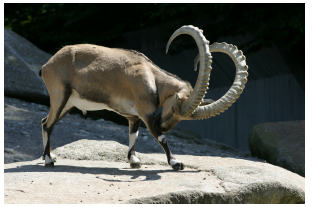
Experimental Linguistics with IBEX

A Walkthrough

2021-01-22

Presentation Outline



Introduction to IBEX

- Basic ideas
- Navigating the platform

Scripting an experiment

- Overview of critical components
- Code walkthrough

Data Analysis

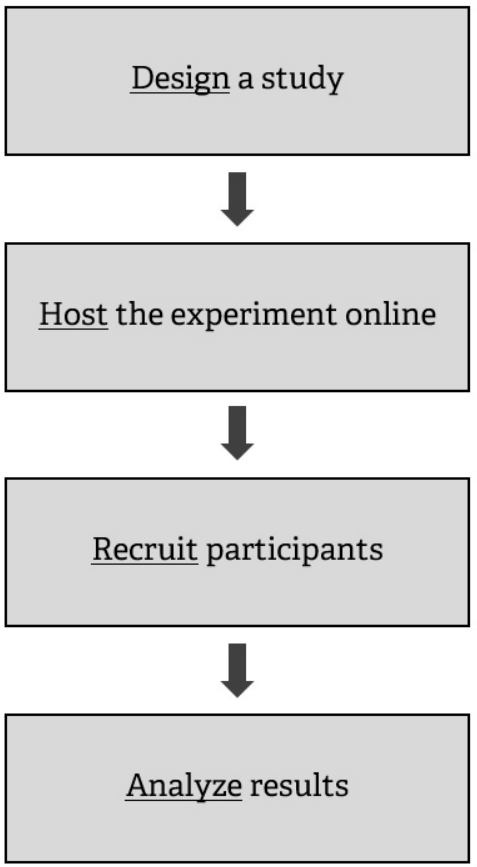
- Understanding the output format
- Importing into R

Introduction to IBEX

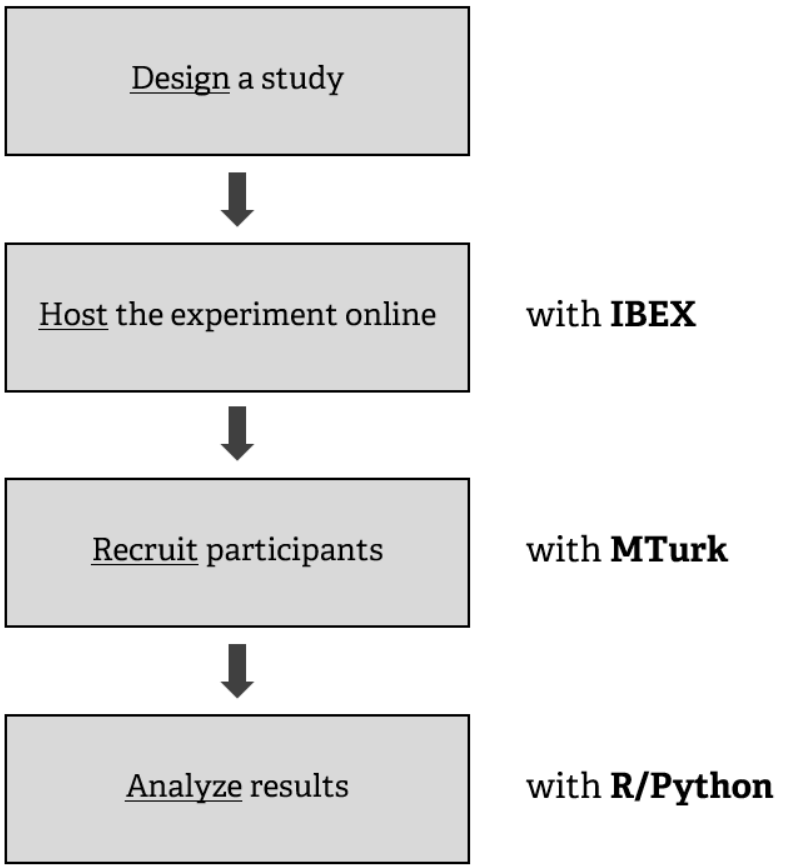
What is IBEX?

- Stands for (**I**)nternet-(**B**)ased (**EX**)periments
- **DOES**:
  - Host experiments with webpage links
  - Log user interactions
  - Store data on a secure server
- **DOES NOT**:
  - Recruit participants (*see* Amazon Mechanical Turk)
  - Provide an analysis of the results (*see* R, Python)

Where does IBEX fit?

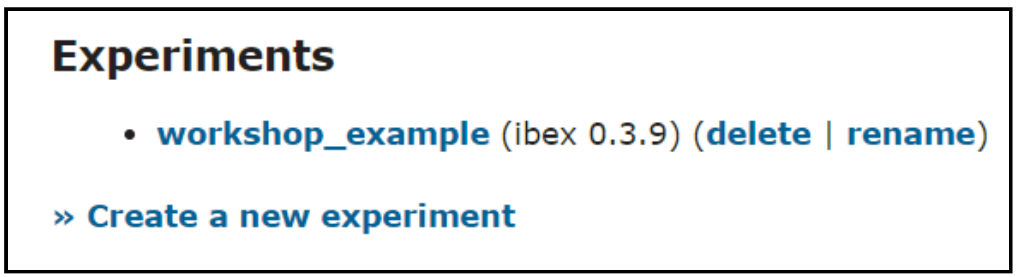


Where does IBEX fit?

