Starting out

- make sure to tell them that this is not online and the differences
- Take them to the website and show them where to find info, download and version control

Setting up for "Hello world!"

- Download zip file from repo
- Save it somewhere good like your desktop

Looking at the content of the file

- Double click on the FirstExperiment.html file
- This will open the file in your default browser
- People should now see "Hello world!" written in the browser
- Try pressing any button on the keyboard (the words disappear)
- So how is it doing this? Let's have a closer look:

```
 1 <!DOCTYPE html>
 2 <html>
 3   <head>
 4     <title>My experiment</title>
 5     <script src="jspsych/dist/jspsych.js"></script>
 6     <script src="jspsych/dist/plugin-html-keyboard-response.js"></script>
 7     <link href="jspsych/dist/jspsych.css" rel="stylesheet" type="text/css" />
 8   </head>
 9   <body></body>
10   <script>
11     const jsPsych = initJsPsych();
12 
13     const hello_trial = {
14       type: jsPsychHtmlKeyboardResponse,
15       stimulus: 'Hello world!',
16       choices: ['a']
17     }
18 
19     jsPsych.run([hello_trial]);
20   </script>
21 </html>
```

- This is what is in the html file
- First we have some set-up code needed to run the jsPsych library javascript code
- We can play with some of this, in line 4, change the text from "My experiment" to "First experiment".
- Save the file and open it again/reload it in your browser.
- If the file ever doesn't change, try a hard reload!
- This changes what appears in the tab for that file.
- Then we have lines that read in the source file, that reads in the specific plugins and one that reads in the CSS file.
- All these are in the folder we downloaded!
- Look at jsPsych file, show them where the plugins are
- Why is this an html file if we are working in a JavaScript library?
- jsPsych code all wrapped up in an html sandwich!
- Script tags to interpret content as JavaScript
- It is between these two tags that we write our main experiment code!

Playing with "Hello world!"

- We will now focus on what is between the script tags.

```
const jsPsych = initJsPsych();

const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: 'Hello world!',
  choices: ['a']
}

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

- Two lines of code that must be in all jsPsych experiments (in this version)
- In one line we initialize jsPsych, essentially telling the document. This needs to be initialised for the rest of the document content to be treated as jsPsych code.
- Explain about colour highlighting (help with editors).
- Run line is what actually starts the experiment, it contains something called a timeline, which is the series of trials that make up your experiment.
- "name" of trials in the timeline
- We use something called a declaration, here the keyword "const" is used to declare a variable called "hello_trial" (note the underscore).
- Different ways to declare/define variables depending on how they will be used.
- Contained within the curly brackets we then have information about this specific trial.
- Almost all trials need to contain a parameter called "type".
- Can anyone tell me what is defined in the parameter?
- It is the kind of trial we have, so in this case this is the piece of code which points at a specific plugin, in this case HtmlKeyboardResponse.
- Then we have the "stimulus" parameter, which in this case just contains some text (or a string, in programming language).
- The stimulus is what is displayed on the document when you open it.
- So why did the screen just turn blank after you pressed 'a' button?
- Let's go to the website!
- Find the plugin, find the choices parameter and explain what we have defined!
- Ask them to comment out choices

```
const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: 'Hello world!'//,
  //choices: ['a']
}
```

- Now any button can be pressed!
- Now let's make it an option to press several buttons.
- Explain that this is an array, with two elements.

```
const hello_trial = {
  type: jsPsychHtmlKeyboardResponse,
  stimulus: 'Hello world!',
  choices: ['a', 'b']
}
```

- Now we can press either a or b.
- Remove last bracket in array and reload
- Oh no! we broke it and nothing is happening.
- Open your browser's web developer tools. Usually under "tools" or "view". And find the console.
- Here there is an error, and it is very informative! It tells us exactly what is wrong and where in the code we need to make an edit to fix it.

```
 const hello_trial = {
   type: jsPsychHtmlKeyboardResponse,
   stimulus: 'Hello world!',
   choices: ['a', 'b']
 };
```

- Much better!
- But how will anyone know that they must press "A" or "B"? We can write that as the text in the stimulus parameter, instead of hello world.

```
 const hello_trial = {
   type: jsPsychHtmlKeyboardResponse,
   stimulus: 'Press "A" or "B" to continue',
   choices: ['a', 'b']
 };
```

Building a Bouba/Kiki experiment

- Does anyone here know about the bouba/kiki effect?
- Sound symbolism – certain sounds associated with certain shapes.
- Let's build a simple version of an experiment which tests this.
- We will need:
  - A consent/instruction trial
  - Some test trials
  - Perhaps some demographics questions/debriefing trial
- Let's start with the consent trial, this involves a minimal change from the current Hello World trial.
- First let's change the text.
- We will change this to be an html button response trial instead so people can indicate consent via a button press.

