/Warsaw. **Expected work:** about 23 hours across design, implementation, testing, and revision. Work individually; acknowledge assistance.

Goal and scope

Build a small behavioral task whose code, instructions, design, and data agree. Choose a feasible question you can explain. A carefully tested two-condition task is sufficient. Mouse tracking, eye tracking, custom plugins, and complex statistics are not required for a high mark.

Your task must have understandable instructions, at least two meaningful conditions or trial categories, repeated trials with a justified order rule, explicit response handling, identifiable task rows, and a working data export/save path. Include practice where task understanding requires it; a simple rating task may justify omitting criterion practice. Match the response and inclusion rules to the task rather than copying them mechanically.

Choose a small teaching item set (typically 12-24 test trials) and justify the counts. It must exercise the design, not claim sufficient power for a research conclusion. Code may extend a class starter if the change is meaningful and documented. Repeating a reference solution without an adaptation does not demonstrate independent design.

Deliverables

1. **Runnable source package:** HTML, JavaScript, dependencies, assets, and launch instructions. Include all required files; a hosted link alone is insufficient. Verify the archive on extraction into a fresh directory.
2. **Study README (800-1200 words):** question and prediction; conditions/design; trial flow; responses and deadlines; randomization/allocation; expected data and filters; source/asset credits; limits; how to run locally and on the selected runner. Use the template.
3. **Data dictionary:** define fields, types, allowed values, units, and missing-value behavior. Include task/block filters needed for summaries.
4. **Pilot evidence:** small synthetic/self-test CSV and JSON, a hand-check of at least two rows, a completed pilot log, and one screenshot of the participant view. Mark generated/test data as such. Do not collect a research sample.
5. **Deployment evidence:** a private classroom runner link where available, plus evidence that one complete pilot was retrieved from the runner and includes the final trial. If the instructor cannot provide a runner, submit a local reproducible package with the runner adaptation and an explicit untested-server checklist; this infrastructure fallback can earn full marks when agreed for the class.
6. **Change and assistance note:** changes since the pilot; known limitations; third-party code and asset sources; AI/collaboration disclosure.

For the course, demonstration tasks should use non-sensitive stimuli and voluntary self/classroom testing. Real recruitment or personal-data collection needs the relevant institutional process beyond this coding assignment. A public site is not required; keep classroom pilot links private.

Final project rubric

Criterion	Full-credit evidence	Points
Design and reasoning	Clear question, coherent manipulation and response, justified counts/order, realistic limitations	12
Implementation	All intended paths run; correct trial structure, mappings, conditions, and responses; suitable practice/branching	15
Data integrity	Complete labeled rows, correct scoring if relevant, missing-response handling, usable export and dictionary	12
Testing and reproducibility	Normal and edge cases, independent answer checks, fresh-extraction run, documented save verification	9
Participant experience	Readable instructions/layout, suitable assets/preload, clear completion/failure messages, accessible route where needed	6
Documentation and attribution	Clear README, sources, change log, assistance disclosure, deployment evidence or agreed fallback	6
Total		60

Partial credit follows the assessment guide. A complex feature earns credit only if it supports the design and works. A null or reversed descriptive result is not a grading failure. Do not remove incorrect/slow observations merely to make an expected effect appear.

Final self-check

Can another student run the archive without asking you where files are? Can they explain what one row means? Can you show which trials enter your summary? Can you demonstrate what happens after no response and after a save failure? Can you name one methodological claim your prototype cannot support? If yes, you have addressed the central goals of the course.

Debugging and data checks

Start with the first error

Open the Console, clear old messages, refresh once, and read the first error. Later errors often follow from it. Keep the Network panel available for missing scripts and images. Change one thing at a time and rerun the smallest relevant case.