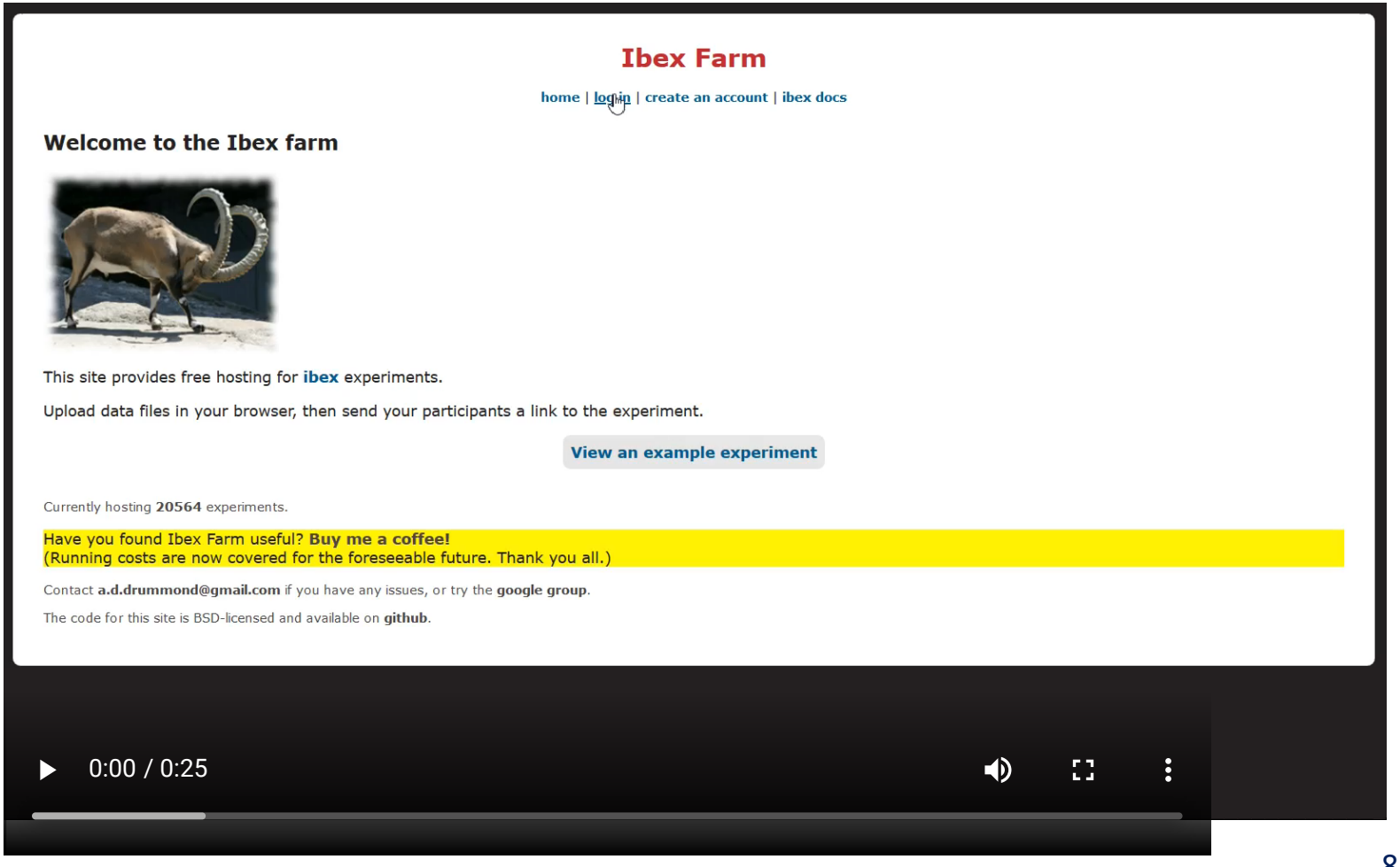


Navigating IBEX

- Go to https://spellout.net/ibexfarm
- Click **create an account** (or **log in**)
- Click **manage my experiments**
- Click **Create a new experiment**
- Give it a unique name like "*workshop_example*"
- Done!