```
 const hello_trial = {
   type: jsPsychHtmlKeyboardResponse,
   stimulus: 'Welcome to the experiment! <br> Here are some instructions. \
   If you agree to take part in the experiment, press the button below.',
   choices: ['a', 'b']
 }
```

- Nice! We should change it so that there is a button response instead.
- Start by adding the html-button-response plugin to the experiment

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="jspsych/dist/jspsych.js"></script>
    <script src="jspsych/dist/plugin-html-keyboard-response.js"></script>
    <script src="jspsych/dist/plugin-html-button-response.js"></script>
    <link href="jspsych/dist/jspsych.css" rel="stylesheet" type="text/css" />
  </head>
```

- Then change the plugin name
- Now if you run the experiment there are two buttons.
- It counts each element in an array as its own button!

```
    const hello_trial = {
      type: jsPsychHtmlButtonResponse,
      stimulus: 'Welcome to the experiment! <br> Here are some instructions. \
      If you agree to take part in the experiment, press the button below.',
      choices: ['a', 'b']
    }
```

- We can change the button text.

```
    const hello_trial = {
      type: jsPsychHtmlButtonResponse,
      stimulus: 'Welcome to the experiment! <br> Here are some instructions. \
      If you agree to take part in the experiment, press the button below.',
      choices: ['I agree']
    }
```

- This does mean that the name of the trial is a bit odd, it is no longer a hello trial. Let's change that!
- We also need to change the name of the trial that we have in the timeline now

```
    const welcomeTrial = {
      type: jsPsychHtmlButtonResponse,
      stimulus: 'Welcome to the experiment! <br> Here are some instructions. \
      If you agree to take part in the experiment, press the button below.',
      choices: ['I agree']
    }

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

- Now let's work on the test trial!
- Look at image folder
- Make sure they are called "spikey.png" and "rounded.png"
- Can you figure out what plugin to use?
- Then we need to load the appropriate plugin

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="jspsych/dist/jspsych.js"></script>
    <script src="jspsych/dist/plugin-html-keyboard-response.js"></script>
    <script src="jspsych/dist/plugin-html-button-response.js"></script>
    <script src="jspsych/dist/plugin-image-button-response.js"></script>
    <link href="jspsych/dist/jspsych.css" rel="stylesheet" type="text/css" />
  </head>
```

- To start the new trial we can write a comment to explain what this new trial is
- And give the trial a name “testTrial”
- We set the “type” parameter to “jsPsychImageButtonResponse”
- And the stimulus parameter will now include a path to the image we want to display
- Here the folder name and file name ‘images/spikey.png’
- And then we set a prompt which tells participants what to do.

```
 const testTrial = { //defines first test trial
   type: jsPsychImageButtonResponse,
   stimulus: 'images/spikey.png',
   prompt: '<p> Which of these two names best fit the shape above?</p>',
   choices: ['I agree']
 }
```

- Try to run, no trial appears! Why?
- Need to add to timeline.

```
 jsPsych.run([welcomeTrial, testTrial]);
```

- Now the trial appears, but the image is really big!
- Try to figure out how you can change the size of the image so that it is smaller.

```
 const testTrial = { //defines first test trial
   type: jsPsychImageButtonResponse,
   stimulus: 'images/spikey.png',
   stimulus_height: 306,
   prompt: '<p> Which of these two names best fit the shape above?</p>',
   choices: ['I agree']
 }
```

- Finally we also want to change the trial so it contains two buttons, one with ‘kiki’ and one with ‘bouba’.
- Anyone who can tell us how to do this?

```
const testTrial = { //defines first test trial
  type: jsPsychImageButtonResponse,
  stimulus: 'images/spikey.png',
  stimulus_height: 306,
  prompt: '<p> Which of these two names best fit the shape above?</p>',
  choices: ['bouba', 'kiki']
}
```

- Currently we are only displaying one of the images! It would be great to have one trial for each shape.
- How could this be achieved? Work with your partner

Different options! Write the code out twice? This is costly!

- One suggestion is to use a nested timeline.
- Let's look at that on the website!
- This allows you to set different values for the parameters in a trial and that trial will run for the number of different values you have in the nested timeline.
- Let's create a new trial to exemplify this behaviour.
- Start by copying the testTrial code and pasting it right below the first testTrial.
- Then change the name to "complexTestTrial"
- Delete the value of the stimulus parameter
- Instead add a new parameter called "timeline" under the "choices" parameter
- Write the following code

```
const testTrial = { //defines first test trial
  type: jsPsychImageButtonResponse,
  stimulus_height: 306,
  prompt: '<p> Which of these two names best fit the shape above?</p>',
  choices: ['bouba', 'kiki'],
  timeline: [ //array of objects
    {stimulus: 'images/spikey.png'},
    {stimulus: 'images/rounded.png'}
  ]
}
```

- Don't forget to add the new trial to the timeline!

```
jsPsych.run([welcomeTrial, complexTestTrial]);
```