Symptom	Likely cause	Check
initJsPsych is not defined	Core script missing or loaded too late	HTML script path and order
Plugin variable undefined	Plugin not loaded or wrong global name	Match script filename with documented plugin class
Page shows [object Object]	v8 placeholder inserted inside a function	Use <code>evaluateTimelineVariable</code> inside the function
Every item looks identical	Literal text or parameter computed too early	Dynamic stimulus function and current item
Unexpected token	Missing comma, quote, or brace	First indicated line plus the preceding line
Image missing	Wrong path or filename case	Open asset URL directly; inspect preload result
Keys do nothing	Wrong allowed keys, page lacks focus, deadline ended	Click page, inspect choices, try again
Correct answer marked wrong	Key mapping, type mismatch, or wrong answer key	Hand-score one saved row
Practice never ends	Cumulative data or unbounded loop	Current attempt filter and maximum attempts
NaN on summary	Empty eligible RT set	Return N/A if no eligible trials
Results disappear on refresh	Browser memory was not saved	Use download controls before refresh
JATOS page remains local	Wrong entry HTML or not launched from runner	Use generated <code>index-jatos.html</code> through JATOS

Four-minute data audit

1. Run a short pilot with one correct answer, one wrong answer, and one deliberate timeout.
2. Filter to the task and block. Count trials independently from the stimulus list.
3. For one row, reconstruct the expected response from the displayed stimulus and mapping. Do not use the saved `correct_response` field as the sole source of truth.
4. Check that the omission has `response: null`, `rt: null`, and `correct: false` where accuracy is scored.
5. Confirm the accuracy denominator includes all test trials. Confirm RT includes only correct, finite responses.
6. Download CSV and JSON. Open both; check the final trial is present. Nested pointer data is easier to inspect in JSON.

Bug report template

- Page/path and browser:
- Action I took:
- Expected behavior:
- Actual behavior:
- First Console error (copy text):
- Smallest example that reproduces it:
- One change I already tested:

A useful bug report describes an observation, not merely “it doesn’t work.” Keep identifying participant information out of examples.

A small, synthetic data audit

synthetic-pilot.csv was constructed for teaching. It is not human participant data and is not intended to demonstrate a real Stroop effect.

Filter to task = stroop and block = test. Empty CSV cells represent missing values here; parse numeric and Boolean columns deliberately rather than treating every cell as a string. The corresponding experiment JSON uses null, true, and false.

Before viewing the key below, calculate the number of test trials, omissions, accuracy, and mean RT for correct responses overall and by condition. Explain what happens if you include the instruction row or incorrect trials.

Answer key

Eight test trials; two omissions. Four correct responses divided by eight trials gives **50% accuracy**. Mean correct RT is $(500 + 700 + 900 + 1100) / 4 = 800$ ms. Each condition has 50% accuracy; congruent correct RT is 600 ms; incongruent correct RT is 1000 ms. The descriptive difference is 400 ms in these invented numbers, with no inferential significance.

If all four trials in a condition timed out, accuracy would be 0% and mean correct RT would be **N/A**, not 0 ms. If a condition had no trials at all, both accuracy and RT would be N/A. These are different situations.

The course has no automatic trimming threshold. A real study needs justified, documented exclusion rules and a suitable design/sample; do not select cutoffs after seeing which result they produce.

Experiment ideas that fit this course

Each option is a small teaching prototype, not a validated instrument or full replication. Start with a complete minimal version and inspect its data before adding features.

Family	Minimal task	Building blocks	Limitation to discuss
Choice reaction time	Choose a key for one of two shapes	HTML/image keyboard response	Device latency and anticipation
Stroop/flanker	Respond to ink/central arrow, ignoring distractors	Condition labels, practice, keyboard response	Item balance, mapping, color access
Lexical decision	Decide word/nonword	HTML keyboard response and item table	Language proficiency, frequency, length, nonword quality
Self-paced reading	Advance through sentence segments, then answer a question	Repeated keyboard trials and comprehension response	Segment construction and interpretation of segment RT
Recognition memory	Study words, then judge old/new items	Study/test blocks and separate lists	Lure selection, false alarms versus misses
Visual search	Find a target in a small static array	Prepared images or HTML/CSS layout	Set size and target position balance
Judgment/decision	Compare options or rate a statement	Button/slider response and item order	Wording and scale use
Learning from feedback	Learn two simple shape categories	Feedback and practice/test phases	Feedback changes behavior; enough observations needed
Ratings + task	Rate plausibility, then perform a small task	Button ratings or survey plugin	Question wording and carryover
Mouse trajectories	Choose living/nonliving from a common start point	Button response + mouse extension	Geometry, device, irregular event sampling

Ratings, recognition, and reading are feasible project options, not extra required workshops. A button-based rating scale works with the included plugins. Slider/survey plugins need an explicitly compatible package and a pilot. A reflective judgment need not inherit an RT task's deadline.