Navigating IBEX



Experiment file structure

chunk_includes

- Stand-alone files go here

css_includes

- Style specifications go here

data_includes

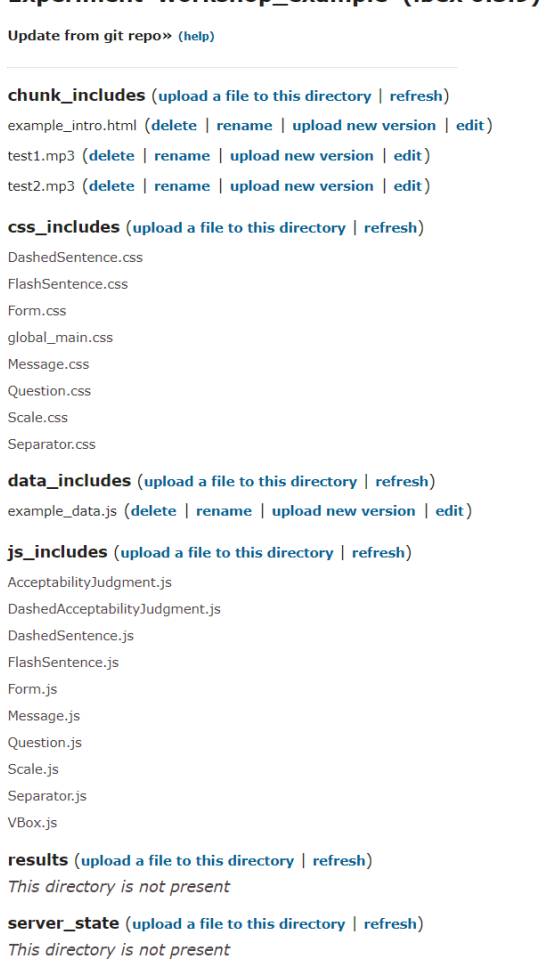
- Experiment scripts go here

js_includes

- Modules ("controllers") go here

results & **server_state**

- Automatically generated/updated



Experiment file structure

chunk_includes

- Stand-alone files go here

css_includes

- Style specifications go here

data_includes

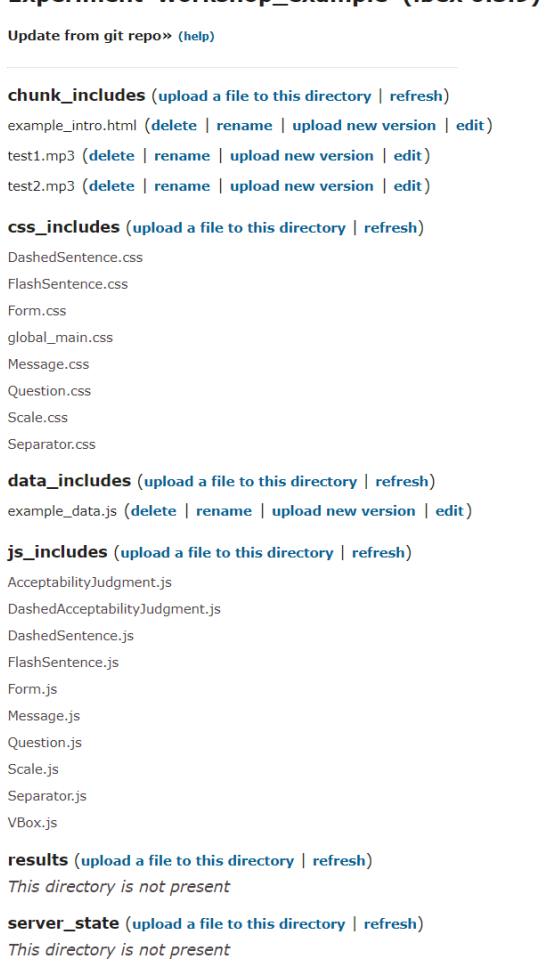
- Experiment scripts go here

js_includes

- Modules ("controllers") go here

results & **server_state**

- Automatically generated/updated



Scripting an experiment

The script

Only need to modify **one file**: *example_data.js* (can also be renamed later)

At creation, the default file looks like this:

```
var shuffleSequence = seq("intro", sepWith("sep", seq("practice", rshuffle("s1", "s2"))),
sepWith("sep", rshuffle("q1", "q2")));
var practiceItemTypes = ["practice"];

var defaults = [
    "Separator", {
        transfer: 1000,
        normalMessage: "Please wait for the next sentence.",
        errorMessage: "Wrong. Please wait for the next sentence."
    },
    "DashedSentence", {
        mode: "self-paced reading"
    },
    "AcceptabilityJudgment", {
        as: ["1", "2", "3", "4", "5", "6", "7"],
        presentAsScale: true,
        instructions: "Use number keys or click boxes to answer.",
        leftComment: "(Bad)", rightComment: "(Good)"
    },
```

The above script creates this self-paced reading experiment.

Writing your own script

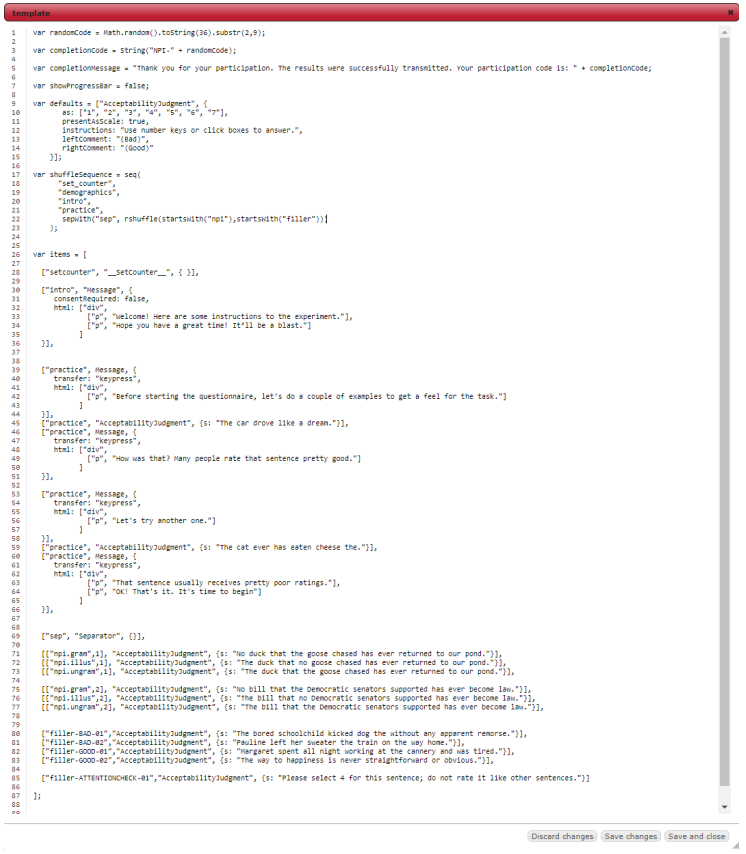
The *example_data.js* script works, but is not very friendly.

We'll use **our own** - **template.js** - to demonstrate how the script works.

```
// Options and Other Variables //

//// Generates random number assigned to participants (Participant ID)
var randomCode = Math.random().toString(36).substr(2,9)
//// A message to show to participants at completion (useful for confirmation, raffle entry, etc.)
var completionMessage = "Thank you for your participation. Your participation code is: " +
randomCode

//// Show a progress bar at the top? (true/false)
var showProgressBar = false

//// Override default settings for controllers (parameters go inside the curly braces { })
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
```

The above script creates this acceptability rating experiment.