- Now the experiment runs this trial twice, once with each image!
- But they will always appear in the same order… that's not great!
- To fix this we need to randomize the order that the images appear in the timeline!
- Different ways to randomize order of trials, depending on the required behaviour.
- Let's look at the website!
- To do this we start by creating a new variable called "images" right above the complexTestTrial
- In this variable we open square brackets to create an array, and list both images.
- We now have an array containing these two images, and to randomize them we add a jspsych randomization function.
- jsPsych.randomization.shuffle will work

```
const images = jsPsych.randomization.shuffle(['images/spikey.png',
'images/rounded.png'])
```

- Now we change the content of the stimulus parameter in the nested time line to index into this array
- Explain how indexing works!
- So it should look like this.
- Now the experiment will randomize which image comes in the first trial and which comes in the second trial

```
    timeline: [
      {stimulus: images[0]},
      {stimulus: images[1]}
    ],
```

- But the buttons are still appearing in the same order every time… this is also an issue!
- The randomization behaviour we want here is perhaps a bit different, we might want to randomize the order at each trial, rather than once for the whole experiment.
- The choice will depend on your experiment.
- We can do this by also making “choices” a part of the nested timeline.

```
const complexTestTrial = { //defines complex test trial with both images
  type: jsPsychImageButtonResponse,
  stimulus_height: 306,
  prompt: '<p> Which of these two names best fit the shape above?</p>',
  timeline: [ //array of objects
    {stimulus: images[0], choices: jsPsych.randomization.shuffle(['bouba',
    'kiki'])},
    {stimulus: images[1], choices: jsPsych.randomization.shuffle(['bouba',
    'kiki'])}
  ]
}
```

- Let’s implement a gap between these trials, the fact that they just flash from one to another is a bit odd.
- To do this we can add a setting to the initJsPsych function
- Let’s look at the different things you can set in this function
- Can you find the one we need?
- Default_iti (inter-trial interval)
- Let’s add this to the initJsPsych function

```
const jsPsych = initJsPsych({
  default_iti: 500
});
```

- That looks neat! Let’s add a debrief/demographics question trial to finish up
- We want one question where people can type in their date of birth, and one where they can tell us their native language(s)
- Find a plugin!

- There is one called survey-text, that should work!
- Let's add this to the opening lines

```
<!DOCTYPE html>
<html>
  <head>
    <title>My experiment</title>
    <script src="jspsych/dist/jspsych.js"></script>
    <script src="jspsych/dist/plugin-html-keyboard-response.js"></script>
    <script src="jspsych/dist/plugin-html-button-response.js"></script>
    <script src="jspsych/dist/plugin-image-button-response.js"></script>
    <script src="jspsych/dist/plugin-survey-text.js"></script>
    <link href="jspsych/dist/jspsych.css" rel="stylesheet" type="text/css" />
  </head>
```

- Then we define a new trial called "surveyTrial"
- Add the new plugin as the "type"
- And inside the "question" parameter we can add several features like prompts and place holders like, this acts in a similar way to the nested timeline except here they are all displayed on the same page.
- We can also make some questions required and others not.

```
const surveyTrial = {
  type: jsPsychSurveyText,
  questions: [
    {prompt: 'What is your date of birth?', placeholder: 'dd/mm/yyyy',
    required: true},
    {prompt: 'What is your first language(s)?', placeholder: 'e.g. Swahili'}
  ]
}
```

- Finally, add this trial to the main timeline so it runs

```
jsPsych.run([welcomeTrial, complexTestTrial, surveyTrial]);
```

- Nice! So we have a little bouba/kiki experiment but what about the data we collect from this?
- Let's look at the data section on the website
- As a first step let's display the data in csv formal

```
const jsPsych = initJsPsych({
  default_iti: 500,
  on_finish: function() {
    jsPsych.data.displayData('csv');
  }
});
```

- Here we see the top row showing the names of the columns, and subsequent rows has data from one trial each.
- A crucial bit of data is missing, can you spot what it is?
- Since we randomize the buttons we need to know what the text was on each individua button.

- We can do that by using a parameter called save_trial_parameters.
- If you look this up on the website, can you figure out how to add this to your complexTextTrial to make sure the button text is also visible in the data??

```
          'kiki'])}
        ],
        save_trial_parameters: {
          choices: true
        }
      }
```

- Now there is a new column with this data!
- What about saving the data?
- Since we are running this experiment locally we can save this csv file directly on our computer.
- If this was run online and via a server, then the data saving process is more complex and not something we will go into in detail today.
- I will show you a little bit about what's involved to do that but first let's just save this locally.
- Replace the previous data call to this

```
        on_finish: function() {
          jsPsych.data.get().localSave('csv', 'someData.csv');
        }
      });
```

- This file should end up in your download folder
- Open this file and have a look at the content
- This saves data at the end, there is also a way to do this trial by trial (especially good if running online).

Additional tasks:
- make an audio version of the bouba kiki experiment
- add a participant number to the data file
- make a cursor centring trial (use html-button-response and timeline_variables)