Keep webcam gaze, adaptive staircases, very brief masked priming, synchronized media, real-time multi-person interaction, physiological devices, and custom computer vision outside the beginner core. They require additional engineering or measurement validation. Simplify a paper's procedure rather than promising an equivalent implementation.

Read a paper's method, identify one trial sequence and critical manipulation, then complete the paper-to-prototype worksheet. A public citation is enough; the anonymous survey does not need uploaded PDFs.

Reference: [official plugin list](#).

Meeting 3: a local JATOS study, export, and import

Meetings 1-2 use ordinary local files. Now reuse the **working Meeting 2 Stroop task** to learn the platform without debugging a new experiment at the same time. Allow 20-30 minutes for preparation before class and 30 minutes for the guided activity. No Docker, API credentials, or server administration is required.

Before class: start JATOS locally

1. Download the instructor-tested JATOS release for your OS. This edition is prepared against **3.11.3**. Prefer the bundle with Java; choose the matching Mac architecture where applicable. [Official releases](#).
2. Extract it to a normal local folder, **outside Nextcloud, OneDrive, Dropbox, or another synchronized folder**. JATOS includes a running database; do not put that database in a sync directory.
3. Windows: open loader.bat. macOS/Linux: open a terminal in the extracted folder and run `./loader.sh start`; if needed, first run `chmod u+x loader.sh`.
4. Visit `http://localhost:9000/`. The fresh-install login is admin / admin; change the default password in your local installation. Keep this as a local development instance, not a public server.
5. Confirm that you see the dashboard, then stop there. On Windows, close the loader command window to stop it; on macOS/Linux use `./loader.sh stop`.

Follow [the official installation instructions](#) for OS permission prompts or Java issues. If setup fails, record the message and use a partner's or the instructor's demonstration; do not spend the coding lesson repairing an operating-system setup.

In class: import, run, inspect (0-15 minutes)

1. Start JATOS and check that the address bar says `localhost:9000`.
2. In the Studies sidebar, choose **+** → **Import study**. Select the working Stroop study directly. Do not unzip it first.
3. Open the imported study and use **Run**. Complete the eight-item task.
4. Open **Results**, inspect that run, and export/download its result data. Find eight Stroop rows, four per condition, and the final summary row. One complete task has 18 jsPsych rows: instructions + eight fixations + eight decisions + summary.
5. Locate the study's asset folder from the local dashboard. Open `app-jatos.js`: the experiment is familiar. The platform-specific changes are `jatos.onLoad` at startup and `jatos.startNextComponent(...)` in `on_finish`.

The GUI workflow is described in [JATOS Get started](#). Use normal Run rather than Preview for the data-saving check.

Export a study and move it (15-25 minutes)

1. Give your local study a recognizable classroom title; avoid modifying a study currently collecting research data.
2. Choose **Export** in the study toolbar and save the resulting `.jzip`.
3. Import that archive into a **different** local instance (a partner's is sufficient), or the instructor's classroom server using your assigned account. Check the destination URL before importing. If an import reports an existing study, do not overwrite someone else's work; use a fresh destination or the instructor's prepared copy.
4. Run the imported study and retrieve one new pilot result there. Confirm that the stimuli, mapping, and item counts match the original.

The archive transfers study configuration and assets, **not previous results**. The destination run creates new results. The participant link belongs to the destination instance; your localhost link will not work on someone else's computer. [Official transfer workflow](#).

Name the three different files (25-30 minutes)

File	What it is for	Does it contain participant results?
Course source .zip	Editable HTML/JS/CSS/assets; unzip to work on files	No
JATOS study .jzip	Import a configured study into JATOS	No
Results export	Inspect/analyze the recorded runs	Yes, for the runs exported

Do not rename a source ZIP to .jzip; the latter includes study metadata. In this course, the instructor's packaging script creates both formats, and students use JATOS's own Export/Import controls.