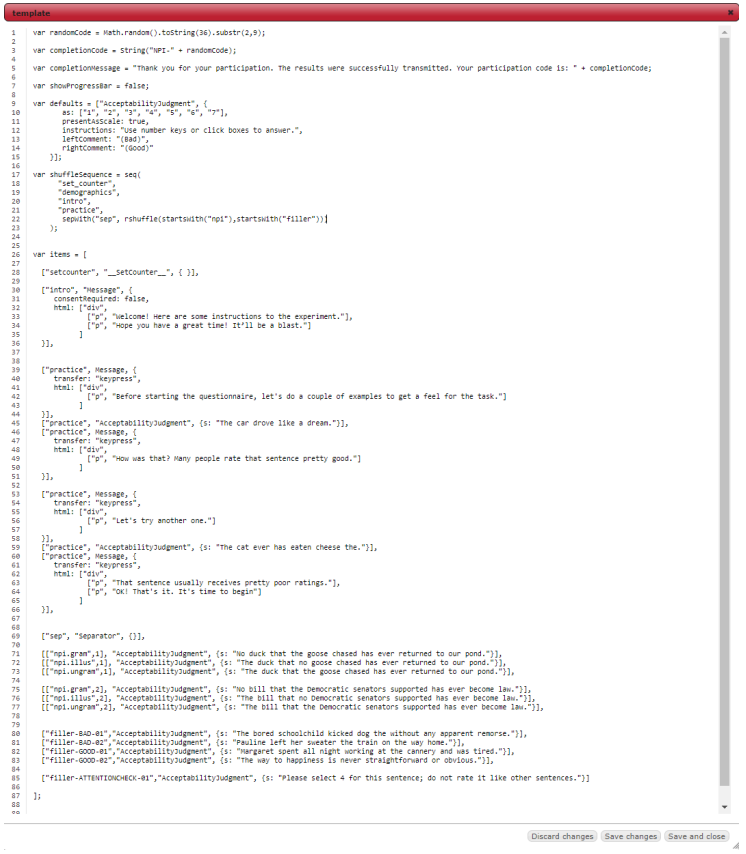
A simple layout



Editing the script

When you open the `.js` file in the `data_includes` section of your experiment on Ibex, it will open up a text editor.

A simple layout



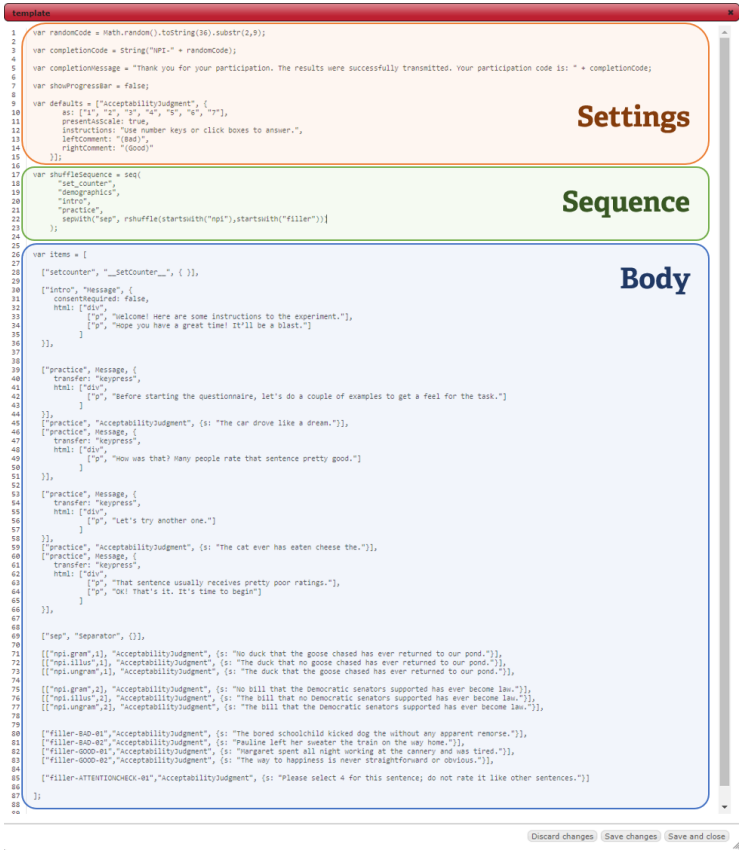
Editing the script

When you open the `.js` file in the `data_includes` section of your experiment on Ibex, it will open up a text editor.

Options for editing the file:

- Make changes directly on IBEX
- Download the file and open with an **editor** that supports **JavaScript** syntax checking (e.g., Atom)

A simple layout



Parts of the Script

Settings:

- Sets various options for the experiment

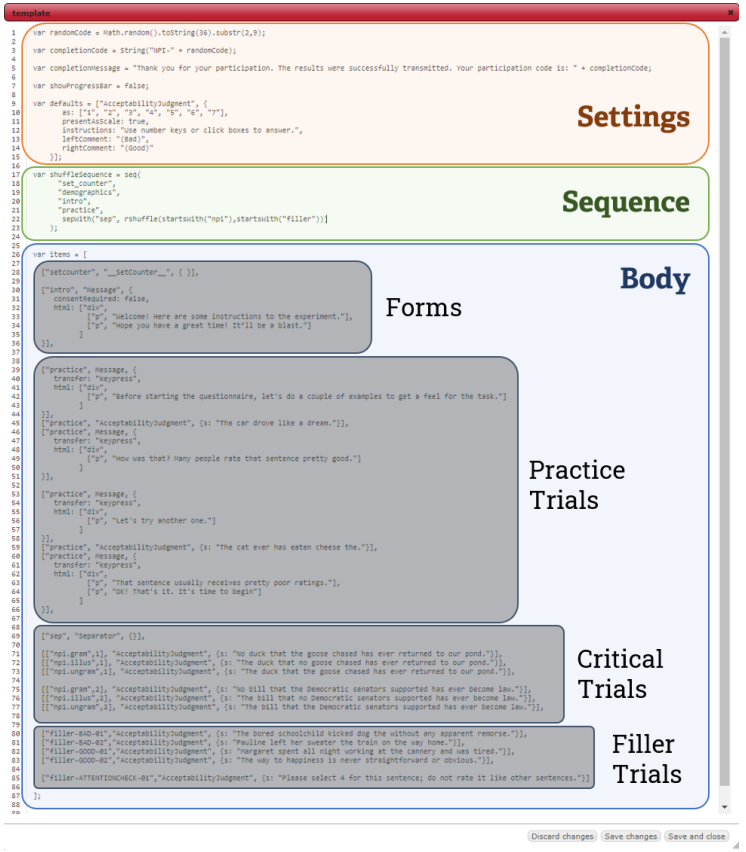
Sequence (`shuffleSequence`):

- Specifies the ordering of the different parts of the experiment

Body (`items`):

- Includes the actual material that will be shown to the participants

A simple layout



Body

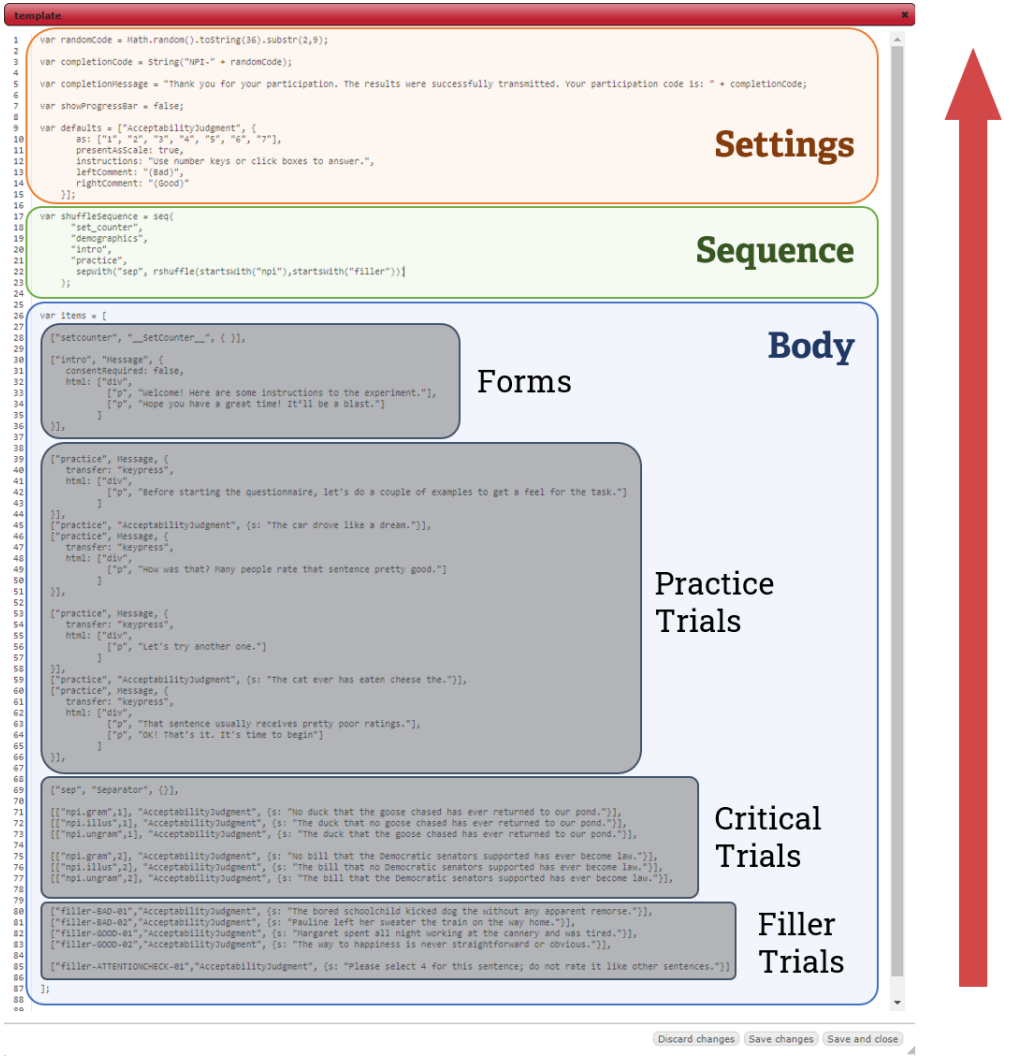
Forms

- Introduction page, consent form, directions, language background, demographic information, etc.

Trials

- All stimuli for the experiment
- *Practice*, *Critical*, *Filler*
- Can be accompanied by messages, questions, etc.

Walkthrough of the components



But first, the experiment design

Study: We are interested in how people recover from **garden-path sentences**.

"While Anna dressed the kitten paid attention."

... *[$_{VP}$ dressed the kitten], ...

... ✓[$_{VP}$ dressed], the kitten ...

Hypothesis: Verbs that are frequently **transitive** make recovery more difficult, compared to verbs that are frequently **intransitive**.

(transitive) - "While Anna ***trained*** the kitten paid attention."

(intransitive) - "While Anna ***dressed*** the kitten paid attention."

Prediction: Lower **acceptability ratings** in the transitive condition (***gp.trans***) than in the intransitive condition (***gp.intrans***).

Note: this is a *within-participant* design!

Trial Syntax

For each trial in our acceptability judgment experiment, we write this code:

[["*Trial name*", *Set #*], "*Trial Type*", {s: "*Sentence*"}]

This is a list (array) of three elements (clearer with spacing):

```
[
    ["Trial name", Set # ],
    "Trial Type",
    {s: "Sentence"}
]
```

```
[ ["Trial name", Set # ],
    "Trial Type",
    {s: "Sentence"} ]
```

We have one big bracket which contains **three elements**:

1. Another list, consisting of the *name of the trial* and *set number*
2. A string specifying the *type of the trial* (called **Controllers**)
3. A curly bracket (*braces*), which has "s" and the *sentence* separated by a colon

Translating the design to code

Example 1:

For the gp.trans condition, the first stimuli in our acceptability judgment experiment is the sentence "While Anna trained the kitten paid attention".