Exit check and fallback

Show where your study lives, where its results live, and which file would move the study elsewhere. If you used a partner/demo because installation failed, complete the same observations and export exercise together; the technical setup issue does not lower the coding-assignment mark.

Cognition.run is the alternative platform: open its prepared task in the browser, run it, and retrieve results. It does not require a local JATOS installation and does not use JATOS .jzip imports. Follow the platform guide. Meeting 5 revisits these steps with your project.

Running the course on Cognition.run or JATOS

Begin with ordinary local files in Meetings 1-2. In Meeting 3, use the local JATOS walkthrough to import, run, inspect, export, and transfer a working task. From then on, use **Cognition.run or JATOS** to run platform pilots. Students learn experiment code, not server administration. The local versions in this folder are a convenient preview/offline fallback.

What the platform does

The platform makes the experiment available at a link and stores its results. jsPsych handles the participant's trials and responses. Our platform integration should remain short; students do not need custom upload, retry, or backup interfaces.

Prepare a platform copy

The instructor can export any starter or solution:

```
python3 .tools/package-experiment.py 05_Mouse_tracking_and_deployment/solutions/workshop
```

The standalone folder/ZIP under `.build/packages/` contains the task, dependencies, assets, local entry page, a JATOS entry page, and `cognition.js`. The `.zip` contains source files; the separate `.jzip` includes JATOS study metadata and can be imported directly. Students can receive the prepared platform copy directly; they do not need to run this packaging script.

Cognition.run

1. Create a classroom task and open its source-code editor.
2. Use jsPsych 8 with the course's plugin versions. Configure the needed JS/CSS libraries through the platform's library facilities; do not load a second copy of jsPsych core.
3. Paste the exported `cognition.js`. It contains the supplied summary functions, item list, and experiment. Its initialization has no local-download or JATOS completion callback: Cognition handles result saving.
4. Apply the included `course.css`. For Meeting 4, upload the SVG assets and replace their paths in the item list with the platform's actual asset URLs.
5. Run the task via its participant link, complete it, and retrieve the result from the dashboard. Check the expected task rows and final trial before distributing the link.

The exact project version selector and asset URLs must be checked in the actual account. The course does not assume a version from the platform's older examples. Follow [Cognition's first-task guide](#) and [getting started](#).

JATOS

1. Import the generated `.jzip` using JATOS Import study. Alternatively, create a component and place the source ZIP's extracted files in its study assets.
2. Set the component HTML path to `index-jatos.html` in the exported folder. JATOS supplies `jatos.js`; do not copy a replacement into the package.
3. Launch the component through JATOS. The exported code differs from the local task in two small places:

```
jatos.onLoad(() => {
  const jsPsych = initJsPsych({
    on_finish: () => {
      jatos.startNextComponent(jsPsych.data.get().json());
    }
  });

  // Define the same trials and timeline here.
  jsPsych.run(timeline);
});
```

The completion call sends the data and moves to the next component, or finishes the study when this is the last component. There is no custom retry UI to explain. For mouse tracking, keep the extension registration in `initJsPsych` alongside `on_finish`.

For Meeting 3's mapping groups, the JATOS export reads `jatos.urlQueryParameters.group` inside `onLoad`. Create a **Personal Multiple** link under **Study Links** for classroom testing. Add

?group=B to that /publix/... link (or &group=B if it already has a query). The dashboard's Run URL does not forward these custom parameters. JATOS also redirects study links, so reading the final component's browser URL is insufficient. The local version keeps its ordinary URLSearchParams line. Verify the displayed mapping and the saved group. See [JATOS URL parameters](#).

4. Retrieve a complete result in JATOS. Check item IDs, conditions, responses, and the final trial. Test the instance's handling of a connection problem in a disposable pilot before class; that is instructor preparation, not an additional student programming assignment.

Reference: [official jsPsych/JATOS integration](#).

Local fallback

Serve the folder locally and open `index.html`. The supplied finish panel downloads CSV/JSON onto that computer. It does not upload to either platform. Download and reopen the files before refreshing. An absent server should not stop students learning trial structure and data checks.