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

Translating the design to code

Example 1:

For the **gp.trans condition**, the first stimuli in our acceptability judgment experiment is the sentence "While Anna trained the kitten paid attention".

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

Translating the design to code

Example 1:

For the **gp.trans condition**, the **first stimuli** in our acceptability judgment experiment is the sentence "While Anna trained the kitten paid attention".

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

Translating the design to code

Example 1:

For the **gp.trans condition**, the **first stimuli** in our **acceptability judgment experiment** is the sentence "While Anna trained the kitten paid attention".

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

Translating the design to code

Example 1:

For the **gp.trans condition**, the **first stimuli** in our **acceptability judgment experiment** is the **sentence** "While Anna trained the kitten paid attention".

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

Translating the design to code

Example 1:

For the **gp.trans condition**, the **first stimuli** in our **acceptability judgment experiment** is the **sentence** **"While Anna trained the kitten paid attention"**.

```
[
    [ "gp.trans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna trained the kitten paid attention."}
]
```

This can be put into a single line:

```
[["gp.trans",1], "AcceptabilityJudgment", {s: "While Anna trained the k
```

Translating the design to code

Example 2:

For the **gp.intrans condition**, the **first stimuli** in our **acceptability judgment experiment** is the **sentence "While Anna dressed the kitten paid attention"**.

```
[
    [ "gp.intrans", 1 ],
    "AcceptabilityJudgment",
    {s: "While Anna dressed the kitten paid attention."}
]
```

This can be put into a single line:

```
[["gp.intrans",1], "AcceptabilityJudgment", {s: "While Anna dressed the
```

Practice

Item set #2:

(transitive) - "Since Dave improved the department was satisfied."

(intransitive) - "Since Dave worried the counselor devised a plan."

```
<code here>
```

Hover for answer

Putting together the stimuli

Wrap in `var items = []` and separate by a **comma**:

```
var item = [

  //// Set #1
  [["gp.trans",1], "AcceptabilityJudgment", {s: "While Anna trained the kitten paid attention."}],
  [["gp.intrans",1], "AcceptabilityJudgment", {s: "While Anna dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2], "AcceptabilityJudgment", {s: "Since Dave improved the department was
satisfied."}],
  [["gp.intrans",2], "AcceptabilityJudgment", {s: "Since Dave worried the counselor devised a
plan."}]

]
```

Good scripting habits:

- Grouping item sets together and separating sets with new line
- Adding comments (starts with two or more slashes `//`)
- Saving often! (editing in IBEX not recommended)

Practice and Fillers

Practice Trials:

- Presented at the beginning, accompanied by instructions and feedback

Filler Trials:

- Mixed in with the critical trials

For both types of trials, only minor tweaks are needed!

The *first element of the big bracket* can just be the *name of the stimuli*, since we **don't need to counterbalance** practice and filler trials.

```
Practice    Filler-good    Filler-bad    Filler-catch
```

Practice #1: "The car drove like a dream"

```
["practice-1",
 "AcceptabilityJudgment",
 {s: "The car drove like a dream."}]
```

Putting together all stimuli

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment", {s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1], "AcceptabilityJudgment", {s: "While Anna trained the kitten paid attention."}],
  [["gp.intrans",1], "AcceptabilityJudgment", {s: "While Anna dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2], "AcceptabilityJudgment", {s: "Since Dave improved the department was
satisfied."}],
  [["gp.intrans",2], "AcceptabilityJudgment", {s: "Since Dave worried the counselor devised a
plan."}],

  //// Fillers (Good)
  ["filler-good-01","AcceptabilityJudgment", {s: "When Harry fell, the audience was shocked."}],

  //// Fillers (Bad)
  ["filler-bad-01","AcceptabilityJudgment", {s: "When Tyler sneezed the driver, he passed a
tissue."}],

  //// Fillers (Catch)
  ["filler-catch-01", "AcceptabilityJudgment", {s: "Please select 4 for this sentence."}]
```

Important: The ordering of the trials here is just for human readability. We haven't yet told the program what order to present them in!

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

List the *names* of each trial in order:

```
"practice-1",
"gp.trans",
"gp.intrans",
"gp.trans",
"gp.intrans",
"filler-good-01",
"filler-bad-01",
"filler-catch-01"
```

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Wrap them in `seq()`:

```
seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Assign to `shuffleSequence`:

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Assign to `shuffleSequence`:

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

The `shuffleSequence` variable handles the *order of presentation* of the **experiment materials** that are stored inside the `items` variable.

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Problems

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

1. Lots of repetition ("-02", "-03", ...)

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Problems

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

1. Lots of repetition ("-02", "-03", ...)
2. Presentation of some trials should be random

Defining a Sequence

```
var item = [

  //// Practice
  ["practice-1", "AcceptabilityJudgment",
{s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1],
"AcceptabilityJudgment", {s: "While Anna
trained the kitten paid attention."}],
  [["gp.intrans",1],
"AcceptabilityJudgment", {s: "While Anna
dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2],
"AcceptabilityJudgment", {s: "Since Dave
improved the department was satisfied."}],
  [["gp.intrans",2],
"AcceptabilityJudgment", {s: "Since Dave
worried the counselor devised a plan."}],

  //// Fillers (Good)
  ["filler-good-
01","AcceptabilityJudgment", {s: "When
Harry fell, the audience was shocked."}],
```

Problems

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

1. Lots of repetition ("-02", "-03", ...)
2. Presentation of some trials should be random
3. How do we counterbalance critical trials?

Sequence: multiple selection

To save us from writing repetitive code, we use `startsWith()`

```
var shuffleSequence = seq(
 "practice-1",
 "gp.trans",
 "gp.intrans",
 "gp.trans",
 "gp.intrans",
 "filler-good-01",
 "filler-bad-01",
 "filler-catch-01"
)
```

```
var shuffleSequence = seq(
 startsWith("practice"),
 startsWith("gp")
 startsWith("filler")
)
```

The function `startsWith()` matches all names that starts with the given string.

Sequence: randomization

To mix critical and filler trials in random order, we use `rshuffle()`

```
var shuffleSequence = seq(
 startsWith("practice"),
 startsWith("gp")
 startsWith("filler")
)
```

```
var shuffleSequence = seq(
 startsWith("practice"),
 rshuffle(
    startsWith("gp"),
    startsWith("filler")
  )
)
```

By wrapping both the critical trials (*gp...*) and the filler trials (*filler...*) in `rshuffle()`, they are mixed together and presented in random order.

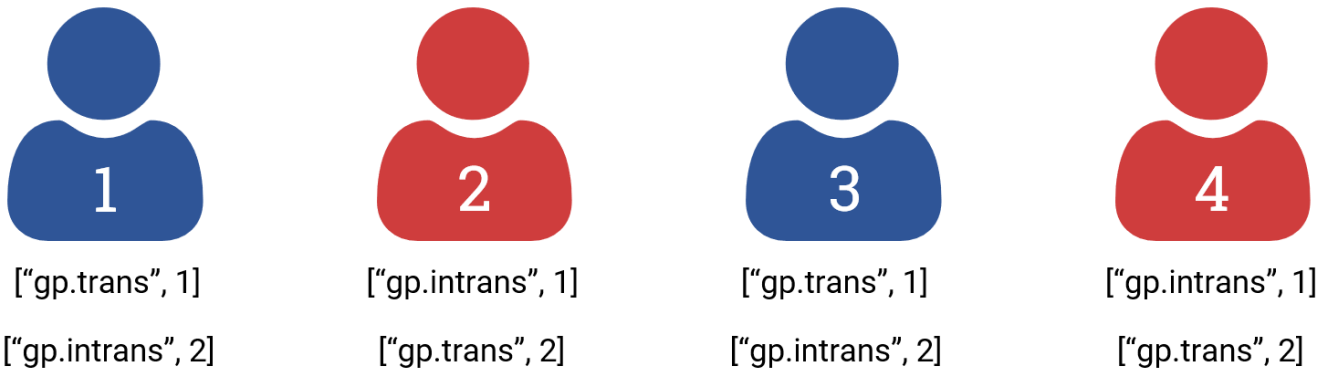
We can also write this out more compactly:

```
var shuffleSequence = seq(
 startsWith("practice"),
 rshuffle(startsWith("gp"), startsWith("filler"))
)
```

Sequence: counterbalancing

The **set number** in our critical trials automatically handles **counterbalancing**:

[["gp.trans", **1**], ...], [["gp.intrans", **1**], ...], [["gp.trans", **2**], ...], [["gp.intrans", **2**], ...]



We just need to add a **counter** inside `items` to track group assignment:

```
["setcounter", "__SetCounter__", { }]
```

Sequence: counterbalancing

Adding the counter to the experiment is simple:

```
var shuffleSequence = seq(
  "setcounter",
  ...
)

var items = [

  ["setcounter", "__SetCounter__", { }],
  ...

]
```

Note: Sometimes you want to place "setcounter" in the *middle* of the experiment

```
var shuffleSequence = seq(
  "intro"
  "consent",
  "setcounter",
  ...
)
```

Body and Sequence together

```
// Presentation Order //
var shuffleSequence = seq(
  "setcounter",
  startsWith("practice"),
  rshuffle(startsWith("gp"), startsWith("filler"))
)

// Experiment Materials //
var items = [

  //// Counter
  ["setcounter", "__SetCounter__", { }],

  //// Practice
  ["practice-1", "AcceptabilityJudgment", {s: "The car drove like a dream."}],

  // Critical Trials //

  //// Set #1
  [["gp.trans",1], "AcceptabilityJudgment", {s: "While Anna trained the kitten paid attention."}],
  [["gp.intrans",1], "AcceptabilityJudgment", {s: "While Anna dressed the kitten paid attention."}],

  //// Set #2
  [["gp.trans",2], "AcceptabilityJudgment", {s: "Since Dave improved the department was
satisfied."}],
  [["gp.intrans",2], "AcceptabilityJudgment", {s: "Since Dave worried the counselor devised a
```

Just need one more step: **Settings**

Settings (Basic)

Consist of miscellaneous options that we can put at the top of the script.

At the very least, we want to do two things:

1. Generate unique **participant IDs**
2. Specify the parameters for the **method design**

Settings (Basic)

Consist of miscellaneous options that we can put at the top of the script.

At the very least, we want to do two things:

1. Generate unique **participant IDs** (`randomCode`)
2. Specify the parameters for the **method design** (`defaults`)

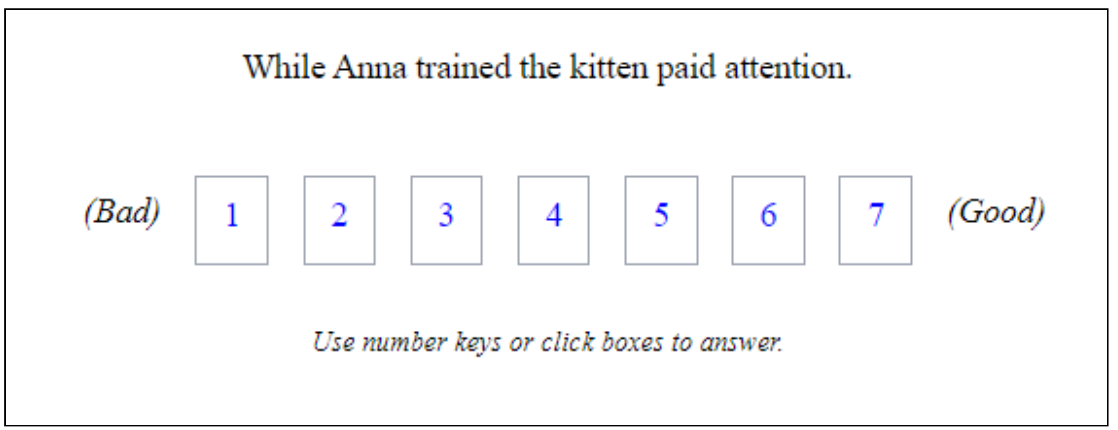
```
// Options and Other Variables //

//// Generates random number assigned to participants (Participant ID)
var randomCode = Math.random().toString(36).substr(2,9)

//// Override default settings for controllers (parameters go inside the curly braces { })
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]
```

More details in the **AcceptabilityJudgment** section of the documentation.