What students must verify

Know which task link/version was run, where results are stored, and how to export them. Retrieve one pilot and inspect its counts and final row. A finished screen alone is not sufficient evidence. No API credentials belong in the task's browser code.

For research use beyond this classroom prototype, use the actual institutional participant-information, consent, data-access, and retention arrangements. The teaching examples are not approved research protocols. Preloading also does not establish physical timing accuracy; see [jsPsych timing guidance](#).

Paper to prototype worksheet

1. **Question:** What relationship or difference are you investigating?
2. **Source (optional):** Citation/DOI/public link. Which experiment, figure, or method section?
3. **Participant action:** What will they see/hear, and what will they do?
4. **Manipulation:** List the conditions and the exact property that changes. What stays constant?
5. **Outcome:** RT, accuracy, choice, rating, recall, or another measure? Define its unit and eligible trials.
6. **Trial flow:** Instructions → practice (if needed) → stimulus sequence → response → next trial → completion.
7. **Items and allocation:** How many per condition? Within or between participants? What is randomized, and what is counterbalanced?
8. **Minimal version:** What can you complete using already taught plugins? What can be omitted without losing the central contrast?
9. **Data row:** Write one hypothetical response row with condition, item, response, RT, and scoring if relevant.
10. **Failure case:** What should happen on a timeout, missing asset, failed practice, or save error?
11. **Feasibility:** Available stimuli and their permitted use; device/language/access assumptions; dependencies; runner.
12. **Limits:** What makes this an adaptation rather than a full replication? Which claim would the pilot not justify?

Instructor feedback: core scope / one necessary correction / optional extension / next checkpoint.

Project title

Run this version

Course/project version, required files, local-server command, entry HTML, supported browser/input assumptions. Explain any URL parameters and valid values. State whether a runner has been tested and how the marker can reproduce the local task without account credentials.

Question and design

Question, prediction, conditions, items, order/allocation rules, and rationale. Identify a source paper if relevant and list departures from its procedure. Distinguish the classroom item count from a research sample-size decision.

Participant experience

Trial sequence, instructions, response mapping, deadlines, practice criterion and maximum attempts, stop route, and completion/save behavior. Describe access assumptions and an appropriate alternative if needed.

Data and interpretation

Link the data dictionary. State the task/block filters, omission policy, accuracy denominator, and RT eligibility. Include one hand-checked row and one missing-response example. Explain nested fields and why you retain JSON.

Verification and deployment

Link the pilot log and clearly labeled test exports. Record actual browser/environment used, normal and failure paths, fresh-extraction check, and whether results were retrieved from a live server. Keep “implemented,” “tested with a mock,” and “verified on a server” distinct.

Limitations and next steps

Known technical issues, stimulus/design limitations, timing assumptions, and what further pilot/ethics work would precede research use.

Credits and assistance

Sources of code, stimuli, dependencies and licenses; collaborator contributions; substantial AI help and how you verified it. Link the assistance note.

Pilot log

Project/version:

Test environment/browser:

Data provenance: synthetic / self-pilot / voluntary classroom pilot with no identifiers

Write expected behavior **before** running each case. Keep failed results and record the retest after a fix.

Cases to cover

Normal complete run; wrong response; no response/timeout; alternative mapping/order; practice boundary/failure; missing image or invalid path; small viewport/device exclusion; download and reopen data; retrieve a complete platform result if used; run after fresh archive extraction.

Record one case (copy this block for each test)

Case and date:

Steps and input: What exactly did you do?

Expected behavior and rows: Write this before running the case.

Observed behavior: What happened? Name the exported file and relevant row(s).

Verdict: Pass / fail / N/A, with a reason.

Fix and retest: What changed, and which new evidence confirms the result?

Mark an irrelevant case "N/A" and explain why; do not claim it passed. Finish with the three most important remaining fixes and the evidence you will use to close them.

Sources and assistance

- Starting template and version:
- Borrowed code (source URL, location in project, what I changed):
- Stimulus sources and permitted use:
- Collaboration (what was discussed or jointly debugged):
- AI assistance (tool, tasks it helped with, code/text affected):
- My verification (checks I ran and what I independently changed/explained):

If none, say so. Do not include private conversations, passwords, or personal participant data. Disclosure is not a substitute for understanding the work.