A minimal experiment

The specifications in `defaults` set the design of the trials:

```
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]
```



Settings (Miscellaneous)

You can also change other options, such as showing a message at the end:

`var completionMessage`

```
//// A message to show to participants at completion (useful for confirr
var completionMessage = "Thank you for your participation. Your particip
```

And whether to show a progress bar:

`var showProgressBar`

```
//// Show a progress bar at the top? (true/false)
var showProgressBar = false
```

You can learn more about these various elements in the **Miscellaneous options** section of the documentation.

A minimal experiment

We now have a minimally working experiment!

```
// Options and Other Variables //

//// Generates random number assigned to participants (Participant ID)
var randomCode = Math.random().toString(36).substr(2,9)
//// A message to show to participants at completion (useful for confirmation, raffle entry, etc.)
var completionMessage = "Thank you for your participation. Your participation code is: " +
randomCode

//// Show a progress bar at the top? (true/false)
var showProgressBar = false

//// Override default settings for controllers (parameters go inside the curly braces { })
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]


// Presentation Order //

var shuffleSequence = seq(
  "setcounter",
```

Interim Summary #1

What we've covered:

- We use a special syntax to create stimuli
- We store all the materials for our experiment inside `items`
- We specify the order of presentation inside `shuffleSequence`
- We set various options at the top of the script, such as defaults for the experiment *method* and the assignment of *participant ID*

A few more things to know:

- How can we add *plain text*?
  - Introduction page, consent form, feedback on practice trials, etc.
- How can we extend this workflow for *other experimental designs*?
  - Self-paced reading, comprehension tasks, etc.

The "Message" controller

A list of 3 elements, similar to the "AcceptabilityJudgement" items:

["*Trial name*", "Message", {html: *text*}]

- ***Trial name*** is used to reference the trial in sequencing, as seen earlier
- **"Message"** tells IBEX that this trial just shows text on a new screen
- Inside of the curly braces **{}** we can add text for the **html** parameter:

```
["intro", "Message", {html: ["p", "Welcome to the experiment!"]}]
```

Notes on **html**:

The code `["p", "<your text here>"]` prints a single paragraph of text.

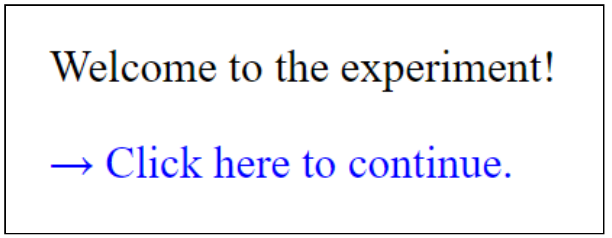
The "p" is called a **tags** and there are many others, but usually messages don't get more complicated than simple paragraphs.

Message examples

1-paragraph | n-paragraphs | consent | keypress | separator

A message composed of a single paragraph:

```
["intro", "Message", {html: ["p", "Welcome to the experiment!"]}]
```



Putting everything together

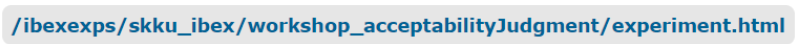
Here's a complete experiment with several **Message** controllers added.

```
// Options and Other Variables //

//// Generates random number assigned to participants (Participant ID)
var randomCode = Math.random().toString(36).substr(2,9)
//// A message to show to participants at completion (useful for confirmation, raffle entry, etc.)
var completionMessage = "Thank you for your participation. Your participation code is: " +
randomCode

//// Show a progress bar at the top? (true/false)
var showProgressBar = false

//// Override default settings for controllers (parameters go inside the curly braces { })
var defaults = [
    "AcceptabilityJudgment", {
       as: ["1", "2", "3", "4", "5", "6", "7"],
       presentAsScale: true,
       instructions: "Use number keys or click boxes to answer.",
       leftComment: "(Bad)",
       rightComment: "(Good)"
    }]


// Presentation Order //

var shuffleSequence = seq(
  "intro",
  "consent",
```

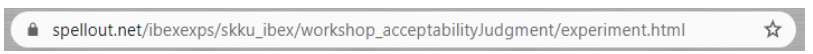
Getting the experiment up

So we have a new script, but how do we host the experiment?

1. Go to the IBEX website and log in.
2. Click on your experiment (or create one if you haven't already).
3. Click **edit** next to the *example_data.js* file in the **data_includes** section.
4. Delete its contents and copy paste your new code.
5. Click on the link at the top of the page.



The URL shown in your browser is the link to your experiment!



https://spellout.net/ibexexps/skku_ibex/workshop_minimal/experiment.html

Making a different experiment

Suppose that after a pilot experiment, we find acceptability judgments to be inappropriate for answering our research question.

We want a *finer-grained measure* of recovery difficulty, so we'd like to change the experiment to **self-paced reading** and look at reading time differences.

Given our existing template, we take the following steps:

1. Go to the documentation and find a Controller for self-paced reading.
2. Specify the design of that controller in the `defaults` variable.
3. Change our trials in `items` from "**acceptabilityJudgment**" to that Controller.
4. Make changes to the text of the "**Messages**" items (directions, feedback on practice, etc.).