Keep this beside your editor

Use this as a map when you know what you want to change but do not yet know where to look. You do not need to memorize the API.

From a question to a program

Write five sentences before adding features:

1. The participant sees
2. They respond by ..., before ...
3. The critical difference between conditions is
4. One task row must contain
5. I will know this works when ... happens in a deliberate test.

In the course folder, `index.html` loads libraries and scripts; `stimuli.js` or `items.js` holds the item list; `app.js` defines the task. In a small lab, setup is supplied around the editable snippet. Downloaded lab JS includes that setup, and the small-example ZIP includes HTML and libraries.

Find the right place to edit

I want to change...	Look here	Check afterward
Words, pictures, condition labels	Item list	Item IDs remain unique; counts still match the design
What appears in one trial	stimulus or prompt	The visible item agrees with its saved fields
Accepted response keys	choices	Instructions and expected keys agree
Maximum response time	<code>trial_duration</code> in milliseconds	A deliberate timeout gives null response/RT
Order of different phases	The array passed to <code>jsPsych.run</code>	Instructions, practice, and test occur in the intended order
Order of items within a phase	<code>randomize_order</code> on the procedure	Counts stay the same; only order changes
Data known before a response	data fields / item list	Saved labels identify item, task, block, and condition
Correctness after a response	<code>on_finish</code>	Correct, wrong, and omitted responses score as intended
Whether to repeat practice	<code>loop_function</code>	Current-attempt criterion and attempt maximum both apply
Whether test is allowed	<code>conditional_function</code>	Failed practice produces no test rows

Four distinctions that prevent many bugs

Building versus running. An ordinary JavaScript loop can build a timeline before any trial starts. A jsPsych timeline loop repeats a procedure while the experiment runs. Choose the tool that matches the required behavior.

Placeholder versus current value (Meeting 2). Use `jsPsych.timelineVariable("text")` directly as a parameter. Inside a JavaScript function or a template string built by that function, use `jsPsych.evaluateTimelineVariable("text")` to obtain the current value.

Expected versus observed response. An expected key is part of the design. response is what actually happened. Validate an answer key against the stimulus and mapping, independently of the key itself.

Completion versus saving. A final screen does not prove that a result was stored. For local work, download and reopen a file. For JATOS or Cognition.run, retrieve the completed result from that runner.

Read one row before summarizing many

This abbreviated row is **illustrative**, not a participant result:

```
{
  "task": "stroop",
  "block": "test",
  "item_id": "i1",
  "condition": "incongruent",
  "word": "BLUE",
  "ink_color": "red",
  "correct_response": "f",
  "response": "j",
  "rt": 640,
  "correct": false,
  "omission": false
}
```

Under F = red, G = blue, H = green, J = black, the expected response is F. This is an incorrect response, not an omission. It contributes one incorrect trial to accuracy and does not enter the mean correct-response RT. If it timed out instead, response and RT would both be null, and omission would be true.

Saved value	What it means
Keyboard response: "f"	An allowed key was pressed
Button response: 0	The first choice was clicked, regardless of its label
response: null, rt: null	No allowed response before the trial ended
A field is absent	That trial did not record that field; inspect its task/plugin
An empty movement array	No events of the tracked kind were recorded; check the task and interaction
correct_response	The programmed key/index, which still needs an independent check

Filter, then calculate

First choose the task and block. For course accuracy, use correct task responses divided by **all** task trials, including omissions. For mean correct RT, use only correct responses with finite RT. An empty eligible RT subset has no mean: store null and display N/A. Do not replace missing RT with zero.

The worked data audit and Meeting 2 practice provide a small calculation you can verify by hand.

A five-case pilot

Run a normal response, an incorrect response, an omission, a relevant loading/control-flow failure, and a complete save/retrieval. Record the expected outcome, observation, evidence, fix, and retest. Use the pilot log. A bug found and explained is more useful than an undocumented "everything worked."

For a specific symptom, use the debugging checklist. For the complete assessed work, follow the assignment briefs.